Inaugural-Dissertation

zur Erlangung der Doktorwürde der

Naturwissenschaftlich-Mathematischen
Gesamtfakultät

der

Ruprecht-Karls-Universität
Heidelberg

vorgelegt von
Diplom-Informatiker

Christian Tobias Dencker

aus Karlsruhe

Tag der mündlichen Prüfung: ________________

Beyond Supervised Learning: Exploring Alternative Forms of Supervision for Visual Recognition

Advisor: Prof. Dr. Björn Ommer

Zusammenfassung

Der Durchbruch von Deep Learning hat das Feld der Computer Vision erheblich weiterentwickelt. Deep Learning, insbesondere in Kombination mit überwachtem Lernen, ist zum Kernstück der meisten Computer-Vision-Algorithmen geworden. Annotierte Daten sind jedoch oft der Flaschenhals datenhungriger Deep-Learning-Methoden, was ihre Leistung und breitere Anwendung einschränkt. Während die Annotation von Daten in großem Umfang teuer und zeitaufwendig ist, weiß der Mensch aus Erfahrung, dass Lernen auch mit begrenzter Überwachung möglich ist. Das Ziel dieser Arbeit ist es, Computer-Vision-Algorithmen zu untersuchen und zu entwickeln, die Deep Learning für Aufgaben mit begrenzter Überwachung ermöglichen.

Lernen ohne vollständige Überwachung erfordert die Nutzung alternativer Überwachungsquellen. Ein gängiger Ansatz besteht darin, Vorwissen zur betreffenden Aufgabe zu verwenden, um das Lernproblem einzuschränken und so den Mangel an Überwachung zu kompensieren. Dies ist bei Deep-Learning-Modellen schwierig, da hier ein End-to-End-Lernparadigma zum Einsatz kommt, das die vorherige Kontrolle über die vom Modell erlernte Repräsentation weitgehend verhindert. Stattdessen verfolgt diese Arbeit die Idee, das überwachte Lernen eines Deep-Learning-Modells in ein iteratives Verfahren einzubetten, das abwechselnd Überwachung aus alternativen Quellen generiert und das Modell verfeinert. Zu diesem Zweck werden Modelle und Lernverfahren im Kontext von zwei Computer-Vision-Anwendungen zur visuellen Erkennung mit verschiedenen Varianten der begrenzten Überwachung entwickelt und evaluiert.

Die erste Anwendung befasst sich mit dem Lernen von Repräsentationen für die Analyse menschlicher Posen aus unüberwachten Videodaten. Die visuelle Erkennung menschlicher Posen hat viele interessante Anwendungen, ist jedoch aufgrund der Vielfalt der Körperhaltungen und des Aussehens sowie des Problems des Selbstverdeckung eine Herausforderung. In dieser Arbeit wird ein selbstüberwachter Lernansatz verwendet, indem zwei Ersatzaufgaben entworfen werden, die ihre eigene Überwachung generieren, um aus raum-zeitlichen Informationen in Videos zu lernen. Um die Robustheit des Lernverfahrens zu erhöhen, nutzt der Ansatz die inhärente Selbstähnlichkeit menschlicher Bewegung über die Zeit zur Verfeinerung der generierte Überwachung und erstellt einen Lernplan, dessen Schwierigkeit sich schrittweise steigert. Das Verfahren erlernt eine umfassende Repräsentation der menschlichen Pose, die im Vergleich zu anderen aktuellen Verfahren in der Literatur eine konkurrenzfähige Leistung zeigt.

Die zweite Anwendung befasst sich mit der Herausforderung des Lesens von Keilschrift in antiken Tontafeln. Um Assyriologen bei ihrer Analyse zu unterstützen, wird ein schwach überwachter Lernansatz verfolgt, um einen Keilschriftdetektor zu erhalten, der Keilschriftzeichen lokalisieren und klassifizieren kann. Anstatt Tausende von Zeichenannotationen zu benötigen, lernt der Ansatz inkrementell, indem er abwechselnd überwachte Daten aus schwacher Überwachung generiert und einen Keilschriftdetektor trainiert. Um die Genauigkeit und die Trefferquote des Zeichendetektors zu verbessern, wird zur Überwachungsgenerierung eine Explorations- und Exploitationsstrategie eingesetzt, die zuverlässige und vielfältige Beispiele für das Lernen bereitstellt. Die Effektivität des vorgeschlagenen Ansatzes wird anhand des ersten großen Datensatzes zur Erkennung von Keilschriftzeichen, der im Rahmen dieser Arbeit erstellt wurde, gründlich evaluiert. Schließlich untersucht diese Arbeit einen Ansatz zur linguistischen Verfeinerung, um die Ergebnisse des trainierten Zeichendetektors weiter zu verbessern. Ein Textkorrekturmodell wird mittels Selbstüberwachung erlernt, wobei Bottom-up Informationen aus der Zeichenerkennung und Top-down Informationen aus der in Keilschrift kodierten Sprache kombiniert werden. Der Ansatz zeigt die ersten Schritte zur automatischen Transliteration von Keilschrift aus Bildern von Tontafeln.

Abstract

The rise of deep learning has significantly advanced the field of computer vision. Deep learning, especially in combination with supervised learning, has become the backbone of most computer vision algorithms. However, labeled data is often the bottleneck of data-hungry deep learning methods that limits their performance and broader application. While large-scale annotation is expensive and time-consuming, humans know from experience that learning with limited supervision is possible. This thesis aims to study and devise computer vision algorithms that enable deep learning for tasks with limited supervision.

Learning without full supervision requires exploiting alternative forms of supervision. A common approach is to use prior knowledge about the task to constrain the learning problem and compensate for the lack of supervision. This is difficult in the case of deep learning models, where an end-to-end learning paradigm largely prevents prior control over the representation learned by the model. Instead, this work pursues the idea of wrapping the supervised learning of a deep model into an iterative procedure that alternates between generating supervision from alternative sources and refining the model. To this end, models and learning procedures are developed and evaluated in the context of two computer vision applications for visual recognition with different variants of limited supervision.

The first application deals with representation learning for human pose analysis from unsupervised video data. Visual recognition of human pose has many interesting applications, but is challenging due to the high variability of pose and appearance as well as the problem of self-occlusion. This thesis employs a self-supervised learning approach by designing two auxiliary tasks that generate their own supervision to learn from spatiotemporal information in videos. To increase the robustness of the learning procedure, the approach exploits the inherent self-similarity of human motion for refining the generated supervision and creates a curriculum for learning that gradually increases in difficulty. It learns a meaningful representation of human pose that shows competitive performance to the state of the art.

The second application addresses the challenge of reading cuneiform script in age-old clay tablets. To support Assyriologists in their analysis, a weakly supervised approach to train a cuneiform sign detector is proposed that can locate and classify cuneiform signs. Rather than requiring thousands of sign annotations, the approach incrementally learns by alternating between generating supervised data from weak supervision and training a cuneiform sign detector. To improve the precision and recall of the sign detector, the supervision generation implements an exploration-exploitation strategy that produces reliable and diverse examples for learning. The effectiveness of the proposed approach is thoroughly evaluated on the first large-scale dataset for cuneiform sign detection which is established as part of this thesis. Finally, this thesis investigates an approach for linguistic refinement to further improve the results of the trained sign detector. A text correction model is learned in a self-supervised fashion that combines the bottom-up information from the sign detections and the top-down information from the language encoded in cuneiform script. The approach demonstrates the first steps towards the automatic transliteration of cuneiform clay tablets from images.

Acknowledgment

A work like this benefits greatly from the support of many people. With the following lines, I would like to express my special thanks to some of those who helped in its creation.

This thesis was conducted at Heidelberg Collaboratory for Image Processing (HCI) part of the Interdisciplinary Center for Scientific Computing (IWR) of Heidelberg University within the DFG Research Training Group 1653 "Spatio/Temporal Graphical Models and Applications in Image Analysis". I am grateful to Björn Ommer, head of the Computer Vision research group (CompVis) of HCI, for the valuable discussions, for advising me and giving me the opportunity to work in such an inspiring research environment. I would also like to thank Christoph Schnörr, my co-advisor, for reviewing the thesis, for his suggestions and encouragement.

Thanks to the people at HCI and especially to the entire CompVis group, who have helped to make things entertaining and stimulating, and who have often provided a welcome break from research. In particular, I would like to thank Nikolai for our many fruitful discussions, for his helpfulness and for the always welcoming atmosphere he created in our office. Next, I would like to thank Timo for spreading his good humor, providing advice and support, and simply listening. Many thanks to my former office mates Mattia, Robert, Franceso and Niko for maintaining a creative and diverse work space and for many memorable light-hearted moments. I would also like to thank Ömer for the valuable collaboration and his artistic talent, Nasim for his insights and inspiring discussion, Matthias and Jan for letting me hang out in their office and for their openness to non-convex stuff and formal advice, and Fabian and Barbara for introducing me to the best bike routes through the hills of Odenwald and for their encouragement. Many thanks go to the students working together with me at the CompVis group on several research projects, especially to Dominique and Johannes. Their interest and engagement added a great deal to the success of this thesis.

I would also like to thank Stefan Maul and his highly competent team at the Institute of Assyriology for the successful collaboration, for supporting me with their expertise, and for introducing me to the fascinating world of our ancient history.

To my family, my father Peter, my brother Daniel and all my friends, thank you for a lifetime of care and support. To Sabrina, thank you for your loving support and understanding throughout the ups and downs of my time as a Ph.D. student.

I dedicate this thesis to the memory of my mother Renate, who always believed in my ability to succeed, if I just give it a fair chance. Without your lasting belief and care throughout my life, this journey would not have been possible.

Heidelberg, January 2021 Tobias Dencker

In memory of my mother.

Contents

1 Introduction **1**

1.1 Making Sense of the Visual World 1

1.2 Computer Vision 2

1.3 Machine Learning is Essential 4

1.4 Computer Vision Algorithms for Limited Supervision 5

1.5 Objective of this Thesis 6

1.6 Current Challenges 7

1.6.1 Related to Machine Learning 7

1.6.2 Related to Visual Recognition 9

1.7 Contributions 10

1.8 Structure of this Thesis 11

2 Background of Computer Vision Algorithms for Limited Supervision **13**

2.1 Task as Problem Specification 14

2.1.1 Objective 14

2.1.2 Data 14

2.1.3 Prior Knowledge 15

2.2 Machine Learning Algorithm 16

2.2.1 Components 16

2.2.2 Development Phases 19

2.3 Deep Learning Method 20

2.3.1 Deep Learning Models 20

2.3.2 Optimization 21

2.3.3 Regularization 22

2.3.4 Network Architectures 23

2.4 Machine Learning Algorithms for Limited Supervision 26

2.4.1 Control the Space of Solutions 27

2.4.2 Generate Supervised Data 28

2.5 Background of Language Modeling 32

2.5.1 Language Model 32

2.5.2 n-gram Language Model 32

2.5.3 Tokenization 33

2.5.4 Word Representation . . . 33
2.5.5 Neural Network as Language Model . . . 34

3 Self-Supervised Human Pose Analysis 35
3.1 Introduction . . . 35
3.2 Related Work . . . 36
3.3 Approach . . . 38
3.3.1 Self-Supervised Pose Embeddings: Temporal Ordering and Spatial Placement . . . 38
3.3.2 Creating a Curriculum for Training . . . 39
3.3.3 Mining Repetitive Poses . . . 40
3.3.4 Network Architecture . . . 41
3.4 Experiments . . . 42
3.4.1 Training and Testing Details . . . 42
3.4.2 Curriculum Details . . . 43
3.4.3 Examples of Mined Repetitions . . . 44
3.4.4 Ablation Experiments on Posture Analysis in Olympic Sports Dataset 45
3.4.5 Pose Retrieval . . . 47
3.4.6 Pose Estimation . . . 49
3.5 Discussion . . . 51

4 Reading Cuneiform Script 53
4.1 Historical Background . . . 53
4.1.1 Writing System . . . 53
4.1.2 Development and Usage . . . 54
4.2 Scientific Study . . . 55
4.2.1 Assyriology . . . 55
4.2.2 Reading Comprehension . . . 55
4.3 Computer Vision Perspective . . . 57
4.3.1 Cuneiform Sign Detection . . . 58
4.3.2 Challenges of Cuneiform Sign Detection . . . 58
4.3.3 Potential Sources of Inductive Bias . . . 59

5 Building a Dataset for Cuneiform Sign Detection 61
5.1 Design Decisions . . . 61
5.1.1 Neo-Assyrian Cuneiform . . . 61
5.1.2 2D Images of 3D Clay Tablets . . . 62
5.1.3 Transliteration in place of Bounding Box Annotations . . . 62
5.2 Data Preparation . . . 62
5.2.1 Data Collection . . . 63
5.2.2 Data Preprocessing . . . 64
5.2.3 Data Cleaning . . . 66
5.3 Data Annotation . . . 68
5.4 Results . . . 70
5.5 Data Splits . . . 71
5.5.1 Cuneiform Sign Detection Dataset . . . 73
5.5.2 Legacy Dataset . . . 74

5.6 Data Understanding 74
5.6.1 Tablet Image Statistics 74
5.6.2 Sign Code Classes 75

6 Weakly Supervised Cuneiform Sign Detection 77
6.1 Introduction 77
6.2 Related Work 79
6.3 Approach 80
6.4 Methods 85
6.4.1 Iterative Training with Weak Supervision 85
6.4.2 Line Detection 86
6.4.3 Sign Placement Method 88
6.4.4 Sign Detector Training 89
6.4.5 Line-level Alignment 90
6.4.6 Sign-level Alignment 92
6.5 Results: Learning Approach 94
6.5.1 Evaluation Metrics 94
6.5.2 Training and Testing 95
6.5.3 Effect of Line-level Alignment 96
6.5.4 Effect of Alignment and Placement Method 96
6.5.5 Effect of Iterative Training 98
6.5.6 Effect of Available Annotations 100
6.5.7 Qualitative Detection Results with Visual Feature Activation 101
6.5.8 Apply Sign Detector to Different Cuneiform Script 104
6.6 Results: Sign Detector Performance 105
6.6.1 Class-wise Detection Results 105
6.6.2 False Positive Analysis 107
6.6.3 Similar Class Confusion Analysis 107
6.6.4 Individual Sign Detections 109
6.6.5 Detection Results on Tablet Images 110
6.6.6 Backbone Architecture Comparison 110
6.6.7 ImageNet Dataset Pre-training 113
6.6.8 Web Application of Cuneiform Sign Detection 113
6.7 Discussion 114

7 Linguistic Refinement of Cuneiform Sign Detection 117
7.1 Introduction 117
7.2 Related Work 119
7.3 Approach 120
7.3.1 Line Text Prediction 121
7.3.2 Line Text Correction 122
7.4 Experiments 125
7.4.1 Transliteration Dataset 125
7.4.2 Evaluation Metric 127
7.4.3 Line Text Prediction 128
7.4.4 Line Text Correction 133
7.5 Discussion 141

8 Conclusion 143

Bibliography 155

CHAPTER 1

Introduction

Humans have always been exploring the world, first relying on their own senses and only very recently with the help of technology. Many inventions, from microscopes to telescopes, have literally made the invisible visible and revealed new worlds, from the quantum to the astronomical scale. As innovation continues, technology is becoming ubiquitous and is solving increasingly complex tasks. To be more useful, it needs to become smarter by being more aware of its surroundings and making more informed decisions. It is clear that there is no one-size-fits-all solution, but that technology - similar to humans - needs to learn in order to adapt to different environments. While learning often involves a teacher providing manual supervision, there are also more autodidactic forms of learning that rely on alternative forms of supervision. In this work, we focus on the aspect of how machines can learn visual recognition, and we develop methods that are able to make sense of visual data with only minimal supervision.

1.1 Making Sense of the Visual World

While there are several human senses, *visual perception* is our preferred and most sophisticated sense for analyzing our environment [113]. It allows us to examine objects from a distance with high spatial precision (unlike touch and smell) which has been of great evolutionary utility. Because of its importance, the expansion of our scientific understanding of the world is closely linked to the expansion of our visual world [105]. Interestingly, the visual world does not stop at our retina, but extends into our mind's imagination. Our visual perception is a form of information processing that actively constructs a mental representation of reality from visual information. Thus, it deals with both *representing and processing of what we see* [163].

Essentially visual perception gives us a meaningful representation of the world that enables us to act efficiently within it. It provides not only a "pixel-wise" understanding of our environment, but also valuable high-level abstractions that are useful for general problem solving. Employing mental representations simplifies and speedups visual perception by replacing the full processing of our surroundings with selectively verifying our expectation (visualized in our mind) against reality. Likewise, it helps in understanding and communicating complex ideas: A picture is worth a thousand words, for example, by means of illustrations, visualizations, and plots. Human visual perception is primarily shaped by life on earth and represents an evolutionary trade-off between various factors such as speed, precision, and energy consumption. In addition to these evolutionary biases, our visual perception is also biased by cognitive constraints such as available attention, thoughts, associations, and emotions. Therefore, our mental representation sometimes fails to capture important details of physical reality. The existence of various optical illusions convincingly demonstrates possible limitations of our perception [163].

1.2 Computer Vision

Inspired by human visual perception, the field of *computer vision* aims to create algorithms that enable computers to make sense of the visual world. Computer vision is motivated by the promises of automation (e.g. robotics), super-human visual perception (e.g. for medical diagnosis), and artificial intelligence in general. Additionally, researchers hope to better understand the visual system and cognition by reengineering it.

Computer vision is sometimes described as a form of *"inverse computer graphics"* [94, 123]: While a computer graphics engine seeks to create an artificial photo-realistic world from a set of configuration parameters, computer vision attempts to identify those parameters in the physical world. The configuration parameters of the physical world are also called *factors of variation*, since they are responsible for the variability of objects and whole scenes. Given our prior knowledge about the physical world and its underlying mechanisms (our natural graphics engine), the factors of variation explain what we see at a more abstract level without having to explicitly count all photons. Even though the factors of variation are generally not directly observable, these latent factors constitute a *meaningful representation of our visual world.*

The *manifold hypothesis* provides a complementary intuition about the utility of representations for computer vision [83]. Since most natural data sources are analog signals (e.g. images), digital sensors and computers have to discretize the continuous signal for processing (e.g. raster graphics). To preserve most details (e.g. high resolution), the signal is usually converted into a high-dimensional vector in memory. The manifold hypothesis claims that if we would plot all the possible natural images as vectors in high-dimensional space, they would only occupy a small portion of the high-dimensional space due to the structural properties of the data-generation process (i.e. the underlying mechanisms of the physical world). The occupied space takes the form of lower-dimensional manifolds. Each manifold is a connected region which locally has the properties of Euclidean space and the interpolation of any two points of this region is on the manifold. The manifold hypothesis is supported by the following observation: Sampling a random gray-scale image (e.g. with 100x100 pixels where each pixel takes one of 256 shades of gray) will most likely result in an image of noise like an analog TV without signal. Only a small subset of all possible images (in this example 10000^{256}) represents images from the natural world. Instead of dealing with natural data in its high-dimensional form, working with the lower-dimensional manifold is more efficient and robust, as only the relevant factors of variation are considered. To achieve this, the creation of a representation that disentangles the factors of variation (that help explain the shape of the manifold) is an integral part of computer vision.

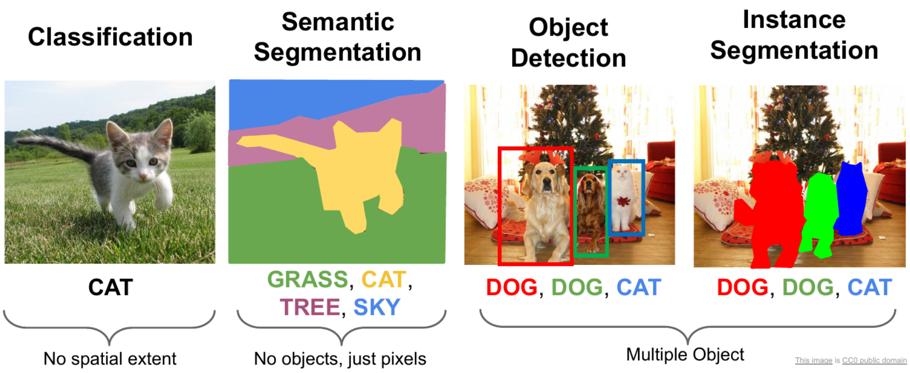


Figure 1.1: Computer vision tasks are categorized according to a set of core tasks. Four important core tasks related to visual recognition are visualized: (a) image classification assigns a label to an image; (b) semantic segmentation assigns a label to each pixel; (c) object detection identifies different instances of an object as labeled bounding boxes; (d) instance segmentation identifies different instances of an object and assign a label to the corresponding pixels. The tasks vary in their difficulty. For example, (a) requires no localization, (c) bounding boxes, (b) and (d) even a pixel-wise segmentation mask. [130]

From a more practical perspective, computer vision is usually concerned with answering specific questions about the visual world, such as ”Is there a pedestrian crossing the street?”. Most such questions can be decomposed into core tasks that are studied by computer vision. This thesis focus on core tasks related to visual recognition: Image classification, object detection, and semantic segmentation, as visualized in Fig. 1.1, are among the most important. To solve these tasks, a *computer vision algorithm* constitutes an information processing pipeline that at least includes *two steps*: First a *feature extraction* step (to obtain a meaningful representation) followed by a *decision making* step (to solve the task based on the representation). The selection of the *features* (the pieces of the representation) influences the subsequent decision making. In the worst case, a representation removes all information relevant to a computer vision task, impeding the decision making step. In the best case, a representation is so comprehensive that the solution of the task can be retrieved from it without further processing. A well-designed representation maps the high-dimensional image data to a fixed-size low-dimensional representation (i.e. dimensionality reduction) while retaining the information relevant for the decision making step.

In traditional computer vision, representations are based on simple visual features such as visual edges or corners in an image. The best feature combination for a specific task is determined in a time-consuming manual design process (handcrafted representation). The arrival of *deep learning*, a powerful machine learning method further described in Sect. 1.3, caused a paradigm shift in computer vision from handcrafting representations to learning representations from data [83]. *Representation learning* has several advantages: The best representation is the result of an optimization procedure that only requires data and is fast compared to manual feature engineering. The learned representations capture not only low-level but also high-level features (e.g. faces instead of edges). They form a hierarchy of features with multiple levels of abstraction, making them more semantically meaningful (deep) than handcrafted representations (shallow). Moreover, learned representations transfer well to similar computer vision tasks without additional training (e.g. dog representation used in cat classification). Thanks to off-the-shelf learned representations, it is possible to quickly achieve good performance for computer vision tasks in many domains. However, these learned representations are still not as expressive and general-purpose compared to what we are used to from our own experience in reasoning about the world.

1.3 Machine Learning is Essential

Machine learning is at the core of computer vision algorithms. Originally, machine learning in computer vision was mostly concerned with the decision making step (e.g. learning a classifier based on a representation). Deep learning has shifted the focus to representation learning for the feature extraction step. Recently, most computer vision algorithms are based on deep learning models that jointly learn feature extraction and decision making in the spirit of an *end-to-end learning* philosophy.

The idea of learning in computer vision is inspired by how infants and children learn to perceive the world. In children, it takes up to the age of five until the visual system is fully developed. While some low-level processing is hard-coded from birth, mid- to high-level visual perception (e.g. new categories in object detection) must be learned entirely from experience. As adults, we retain the ability to learn new concepts efficiently (e.g. from recognizing new yoga poses to studying new subjects at university).

Learning is defined in neuroscience as long-lasting change in behavior as effect of experience [234]. Machine learning approximates this definition by making the following assumptions: First, behavior can be expressed in the form of a *functional input-output relationship* with input $\boldsymbol{x}$ and desired output $f(\boldsymbol{x})$ that is *learned by a model* f. For example, $\boldsymbol{x}$ might represent an input image, $f(\boldsymbol{x})$ is supposed to be the label t of the object at the center of the image, and f an image classifier (i.e. a function that assigns a label to an image). Second, learning from experience can be simplified to a form of *deductive learning* (learning from general rules) or *inductive learning* (learning from examples). In deductive learning, the underlying rules governing a problem must be known a priori, whereas in inductive learning the rules are implicitly learned from examples (data). Because determining the rules of learning problems is often difficult (e.g. computer vision problems), most recent machine learning algorithms rely on inductive learning and automatically discover patterns in the data rather than depending on hard-coded rules.

Deep learning is a recent and very successful inductive machine learning method whose ability to learn powerful representations has had an profound impact on computer vision. It can approximate a wide variety of functional relationships [100] by learning a large parametric model f (often with millions of parameters) from data. Since f only needs to be differentiable, deep learning supports the construction of a flexible and scalable model that generally amounts to a composition of nonlinear functions stacked in multiple layers (thus deep). Each consecutive layer forms a more abstract representation of the input, so that the full model represents a hierarchical representation from low-level features (e.g. edges) at the bottom to high-level features (e.g. faces, parts of objects) at the top. Such deep models are also called *neural networks* or deep networks as their original construction was biologically motivated by the neural wiring of the human brain [74, 104, 180]. Despite their name, artificial neural networks are generally not considered realistic models of the brain [83]. Nevertheless, they contribute to the understanding of the brain and intelligence in many ways, e.g. by shifting the boundaries of what is considered possible in AI. Despite the large number of parameters, deep models can be successfully learned from data using stochastic gradient descent methods, which was initially surprising to the research community and is still far from being understood theoretically. The full power of deep learning is only realized at a certain scale, thus the dataset and deep learning model must be sufficiently large (e.g. image classification for ImageNet challenge with about a thousand examples per category). After the first successful applications of deep learning in the 1990s [127], its breakthrough did not occur until around 2010 [47, 121], when the computing power and the size of datasets reached the necessary scale.

A *machine learning algorithm* is characterized by three components: *data*, *model*, and *learning*. Learning is formulated as an optimization problem that searches for the best model given the data about the functional relationship between $\boldsymbol{x}$ and t. The goal is to learn a model on available data to perform well on previously unseen data, an ability known as *generalization*. Humans are great at generalization, often requiring very few examples to learn a new concept and easily exploiting connections (often abstract in nature) between known and unknown concepts to simplify learning [44].

In contrast, existing machine learning algorithms based on deep learning usually require large amounts of data with hundreds of examples for each new concept. Additionally, the transfer of learned models between related tasks is only possible to a limited degree and additional data is a prerequisite. Thus the performance of a machine learning algorithm strongly depends on the quality and quantity of the available data.

Data (for inductive learning) is distinguished according to the available *supervision* information. In the case of *supervised learning*, data usually takes the form of tuples of input and target $\{(\boldsymbol{x}, t)^N\}$ (where N is the number of data tuples), i.e. for each input $\boldsymbol{x}$ a target t is available that fully describes the input-output relationship of a particular instance. In the case of *unsupervised learning*, data only consists of inputs $\{(\boldsymbol{x})^N\}$.

1.4 Computer Vision Algorithms for Limited Supervision

Unfortunately, *data in the wild* rarely comes fully supervised. A computer vision task for visual recognition often deals with a large variety of object categories that need to be distinguished. The corresponding computer vision algorithm usually requires hundreds of annotations per category to learn a suitable model to solve the task. The supervision for learning is obtained by manual annotation which is time-consuming, expensive, and often requires expert knowledge. While unsupervised learning is desirable because it does not require supervision, unsupervised approaches are still outperformed by supervised ones in most visual recognition tasks [13]. Fortunately, data rarely comes fully unsupervised, as there is often some level of supervision available that can be exploited. This has given rise to research into alternative forms of learning such as *semi-supervised* (only part of the data is supervised), *weakly supervised* (data comes with weaker supervision than required), and *self-supervised learning* (data is unsupervised, but labels for a auxiliary learning task can be generated).

Motivated by the human ability for generalization, research in machine learning strives for a reduction of the strong data dependency, in particular paving the way for learning with limited supervision. In this thesis, we explore the space between supervised and unsupervised learning methods to build computer vision algorithms for applications with limited supervision. Of course, generalization is not only concerned with sample-efficient learning and research increasingly focuses on a wider range of topics that seem fundamental to our ability to generalize like meta-learning [101] (i.e. the ability to learn how to learn) and causality [170] (i.e. modeling the underlying mechanisms).

Central to learning with limited supervision is the concept of *inductive bias.* It corresponds to a set of assumptions that guide the machine learning algorithm in the search for the best model, independent of the available data. According to the no-free-lunch theorem for machine learning [226], inductive bias is a necessary condition for successful inductive learning in general. In the case of learning with limited supervision, the inductive bias can compensate for missing supervision by constraining the learning process.

The inductive bias shapes the model and learning components of a machine learning algorithm by integrating prior knowledge about the ”structure” of the task to be learned, i.e. the underlying data-generation process and the space of valid solutions [18]. When facing a computer vision problem with limited supervision, the inductive bias is integral. Computer vision algorithms often assume for objects in natural images to exhibit compositionality and equivariance to translation, i.e. that objects and scenes in images can be decomposed in ever smaller building blocks shared across the image (i.e. parts, parts of parts etc.) and that objects appear the same if shifted in space. Deep learning leverages compositionality by learning deep hierarchical representations where the description of the whole is distributed across features in multiple layers of abstraction. Convolutional neural networks, a class of deep learning models, exploit equivariance to translation by applying the same filters across image patches.

To integrate an inductive bias in a machine learning algorithm, one first needs to identify the relevant assumptions given the prior knowledge about the learning task. There exist many forms of inductive bias from very general to very task-specific. Then one has to revise the algorithm to incorporate these assumptions. In the context of deep learning, the inductive bias is usually realized in the architecture of the deep neural network model (architecture engineering) and in the design of the learning procedure. If the inductive bias can be explicitly built into an algorithm (e.g. in the model), it will have an higher impact compared to assumptions only expressed implicitly (e.g. soft regularization). Since expressing inductive bias in data-driven approaches like deep learning is often difficult, it is important to consider them jointly with more structural approaches like probabilistic graphical models. How to best encode task-specific assumptions in terms of inductive bias of a machine learning algorithm is subject to active research and central to this thesis.

1.5 Objective of this Thesis

The main objective of this thesis is to study and devise computer vision algorithms for applications with limited supervision. We aim to design learning procedures and models that make efficient use of alternative sources of supervision and turn prior knowledge about the problem structure into valuable inductive bias. To tackle applications with limited supervision, this thesis explores methods for self-supervised, semi-supervised, and weakly supervised learning of computer vision models that jointly learn meaningful representations and decision making. We investigate two distinct computer vision applications: *human pose analysis* (”What is the pose of this person?”) and *cuneiform sign detection* (”What signs are visible in the image of a cuneiform clay tablet?”).

In the case of human pose analysis, the goal is to learn a representation that captures the visual characteristics of human pose. A good representation of human pose serves as basis for the development of more specialized tasks like key point detection (identifying body parts) or pose retrieval (finding an image of a person in a similar pose). Instead of relying on the expensive labeling of thousands of images, this thesis explores how to learn such a representation in a self-supervised fashion by automatically analysing the dynamics of human pose in videos.

In the case of cuneiform sign detection, the goal is to automatically localize and classify cuneiform signs in images of age-old clay tablets. While the task is an instance of fine-grained object detection, it intersects with several specialized topics of computer vision, in particular optical character recognition, text spotting, handwritten text recognition, and scene text detection. Since prior work does not address the unique set of challenges posed by cuneiform sign detection, a novel approach is required. Instead of the extra manual annotation of ten thousands of cuneiform signs, this thesis explores how to obtain a cuneiform sign detector with weakly supervised learning that turns weak supervision without spatial information into full supervision by automatically inferring the exact location of signs. Since cuneiform signs represent written language, this thesis also investigates how to employ language modelling techniques from the field of natural language processing to improve cuneiform sign detection.

This thesis aims to leverage the power of deep learning despite limited supervision in both applications. To compensate for the missing supervision, we augment the learning process with inductive bias derived from alternative sources of supervision and prior knowledge. It is possible to encode some general assumptions about the computer vision task by choosing a deep model architecture. However, many specific assumptions cannot be encoded in the model explicitly, since deep models only learn representations implicitly. Another strategy is to leverage inductive bias to directly generate supervised data for learning. To this end, this thesis investigates self-supervised, weakly supervised and semi-supervised learning as three alternatives to plain supervised learning. These methods wrap the supervised learning process in a larger iterative procedure that automatically generates supervision for unsupervised or weakly supervised data. Since such generated supervision contains many errors which inhibit supervised learning, this thesis additionally aims to devise learning strategies and deep learning models that can cope with such noisy supervision.

1.6 Current Challenges

Deep learning has significantly advanced computer vision by learning powerful representations from data. In recent years, computer vision algorithms have moved from the lab to many less constrained real-world environments, such as on our smartphones. However, the ease, accuracy, and versatility of human perception still remains unmatched in many applications. Data is often the bottleneck and learning with limited supervision is still a challenge. The challenges addressed in this thesis can be broadly divided into two categories, one with challenges related to machine learning with limited supervision, the other with challenges related to the particular visual recognition task. In the following, we summarize the challenges of each category:

1.6.1 Challenges related to Machine Learning

The first set of challenges is related to the machine learning component of a computer vision algorithm which has to be adapted for tasks with limited supervision.

Model architecture that properly regularizes learning To improve the search of the best model for a particular computer vision task, the space of valid models needs to be constrained by the inductive bias (based on prior assumptions and the underlying machine learning method). In the case of deep learning, inductive bias is usually encoded by choosing the deep model architecture (space of valid models), as it controls the model's capacity and regularization during learning. The challenge is to find a deep architecture for a computer vision task that represents a good *bias-variance tradeoff* [73]: On the one hand, constraining the model flexibility reduces the risk of *overfitting* on the data, and increases the ability of sample-efficient learning which helps with limited supervision. On the other hand, increasing the model flexibility provides the capacity to deal with all kinds of variations in the input data.

Learning procedure that explicitly encodes inductive bias Since in many applications data only comes with limited supervision, supervised learning has to be replaced with weakly supervised, semi-supervised, or self-supervised learning depending on the degree of supervision. At their core these learning methods leverage inductive bias to generate artificial supervision. The challenge is to wrap supervised deep learning inside a learning procedure that encodes the inductive bias (e.g. underlying data-generation process) as explicitly as possible in order to compensate missing supervision. In the case of self-supervised learning, the challenge for the algorithm designer is to find suitable auxiliary tasks for which supervision can be easily generated from unsupervised data and which learn a meaningful representation for the actual computer vision task. In the case of weakly supervised learning, the challenge is to bridge the gap between weak and full supervision.

Learning with noisy generated supervision In contrast to manual annotations, generated supervision often contains a large percentage of errors (noise). Since the performance of deep learning algorithms suffers with increasing levels of noise, the challenge is to devise methods that reduce the impact of noise during learning, e.g. filter the errors in the generated supervision or control the level of noise in the generated supervision. Another problem is that generated supervision may be incomplete (e.g. only parts of an image are annotated) which inhibits performance of deep learning algorithms. Thus it is necessary to adapt supervised deep learning in order to cope with incomplete annotations.

Side effects of iterative learning The learning procedures for limited supervision often alternate between generating artificial supervision and re-train the deep learning model given this supervision. This iterative learning comes with its own challenges, since there is no guarantee that each step of the learning procedure improves results. Iterative learning may suffer from *drift*, i.e. an iterative degradation of performance due to a systematic error during learning. As drift is usually the product of interactions between several components of the learning procedure, it is challenging to identify and to deal with. Iterative learning is particular susceptible to *imbalanced supervision*, where the amount of supervised data per concept varies significantly. This may cause the learning to ignore the underrepresented concepts. Iterative learning must ensure that all concepts are learned. Some concepts are easier to learn than others independent of the amount of available supervision. Thus the learning procedure needs to be adapted to take this into account.

1.6.2 Challenges related to Visual Recognition

The second set of challenges is specific to the tasks of visual recognition associated with the considered computer vision applications.

Data source In the case of human pose analysis, the data for learning is based on videos that feature human motion dynamics. While videos are a rich data source, they come with their specific set of challenges. There are many degrees of freedom like camera motion, foreground motion, background motion, and illumination that change between frames of a video. Additional typical challenges in videos are occlusions and motion blur. Occlusions occur if parts of the object of interest are covered by other objects in the scene. In the context of human motion, we also observe self-occlusion, a special form of occlusion often found with articulated objects where part of the object blocks the view on other parts (e.g. a person holding a hand in front of the face). Motion blurr affects video frames if the motion in front of the camera is too fast so that it changes during exposure. Motion blurr artifacts can make the tracking of human pose during fast actions more challenging.

In the case of cuneiform sign detection, the data for learning is based on 2D images of 3D clay tablets. Since the writing resembles small depressions on a surface, the appearance changes depending on illumination. Occlusion is also an issue, as the writing can continue along the edges of a tablet which is not visible from a single viewpoint. The state of preservation of the clay tablets also poses a problem for detection.

Intra-class variance and inter-class similarity Computer vision tasks often require the visual recognition of a particular category of objects (object class), such as the class of paintings, chairs, or birds. Usually a computer vision model learns a representation that captures the characteristics of the class to distinguish it from other classes. The granularity of a class determines the degree of invariance to visual features that is required to solve the task. For example, a chair class might only contain chairs with four legs or it might contain all kinds of chairs independent of the number of legs.

When classifying objects, computer vision models must deal with intra-class variance and inter-class similarity. Intra-class variance means the variation inside an object class like in terms of appearance (e.g. red or blue chair) or geometry (e.g. 3-legged chair vs 4-legged chair). Finding a representation that captures all factors of variation of an object class is challenging. Inter-class similarity is related to the similarity of different object classes. The more similar two object classes are, the more difficult it is to separate them reliably. The complexity of the task increases with the number of classes to separate.

Fine-grained object detection is particularly concerned with the problem of inter-class similarity and intra-class variance. If two instances of the same category look more different than two instance of two different categories, this problem becomes particularly challenging like in the problem of cuneiform sign detection.

Learning invariance Learning a representation for a visual recognition task essentially involves learning invariance to a certain set of transformations: For example, changes in viewpoint (e.g. scaling, translation, rotation), changes in appearance (e.g. illumination, color), or changes in geometry (e.g. articulation, configuration). Finding a particular painting (exemplar search) requires invariance to a different set of transformations than finding any painting. Learning invariance requires generalization from few examples, since supervised data rarely covers the full range of transformations (e.g. a photo of a person usually shows the front, but not the back).

1.7 Contributions

The main contributions of this thesis are the following:

- A self-supervised learning strategy is proposed to obtain a meaningful representation for human pose analysis without requiring pose annotations. Representation learning is driven by two auxiliary tasks that generate their supervision from spatiotemporal relations in videos by assuming temporal coherence and spatial neighborhood.

- The robustness of the self-supervised approach against noisy generated supervision is increased by employing a curriculum-based learning strategy. The supervised data samples are organized in a curriculum that increases the difficulty of samples over the course of the learning process.

- The self-supervised approach is extended with a repetition mining method that identifies repetitive poses in videos by searching for self-similarity patterns across video frames. This method yields valuable supervised data points that are almost free of errors and mitigate side effects of iterative learning like drift.

- Since reading cuneiform script is difficult and time-consuming even for experts, a deep neural network-based sign detector is proposed that can localize and classify cuneiform signs directly in 2D images of clay tablets. The trained cuneiform sign detector is capable to deal with the fine-grained nature of cuneiform script. It detects over a hundred different sign code classes and demonstrates good performance on a large and diverse dataset.

- A weakly supervised learning approach is devised to train the cuneiform sign detector using limited supervision. Instead of requiring the extra manual annotation of ten thousands of cuneiform signs, the approach generates supervision for sign detector training using weak supervision from existing transliterations. It constitutes an iterative learning procedure that in each iteration alternates between training a sign detector and generating sign annotations by solving the alignment problem between tablet image and transliteration across hundreds of clay tablets in parallel.

- The quality and quantity of supervised data is increased by splitting the generation into two distinct steps: sign placement and image-transliteration alignment. This implements an exploration-exploitation strategy for data generation that mitigates the impact of noisy supervision and side effects of iterative learning.

- The release of a large-scale dataset for cuneiform sign detection in 2D images makes this challenging problem available to a broader research community. The dataset is the result of extensive data preparation and includes thousands of images of clay tablets with several thousand signs annotated by domain experts using a human-in-the-loop approach.

- The approach is made available with a web application that allows Assyriologist to easily utilize the trained sign detector. It is applicable to cuneiform scripts of different periods, paving the way for large-scale digital analysis in the field of Assyriology.

- A method for linguistic refinement of cuneiform sign detections is proposed that post-processes the sign detections to correct errors based on a language model that learns the language-related dependencies between individual characters of cuneiform script from data. Bottom-up visual information (sign detections) and top-down linguistic information (language model) are combined in a two-step approach that consists of line text prediction followed by text correction.

- The line text prediction offers a weakly supervised evaluation of the sign detector across the whole dataset for cuneiform sign detection independent of manual annotations. Based on this evaluation, the inherent error of the dataset for cuneiform sign detection can be estimated to draw more confident conclusions from the experimental results.

- The text correction model successfully learns an internal language model for cuneiform script that generalizes to unseen data samples on synthetic data. To train the model in the task of error correction, training samples are generated in a self-supervised fashion by artificially injecting noise in the lines of transliterated text. Special tokens are introduced to provide an additional means to integrate bottom-up information.

1.8 Structure of this Thesis

This thesis consists of two main parts, not accounting for the introduction, problem background, and conclusion. The first part (Chapter 3) proposes an approach for learning a representation of human pose in a self-supervised fashion to facilitate human pose analysis. The second and larger part (Chapter 4, Chapter 5, Chapter 6, and Chapter 7) introduces the problem of cuneiform sign detection as fundamental part of reading cuneiform script. Then it presents an approach for learning a cuneiform sign detector from weak supervision, and finally proposes an approach for linguistic refinement of sign detections based on a language model. In the following, a brief overview of the chapters is given:

Chapter 2 provides an overview of the construction of computer vision algorithms for limited supervision. The main components of a machine learning algorithm and the key design decision for modelling the inductive bias are introduced. The deep learning method and specialized deep architectures are presented. The various approaches for learning with limited supervision are summarized. Since in Chapter 7 language serves as an alternative form of supervision, we conclude with a overview of methods for language modeling.

Chapter 3 devises a method to learn a representation for human pose in a self-supervised fashion. The design of the auxiliary tasks for self-supervised learning is described and their implementation in terms of a Siamese network. Then the curriculum learning and repetition mining methods are introduced to facilitate the learning process. The quality of learned embeddings is demonstrated with experiments on several pose analysis datasets.

Chapter 4 gives an introduction to cuneiform script with information about its historical background and its scientific study today. The challenges of reading cuneiform script are presented from the perspective of Assyriology and their implications for a computer vision approach are discussed.

Chapter 5 describes the creation of the dataset for cuneiform sign detection. The process of data collection, preparation, and annotation is presented in detail. Then the dataset is formed with splits for training, validation and test. Finally, statistics of the new dataset are presented.

Chapter 6 describes an approach to learn a cuneiform sign detector using weak supervision from transliterations. The architecture of the sign detector is based on a deep convolutional neural network for object detection. The approach consists of three major components that are arranged as an iterative learning procedure that alternates between generating supervised data and training the detector. The components of the learning procedure are analysed in several ablation experiments and the quality of the learned sign detector is thoroughly evaluated, and impressive quantitative and qualitative results are presented on the newly created dataset.

Chapter 7 investigates how to boost the performance of a cuneiform sign detector with the integration of a language model for cuneiform script. A method for linguistic refinement is presented that post-processes sign detections in two steps to improve their quality. The components of the method are analyzed in multiple experiments. The line text prediction allows to quantify the inherent error in the dataset. The text correction model establishes a proof of concept of learning and integrating a language model for cuneiform script.

Chapter 8 concludes this thesis with a summary and final discussion of interesting findings regarding learning with limited supervision.

CHAPTER 2

Background of Computer Vision Algorithms for Limited Supervision

This thesis investigates the construction of computer vision algorithms for tasks with limited supervision. This chapter provides the background on the design of computer vision algorithms and presents the general strategies to deal with the problem of limited supervision. The considered algorithms leverage deep learning as the primary machine learning method to solve computer vision tasks for visual recognition. Thus the computer vision pipeline is represented by a deep learning model that is partially or completely learned from data. The computer vision tasks only come with limited supervision for learning. This poses a problem for algorithms based on deep learning, since they usually require large amounts of supervised data. To take advantage of supervised deep learning, the missing supervision needs to be obtained from alternative sources such as data with limited supervision and task-specific prior knowledge. Even though this information is not immediately useful for supervised learning, this chapter presents general learning approaches that can successfully exploit alternative forms of supervision for inductive learning.

In this chapter, we first characterize the computer vision task in terms of a task-specific objective, data, and prior knowledge. This provides an overview of all information available for learning. Second, the key components of a machine learning algorithm and the steps in its development are presented to provide a conceptual foundation of supervised learning. Third, an overview of the deep learning method is given that covers the general deep network model, its optimization and regularization, as well as specialized deep architectures, in particular for visual recognition and sequence processing. Finally, we consider the design of machine learning algorithms for limited supervision and provide a general framework for studying the various approaches for limited supervision. Since natural language can serve as an alternative form of supervision for many computer vision tasks, as is the case in Chapter 7, we conclude this chapter with an overview of language modeling methods.

2.1 Task as Problem Specification

In this thesis, we define a computer vision task in terms of its task-specific objective, data, and prior knowledge, which form a problem specification. It comprises a range of information about the task (from general to specialized) that is potentially useful for machine learning. In the following, we provide details on the three components that make up a task.

2.1.1 Objective

A computer vision task usually corresponds to a question about the visual world, whose answer is supposed to be automatically provided by a computer. In this thesis, we are particular interested in tasks related to visual recognition, e.g. answering discriminative questions like ”What is visible?” or ”Where is object XY?”. Fig. 1.1 shows four core computer vision tasks for visual recognition: image classification, semantic segmentation, object detection, and instance segmentation. In the context of machine learning, a task is usually specified in terms of input-output data tuples that serve as examples of how to solve particular instances of the task. Tasks naturally vary in their difficulty due to many factors, e.g. expected level of detail in the answer (”dog” vs ”husky), complexity of the question (”Is there a car?” vs ”Where is the person that is looking at the car next to the Porsche?”), or the data domain (nighttime vs. daytime pedestrian detection).

2.1.2 Data

The data of a task is associated with a data distribution P (i.e. data-generating process) which is usually not known and can only be sampled $(\boldsymbol{x}, t) \sim P$ (i.e. observed). A common data sample $(\boldsymbol{x}, t)$ consists of an input $\boldsymbol{x}$ and a target t, i.e. an example of the input to the task and the correct execution of the task. We also refer to the targets as *labels*, *annotations*, or more general *supervision* depending on the context. The data usually corresponds to a fixed set of data samples $\mathcal{D} = \{(\boldsymbol{x}_i, t_i) | i = 1 : N\}$ (e.g. images $\boldsymbol{x}$ with labels t) where N is the number of samples and $\mathcal{D}$ denominates the dataset. If the data comes with targets, then it is referred to as supervised data otherwise as unsupervised data. In the supervised case, each data sample represents input-output tuple $(\boldsymbol{x}, t)$, in the unsupervised case, only an input $(\boldsymbol{x})$. Supervision for visual recognition mostly corresponds to annotations of the image. The difficulty of a task is significantly determined by the quality and quantity of its data including supervision. For example, low-resolution images or ambiguous labels may increase the difficulty of the learning task.

Degree of Supervision

Supervision comes in different degrees that are task-specific. Besides the supervised case, the literature [111] distinguishes three different cases of *limited supervision*: unsupervised, weakly supervised, and semi-supervised.

- *Supervised data* (full supervision) refers to the case where supervision is at least at the level of granularity required by the task. Examples of full supervision would be category labels in case of image classification, or bounding box annotations in case of object detection.

- *Weakly supervised data* (weak supervision) refers to the case where supervision is more coarse than required by the task. Examples of weak supervision would be category labels in the case of object detection, or bounding box annotations in case of instance segmentation.

- *Unsupervised data* (no supervision) refers to the case where the task needs to be learned without any annotations (only the inputs are provided).

- *Semi-supervised data* (full supervision only in subset) refers to a special case where full supervision is only available for a small subset of the data.

Weak Supervision Weak supervision provides indirect or coarse information per data sample. It can come from a wide range of information sources depending on the task. Weak supervision corresponds to a hint rather than a full answer to a particular question (i.e. particular data sample). Examples of weak supervision used in literature for visual object detection which is commonly supervised by bounding boxes (rectangular boxes drawn around an object) are: image level labels, image captions, object extreme points, foreground segmentation mask, clicks or scribbles inside object, or eye gaze.

2.1.3 Prior Knowledge

Task-specific prior knowledge provides another source of information about a task. Prior information is different from weak supervision, as it refers to general beliefs about the task independent of individual data samples. In the following we provide a list of priors often used in literature for computer vision tasks [4, 67, 83, 195]:

- objectness: likelihood of box containing a generic object

- pairwise similarity: tracking object instances across images or video frames

- compositionality: checking for characteristic structure and parts of an object

- symmetry: check for symmetry

- smoothness: lack of smoothness (e.g. high contrast) often indicates object boundaries

- mutual exclusion: exclude improbable scenarios

- scale/size: the expected size of an object relative to other objects

- shape: the generic shape of an object class

- number of instances: typical number of instances to be expected in an image

- location: typical places object occur in an image

- class distribution: likelihood of encountering an object of a specific class

2.2 Machine Learning Algorithm

A machine learning algorithm is designed to learn how to solve a task from examples. It achieves this by learning a model from data that generalizes to unseen data points of a given task. A machine learning algorithm is not a general problem solver but requires prior assumptions about a task. According to the ”no free lunch” theorem [226] all machine learning algorithms show equal performance if averaged over the data distributions of all possible tasks. Only if we make assumptions about a particular task (by means of task-specific prior knowledge) and select a suitable algorithm that incorporates these assumptions, there is a chance of obtaining a better than average algorithm for the task.

The set of assumptions that is incorporated into the machine learning algorithm is called inductive bias [154]. As mentioned in Chapter 1 most of machine learning is based on induction. Unlike deduction the use of induction for learning is risky by design, since we never observe the full data distribution (only single snapshots), but ”interpolate” from a few examples. The idea of inductive bias is to leverage task-specific knowledge in order to make this induction more reliable (almost like a deduction). In the following we first explain the key components of a machine learning algorithm and then present the general steps in its construction.

2.2.1 Components

A machine learning algorithm consists of three components: *model*, *data*, and *learning* as visualized in Fig. 2.1. The machine learning algorithm we are interested in performs a function approximation to solve a particular task. The algorithm needs to learn a model f that approximates a mapping from inputs $\boldsymbol{x}$ to desired outputs $f(\boldsymbol{x})$. We assume in this section, that the data of the task is supervised and thus the desired output $f(\boldsymbol{x})$ is determined by the labels t in the data, i.e. $\mathcal{D} = \{(\boldsymbol{x}_i, t_i) | i = 1 : N\}$ with inputs $\boldsymbol{x}$, labels t, and N data samples. The components are described in the following:

Data

Data is essential as it provides most of the information (i.e. examples) for inductive learning. It is common practice to split the data $\mathcal{D}$ provided by the task in three parts ($\mathcal{D}_{train}$, $\mathcal{D}_{val}$, and $\mathcal{D}_{test}$) which are usually referred to *training*, *validation*, and *test* dataset. The model is trained on the data samples of $\mathcal{D}_{train}$, the algorithm design and its hyperparameters are tuned on $\mathcal{D}_{val}$, and the final test performance of the model is determined on $\mathcal{D}_{test}$. Since the ground truth data distribution P is only observed through the samples of $\mathcal{D}$, each data split should be selected sufficiently large to be representative. While the training data is usually selected as large as possible in order to retain most examples for training, the validation and test data are selected as large as necessary for proper evaluation.

Model

A machine learning model is designed with respect to a particular task. The task defines a functional input-output relationship that the model approximates in terms of a function $f_\theta : \boldsymbol{x} \mapsto y$ with model parameters $\theta \in \Theta$, input $\boldsymbol{x}$, and output y. An important step in the design of a model is the selection of the *model family*, i.e. a family of functions to choose f from. Deep learning models or graphical models are two popular model families. The

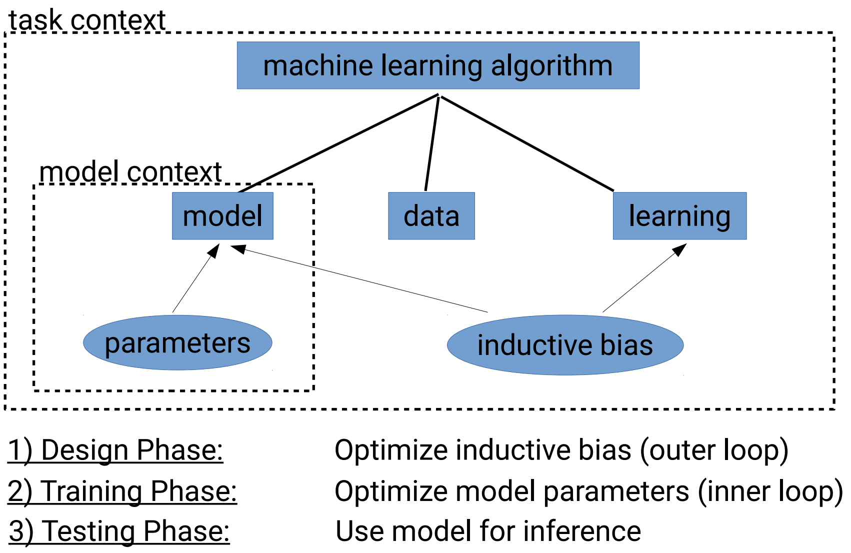


Figure 2.1: Machine learning algorithms consist of model, data and learning components. The goal is to solve a specific task by using a trained model for inference (testing phase). A good model is the result of optimizing its parameters according to some objective function on the available training data (training phase). The inductive bias incorporates prior knowledge about the problem in terms of model selection, hyperparameters and learning procedure which is essential for a good machine learning algorithm (design phase). To assess the quality of the design phase, the model needs to be trained and evaluated in the training phase. Thus the training phase can be considered as a inner loop nested in the outer loop of the design phase.

selection of the model family implies a specific machine learning method with characteristic model architectures and optimization methods for learning. Selecting the *model architecture* (i.e. model f and the set of all its possible parameter configurations Θ) defines the set of learnable functions $\{f_\theta : \boldsymbol{x} \mapsto y \mid \forall \theta \in \Theta\}$, which is also called the hypothesis space or solution space of the model. Since searching for the optimal model architecture is often not a tractable problem in machine learning, best practices and heuristics play an important role in the mostly manual design process. A smaller solution space can simplify the learning of the model, however, it can also impact its ability to generalize.

Learning

Given task-specific data and model, the learning component is formulated as an optimization problem with the goal of finding the *model parameters* θ of *model* f_θ that generalizes best to unseen data sampled from data distribution P associated with the task. We describe the case of supervised learning, where the data includes annotations. Supervised learning also provides the foundation for learning with limited supervision described in Sect. 2.4. The optimization associated with learning is often referred to as *model training* in the machine learning literature. Training the model means determining the model parameters θ_* that minimize a general loss function $\mathcal{L} : \Theta \to \mathbb{R}$ as follows:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta). \tag{2.1}$$

In statistical learning theory [73], the general loss function takes the form of the expected risk:

$$\mathcal{L} = \mathbb{E}_{(\boldsymbol{x},t)\sim P}\Big[E\big(f_\theta(\boldsymbol{x}), t\big)\Big], \tag{2.2}$$

where E is an *error function* that rates the quality of the model prediction $f_\theta(\boldsymbol{x})$ with respect to the label t. The error function is often called by different names like cost function, loss function, or generally objective function, since it relates to the objective of the task.

Minimizing the expected risk directly is not possible, as the ground truth data distribution P is not known. Instead, the minimization of the expected risk is approximated by minimizing the empirical risk $\hat{\mathcal{L}}$ based on the training data $\mathcal{D}_{train}$ sampled from P:

$$\arg\min_\theta \frac{1}{|\mathcal{D}_{train}|} \sum_{(\boldsymbol{x}_i,t_i)\in\mathcal{D}_{train}} E\big(f_\theta(\boldsymbol{x}_i), t_i\big) \tag{2.3}$$

The model parameters θ are obtained by minimizing the empirical risk $\hat{\mathcal{L}}$ across $\mathcal{D}_{train}$. Since $\hat{\mathcal{L}}$ is only an approximation of $\mathcal{L}$, the best parameters are not necessarily the ones that let the model best fit the samples of $\mathcal{D}_{train}$. The best parameters are the ones that let the model best perform on the unseen data sampled from P, i.e. induce the best generalization. To obtain a better measure for the generalization, the trained model is evaluated in terms of the empirical risk based on $\mathcal{D}_{test}$ whose data samples have not been observed during training. If the model is not able to fit $\mathcal{D}_{train}$ and produces a high train error (empirical risk on the train set), this is called *underfitting*. If the gap between the train error and the test error (empirical risk on the test set) is too large, this is called *overfitting*. The generalization ability of a machine learning algorithm depends critically on the simultaneous reduction of overfitting and underfitting. The two challenges along with the need for inductive bias mentioned earlier are summarized in the *bias-variance trade-off* [73, 83], the fundamental design problem of machine learning. Given a particular task, the algorithm design must find the right balance between bias and variance to achieve the best performance.

Closed-form optimization may be possible for learning depending on the choice of model family and error function. In the case of nonlinear models or very large datasets (due to computational reasons), iterative optimization procedures like gradient descent are used.

Inductive Bias

Inductive bias describes a set of assumptions about a task (independent of data samples) that are essential in the design of a machine learning algorithm in order to learn a model that generalizes to unseen data points. The modelling of inductive bias relies on the task-specific prior knowledge described in Sect. 2.1.3. Assumptions fall broadly into two categories: First, assumptions regarding the data-generating process (what do tuples of input and label look like?), and second, assumptions regarding the space of solutions (what do models look like that solve the task?). These assumptions are translated into inductive bias by guiding the key design decisions of the machine learning algorithm that affect the design of the model and the learning procedure. The following elements are critical to the implementation of the inductive bias of a machine learning algorithm [83]:

- **The selected model family:** The choice of model family influences the design of the model and learning procedure. A model family is usually associated with a particular machine learning method (e.g. deep learning) which may dictate the usage of specific model architectures, optimization procedures, and regularization techniques. Furthermore, the model family also determines the degree to which the representation of a model can be shaped prior to learning.

- **The model:** The design of the model (i.e. model architecture) determines the space of solutions from which the model is selected. If the model family allows for explicit modelling of the representation, a strong structural prior on the model can be implemented. Besides explicit modelling, the architecture of a model can regularize the learning process significantly. Similarly, the *model capacity*, the model's ability to fit variety of functions, is an important hyperparameter to control.

- **The optimization procedure:** The optimization procedure is partly determined by the choice of the model family. In the case of iterative optimization, the quality of the trained model is strongly influenced by the optimization method and the choice of its hyperarameters.

- **The error function:** The error function combines the prediction of the model with the supervision to create a training signal for optimization. Thus it is essential that the error function is aligned with the objective of the task to guide learning properly.

- **The regularization techniques:** The regularization techniques comprise methods that are supplementary to the learning procedure associated with the selected model family and that improve the model's ability to generalize (i.e. that improve the test error). Regularization techniques usually influence the model and learning procedure by implementing an inductive bias that prioritizes some solution over others. Usually several techniques are combined depending on the task. A large selection of regularization techniques exists in the literature [122].

2.2.2 Development Phases

The development of machine learning algorithms iterates over three different phases:

1. *Design phase*: Designing the model and learning procedure of a machine learning algorithm to solve particular task as visualized in Fig. 2.2. The modelling of inductive bias allows to incorporate task-specific prior knowledge in both components which is essential for learning. Additionally a machine learning algorithm is governed by a set of hyperparameters that are also configured at this stage. Outputs the learning procedure, model, hyperparameters.

2. *Training phase*: A model is trained by optimizing its parameters according to a objective function on the training data. Outputs the trained model parameters.

3. *Testing phase*: The performance of the trained model is evaluated by applying it to the test data. Outputs the predictions from model inference.

The design and training phase are usually tightly coupled. To assess the quality of the design phase, the model needs to be trained and evaluated in the training phase. We can think about the training phase as a nested inner loop inside the outer loop of the design phase as shown in Fig. 2.1: During development, the algorithm design and the

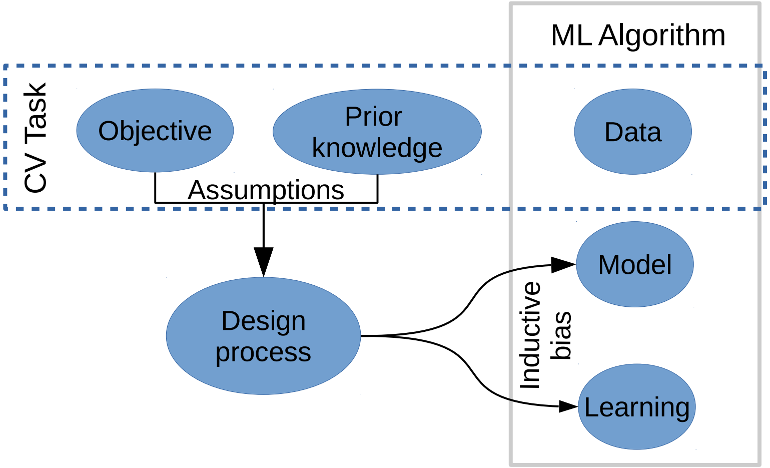


Figure 2.2: Overview of the design process: A computer vision algorithm is based on a machine learning algorithm (ML Algorithm) and designed according to the specifications of a computer vision task (CV Task). The task-specific objective and prior knowledge are manually turned into a task-specific model and learning procedure (inductive bias) in order to facilitate inductive learning from the task-specific data.

hyperparameters are optimized in the outer loop, while model parameters are repeatedly optimized in the inner loop.

While the design phase is often a manual process, part of it or sometimes the whole phase can be subject to automatic optimization. Related research topics are hyperparameter optimization [194], model selection, or more recently automated machine learning [93]. Similarly, meta-learning [70, 189] is looking for ways to learn how to learn, i.e. how to solve not only a single problem set, but transfer acquired problem-solving skills to new unseen problem sets.

2.3 Deep Learning Method

Deep learning is a powerful machine learning method that provides a framework for building and learning models of various complexity. At the core of deep learning is the approximation of functions with a large parametric model that consists of layers stacked together in a deep network. First, we describe the general structure of deep learning models and the design decisions in their construction. Then, we explain how deep learning models are optimized and which regularization techniques are commonly employed. Finally, we present general architectures of deep networks to deal with tasks involving images or temporal sequences. This includes specialized architectures for solving some of the core computer vision tasks encountered in this thesis.

2.3.1 Deep Learning Models

A deep learning model is referred to by various names such as deep neural network, Multi-Layer Perceptron (MLP), or simply deep network. A *deep network* is usually composed of a long chain of nonlinear functions, where the length of the chain represents the depth of the network (thus deep). The composition is often visualized as a stack with the input of the network at the bottom, the output at the top, and the nonlinear functions as intermediate layers (thus layers). The structure of the composition forms a direct acyclic graph (thus network). The functions used in the network are mostly nonlinear, because a network of linear functions can be simplified to a single linear function, which would defy the goal of learning deep representations [188]. Usually, the layer of a network consists of a linear mapping combined with a *nonlinearity* that presents a form of threshold function (inspired by the activation threshold of biological neurons). The nonlinearities are an important factor in the design of deep networks, as they shape the flow information through the network during inference and learning. While nonlinearities are an active area of research, the Rectified Linear Unit (ReLU) [83] is currently one of the most used nonlinearity.

A deep network forms a large parametric model, where the model parameters, also called *network weights*, are distributed over the different layers of the network. Due to the many degrees of freedom in the design, deep architectures are usually constructed from layer-wise building blocks that follow standardized design patterns. Moreover, a lot of empirical findings exist that guide the design of deep networks, e.g. deeper networks need fewer parameters and generalize better, but are considered more difficult to train [90]. Since the final representation of a deep network depends on the training on data, the modelling of the representation is mostly implicit. Besides the depth, we describe more design decision regarding the network architecture in Sect. 2.3.3 and introduce specialized architectures in Sect. 2.3.4.

2.3.2 Optimization

The training of deep networks is performed by an iterative optimization method, since the nonlinearity of deep networks causes most loss functions to become non-convex (no closed-form solutions). Usually a form of *Stochastic Gradient Descent* (SGD) [186] is employed that updates the network weights using the following update rule in each iteration:

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \mathcal{L}(\theta_t, d_t), \tag{2.4}$$

where η_t is the learning rate and $\nabla_\theta \mathcal{L}(\theta_t, d_t)$ is the gradient of the loss function $\mathcal{L}$ evaluated on a mini-batch d_t from the training set $\mathcal{D}$. Typical loss functions $\mathcal{L}$ are the mean squared error or the cross-entropy loss which is widely used for regression or classification respectively [83]. Learning relies on the gradient of the loss function with respect to the network weights. Therefore, it is necessary to calculate the gradient of the complicated function defined by the deep network. The backpropagation algorithm [182] provides an efficient method to compute such gradients. The form of SGD described in (2.4) uses mini-batches d_t to pool gradients from multiple data samples together in order to obtain a robust update signal, while keeping the stochastic behavior to escape saddle points [75]. Many variants of SGD exist as more or less sophisticated optimizers [83] (e.g. RMSProp, AdaDelta, and Adam) that make us of special update rules based on the idea of momentum [223] and are often combined with other techniques such as learning-rate-schedules [97] to improve the convergence speed and robustness of the optimization. The topic of optimization methods for deep learning is a very active area of research, as the performance of trained models often is strongly influenced by the optimization. Since the optimization with SGD is sensitive to initialization, various initialization methods exist to improve the optimization.

Usually small random weights are used to avoid problems like exploding or vanishing gradient.

Modern deep learning libraries like PyTorch [168] and TensorFlow [1] make building and optimizing deep networks easily accessible to a broad user base by providing features such as easy specification of network architectures and automatic differentiation.

2.3.3 Regularization

The representation of a deep network is only shaped to some degree prior to learning, since most of it is determined by end-to-end learning. In this context, regularization has an important role in guiding deep learning towards better generalization while avoiding overfitting. Various regularization methods for deep learning have been proposed that allow regularization of the learning procedure or the deep network architecture.

Regularization of Model A deep network model can be regularized by the selection of a particular network architecture f. The network architecture represents an inductive bias that implements assumptions about the task in order to improve learning (i.e. function approximation) and in particular generalization. [214] show that a well designed deep architecture for visual recognition is a strong feature extractor even before training. The implementation of this inductive bias can be implicit or explicit in terms of the network weights and connectivity. Instead of hardwiring (i.e. handcrafting) the representation of a network, the representation is mostly shaped in a soft or implicit manner that prioritizes certain weight configurations and is only explicit for some properties like the potential maximum depth or width of a network. In the following we provide a list of popular regularization techniques used in deep network architectures:

- Parameter sharing (or weight sharing) allows to share a certain trainable parameter in other parts of the network. This reduces the complexity of the network by reducing the number of free parameters and forces the parameters to become reusable. Convolutional neural networks [126] are a prime example, where the same filter (i.e. group of parameters) is applied to the input in a sliding window manner. This results in learning shift-equivariant features that are sensitive to local image patches.

- Skip connections [140] provide an option to bypass layers in order to simplify the activation and gradient flow during learning. In other words, the network can adapt its depth during learning. A popular architecture employing skip connections is the residual network [91].

- Dropout layers [121] randomly remove part of the feature activations passed through a network. This makes the network robust against random changes in the connectivity.

- Normalization layers normalizes the features (e.g. batchnorm [107], instancenorm, layernorm [15], groupnorm [227]) or weights (e.g. weightnorm) of a neural network to improve optimization and enable very deep networks to converge. The batchnorm layer is probably the most famous variant that normalizes the features of a layer by the mean and variance computed within a mini-batch.

Regularization of Learning Procedure The optimization procedure of a deep network can be regularized in several ways. Usually regularization is achieved by adapting the loss function and its regularization term, the optimization method, or the data augmentation method. In the following we provide a list of popular regularization techniques used in the deep learning procedure:

- Data augmentation creates additional training data by applying transformations (e.g. rotation, cropping, color jitter) to existing data $\mathcal{D}_{train}$. The selection of transformations is task-specific (e.g. mirroring of text might cause ambiguity). While often countless transformations are feasible, the novelty of the generated data is usually limited, since the augmented dataset remains only an approximation of the original data distribution. Nevertheless, data augmentation makes the trained representations robust or even invariant to such transformations.
- Weight decay [83, 171] regularizes the network weights using an norm like L^2 that favors small weights that tend to be more stable with respect to noise.
- Variants of SGD optimizer (i.e. different update rules) based on techniques like momentum as mentioned in Sect. 2.3.2.
- Weight initialization often amounts to sampling the weights from carefully selected distributions (e.g. *Xavier* [82] or *Kaming* [92] initialization) to facilitate the optimization. A variant of weight initialization is pre-training the network on a similar task and then fine-tuning it on a downstream task. Network weights pre-trained on a large dataset like ImageNet [54] offer a strong initialization for many computer vision tasks and can reduce the training time and data required for the downstream task [90].

2.3.4 Network Architectures

General network architectures like Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN) have evolved as alternatives to Fully-Connected Networks (FCN) in order to deal with the special challenges of visual and sequential data (such as text) respectively. CNNs and RNNs rely extensively on the idea of weight sharing to efficiently scale deep networks to handle high-dimensional data such as images or long text sequences. Due to the rapid development in deep learning and computer vision, we cannot provide a complete picture of the existing deep architectures, but focus on some of the most important developments with emphasis on visual recognition tasks.

CNN Architectures

CNNs are a specific type of deep networks that are generally composed of convolutional layers that perform a convolution operation over the input by applying multiple trainable filters with a specific window size (e.g. 3x3 pixels) to the the input data. CNNs are well suited to learn rich feature representations of natural images [39, 59, 145, 196, 236]

Image Classification The image classification performance in the ImageNet challenge [184] has been a good indicator for the progress in the design of CNN architectures over the last couple of years. Starting with the breakthrough of AlexNet [121], the deep architecture design dramatically improved over a period of about five years with rapid innovation. The top-5 error on ImageNet dropped from sixteen percent for AlexNet [121] below five percent for models such as ResNeXt [92, 229] which is also considered the Human error rate for this task [56]. Other famous architectures with top results at the ImageNet challenge are VGG [197], GoogLeNet [203], ResNet [91], and PNASNet [136]. AlexNet is considered the first successful deep CNN architecture. It consists of five convolutional followed by three fully-connected layers. AlexNet combines ReLUs, max-pooling layers, normalization layers and dropout to significantly outperform the best shallow methods. The later deep network designs introduce various modifications: They grow in depth sometimes beyond one hundred layers, replace all fully-connected layers with convolutional layers, drop the intermittent pooling layers in favor of changing the stride of convolutions, use 1x1 convolutions and 3x3 convolutions rather than larger window sizes, use batchnorm, use skip connections (aka residual connections), introduce subblocks with identical design that are stacked together, and various other techniques like improved initialization methods or optimization techniques. Initially, architectures were designed to maxout the existing GPU hardware of the time. Later, flexible and more compact architectures for mobile applications emerged, such as MobileNet [187] and SqueezeNet [106], where performance is optimized in terms of memory and computational efficiency. Many successful deep architectures for image classification have been utilized as so called *backbone networks* in many computer vision task for visual recognition and beyond [141, 233, 239]. If pre-trained on ImageNet, these backbone networks provide strong off-the-shelf feature extractors that can be efficiently used for transfer learning by means of fine-tuning or other techniques [117, 131]. The backbone network is usually extended with a *task-specific head* to solve a particular downstream task.

Object Detection For the task of object detection, various network architectures have been proposed. They mostly differ in the architecture of the detection head and the number of stages used to compute the detection. R-CNN [81] is one of the first deep learning based object detectors. It relies on other algorithms to produce a large number of region proposals (e.g. selective search [213]) for each input image. For each region proposal, the image region is cropped, resized to a fixed size, and passed into a pre-trained backbone CNN to extract features. Then these features are used as input to a set of class-specific linear support vector machines to output an object classification. Bounding box regression is applied as postprocessing to improve the localization of the region detections further. Fast R-CNN [80] is the successor to R-CNN, that applies the backbone CNN once to the whole image and extracts the region proposal directly from the resulting feature map using region-of-interest pooling. This speeds up the computation significantly. Faster R-CNN [176] learns its own region proposal method in an end-to-end fashion in place of the external method. All R-CNN approaches follow a two-stage procedure that computes features and proposals in the first stage and uses the extracted features for classification of the proposals in a second stage. Alternative methods were proposed that compute the localisation and the classification of object detections in a single pass, most famously YOLO [175] and SSD [137]. This is achieved by placing a grid over the input image and predicting for each grid cell an object class and the most likely shape of the object. SSD uses for the box prediction anchor boxes that vary in aspect ratio and scale. Additional multiple grids at different scales can be used to better deal with objects of different size. Bounding box regression is used to improve the fit of the anchor boxes. Many new architectures for object detection have been proposed that combine ideas from other areas of deep learning such as semantic segmentation [134, 135], key point detection [60], or neural architecture search [204] to further improve detection performance.

Semantic Segmentation CNNs for semantic segmentation have similarly undergone rapid development. [140] present one of the first deep architectures for segmentation that employs a pre-trained backbone from image classification by turning the last fully-connected layers into convolutional ones using a net-surgery trick. Such fully convolutional architecture deals with arbitrary input resolutions and outputs a dense prediction map for the whole image. The introduction of U-Net [156, 179], i.e. hourglass-shaped networks with skip connections, and dilated convolutions [40] provide better spatial resolution at deeper, more semantic layers of the network to improve the quality of the segmentation.

RNN Architectures

While CNNs are most successful in processing two-dimensional data, RNNs are specialized on processing one-dimensinal data, i.e. sequences. RNNs are mostly used for natural language processing and speech recognition. They can scale to much longer sequences than possible for general neural networks. Additionally, they can handle sequences of varying length. The RNN can take a sequence as input as well as output a sequence [85]. Inference in an RNN means to propagate information through multiple time steps that are implemented as blocks connected by a hidden state and which take an input and produce an output. Using the hidden state, the computation takes the context information from previous time steps into account. It is possible to stack multiple RNN blocks on top of each other to create a deep RNN. The computational graph of an RNN (i.e. unrolling the recurrent connections according to the sequence of computations) represents a very deep network with extensive weight sharing between subsequent blocks each of which represents a single time-step of the RNN. Due to this depth of RNNs, the computation of the gradient for training requires many multiplications of small numbers which often results in the problem of vanishing gradient. There is also the problem of exploding gradient, but it can be handled by simple gradient clipping. To deal with the vanishing gradient problem, there exist special designs for the RNN block. Some of the best performing RNN architectures are based on the Long Short-Term Memory (LSTM) [96] unit or Gated Recurrent Unit (GRU) [43].

Sequence-to-Sequence Architecture RNNs can also be used to map an input sequence to an output sequence that is not necessarily of the same length (i.e. going beyond mere transduction). This is relevant for many applications such as machine translation. The sequence-to-sequence (seq2seq) architecture [43, 201] provides a general framework for mapping one sequence to another one, independent of their respective lengths. It is based on an encoder-decoder architecture: An *encoder* or input RNN processes the input sequence. The encoder outputs a context $\boldsymbol{z}$ that summarizes the input sequence. A *decoder* or output RNN generates the output sequence conditioned on the context. This architecture effectively decouples the processing of the input sequence from the output sequence. Therefore, the decoder can not only be conditioned on the context from the input sequence, but on arbitrary representations (e.g. conditioned on image features for image captioning [7]). The main bottleneck of the seq2seq architecture is the compression required by the fixed-size context vector $\boldsymbol{z}$. To reduce this problem, the context is expressed as a context sequence of variable length. In addition, an *attention mechanism* [16] is introduced to retrieve only the relevant parts of this context sequence at each step of decoding. The relevant context information is computed as a weighted sum over the elements of the context sequence where the weight is computed by an attention function based on the expectation of the decoder. Different attention functions have been proposed in the literature [16, 142]. The attention mechanism not only enables better memory retrieval, but also provides some insights into the decision making of the neural network through the computed attention weights that select the input at each time step. While the seq2seq architecture has originally been developed based on RNNs, more recent approaches show that the encoder-decoder architecture combined with feedforward networks such as CNNs can be efficiently used for processing sequences. In this context, the transformer architecture [216] introduces special multi-head attention blocks with self-attention and positional encoding with impressive results, in particular in terms exploiting long-term dependencies.

2.4 Machine Learning Algorithms for Limited Supervision

When designing machine learning algorithms for tasks with limited supervision, the central idea is to leverage alternative forms of supervision for learning. The supervised learning setting described in Sect. 2.2 is special, as it requires annotated training samples which are not readily available for the majority of computer vision tasks found in the wild. In case of limited supervision, two alternative sources of supervision from the task specification (cf. Sect. 2.1) are available to correct the imbalance: Data with limited supervision (e.g. unsupervised or weakly supervised data) and task-specific prior knowledge (e.g. geometry of an object). The goal is to create integrative approaches for learning that can utilize these alternative information sources. For this purpose, the inductive bias in terms of the model and learning procedure of the machine learning algorithm have to be adapted. Such integrative approaches rely on two complementary design principles to compensate for the missing supervision:

1. Reduce the supervision necessary for learning by controlling the space of solutions. The idea is to trade off variance for inductive bias in the model and learning procedure.

2. Generate supervised data from alternative sources by extending the learning procedure. The idea is to wrap supervised learning in an iterative process that switches between generating annotations for data with limited supervision and refining the model.

The first principle has already been described as an essential part of supervised learning in Sect. 2.2, however, its implementation requires additional attention in the case of limited supervision. The second principle applies exclusively to the case of limited supervision, since supervision is already given in the other case (usually by means of manual annotation). In the following, we describe how both design principles are utilized for learning with limited supervision in modern machine learning algorithms with a focus on deep learning-based methods.

2.4.1 Control the Space of Solutions

An essential part of designing a machine learning algorithm is to shape the inductive bias and ultimately control the space of solutions as highlighted in Sect. 2.2. The idea is to adapt the model and learning procedure based on task-specific prior knowledge in order to obtain the inductive bias necessary for successful learning. In case of limited supervision, the inductive bias is especially helpful, as it can reduce the overall supervision required for training the model. However, controlling the space of solutions too tightly risks excluding the best solutions prior to learning and results in underfitting. Instead of only increasing the inductive bias, we have to find the best possible bias-variance trade-off for the task at hand. In the following, we review the key design decisions for a machine learning algorithm as introduced in Sect. 2.2 with an emphasis on reducing the amount of supervision required for learning.

Choice of Model Family

The choice of the model family determines the degree to which the bias and variance of a machine learning algorithm can be controlled. The variance corresponds to the representational capacity of the model architecture whose choice is discussed afterwards. The bias corresponds to the inductive bias in the learning procedure and model. A model family defines a certain separation between representation and learning, which is essential for modelling the inductive bias. With a strong separation, the representation can be shaped more explicitly prior to learning. This allows more task-specific prior knowledge, such as structural priors, to be translated into inductive bias. For tasks with limited supervision, it is important to choose a model family that makes the best use of existing prior knowledge.

In the case of deep learning, representation and learning are not clearly separable as we have seen in Sect. 2.3.1. Deep networks learn a representation from data implicitly, making it difficult to enforce explicit assumptions [18, 83]. Therefore, it is not always possible to explicitly express prior assumptions in the model architecture. In contrast, a Probabilistic Graphical Model (PGM) [115], also known as structured probabilistic model, separates representation and learning more clearly. PGMs provide a framework to explicitly integrate task-specific prior knowledge in terms of a graph that expresses the conditional dependence structure between random variables. PGMs can be designed according to structural assumptions about the task, instead of having to discover the structure from patterns in the data. This significantly reduces the necessary training data for learning. However, the ability of PGMs to express strong priors comes with an increased manual design effort compared to end-to-end deep learning. Thus, it can be helpful to combine the rich feature representations learned by deep networks with the structural prior of the PGM [40, 133, 191]. In recent years, research in structural approaches for deep learning has significantly increased, in particular approaches with graph neural networks or graph convolutional networks [18, 114] have emerged that can express structural priors while still learning most of the representation from data.

Design of Model Architecture

The design of the model architecture determines a large part of the bias-variance trade-off, i.e. it controls the model in terms of structural prior (inductive bias) and representational capacity (variance). The trade-off between bias and variance must fit the complexity of the task, otherwise learning may suffer from overfitting or underfitting. Models with too high capacity may memorize the training data rather than learn how to generalize to unseen data. Models with too little capacity may fail to fit the data properly and thus provide a rough over-generalization. A useful heuristic in the selection of a model is Occam's razor: Given multiple models with the same performance for a task, choose the simplest one.

The representational capacity of a model is a good control for the bias-variance trade-off. There are several ways to control the representational capacity [83] which depend on the model family (e.g. linear regression, or deep model). A common hyperparameter for the capacity is the number of model parameters θ. In deep learning, the depth and width of a network are also used to control its representational capacity. For tasks with limited supervision, a reduction of the variance of the model might be required, even though it might bear the risk of overfitting. Instead of only controlling the capacity of the model architecture by reducing the variance, it is important to increase the inductive bias based on prior knowledge.

Regularization of Model and Learning

Regularization techniques aim to prioritize some solutions over others, instead of enforcing a hard constraint on the representational capacity of a model architecture. The techniques are important for dealing with the bias-variance tradeoff. Without reducing the variance of the model, they mitigate the problem of overfitting and improve generalization. For tasks with limited supervision, the application of regularization techniques is essential for reducing the supervision required for learning. In deep learning, the best model is often a large model (i.e. high representational capacity) combined with a lot of regularization [122, 237]. This appears to contradict the intuition of the bias-variance tradeoff, however, it is still relevant for deep models: While regularization does not affect the representational capacity, it does change the *effective capacity* of a model, since many solutions are effectively excluded due to prioritization. Thus, regularization represents an implicit reduction of variance. We present various regularization techniques for deep learning in Sect. 2.3.3.

2.4.2 Generate Supervised Data

The second principle for learning from data with limited supervision is to generate supervised data to compensate for the lack of supervision. In the following, we provide a framework for studying the different implementations of this principle in the literature. The general approach is to extend the standard learning procedure (cf. Sect. 2.2) with a method for generating annotations from data with limited supervision. This is accomplished by effectively wrapping supervised learning in an iterative learning procedure that switches between generating supervised data and training a model as visualized in Fig. 2.3. We refer to this procedure simply as *Iterative Learning* (IL). The generated supervised data allows to train the model, which in turn can be used to generate more supervised data. IL is related to the idea of incremental learning [51], as each iteration gradually expands the scope of the training data and refines the model. Since this form of incremental learning is performed in a bootstrapping fashion (on the same task), we can also think about IL as *self-incremental learning*. IL is independent of a particular model architecture or family, and only relies on task-specific prior knowledge to turn data with limited supervision

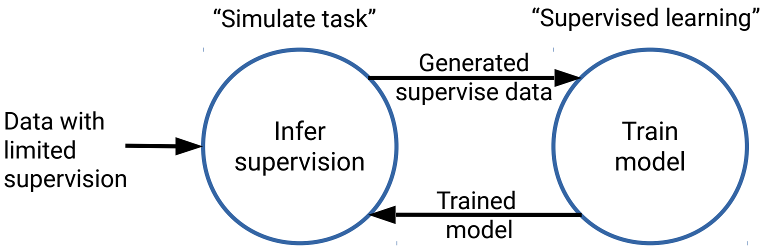


Figure 2.3: Generating supervised data for iterative learning with limited supervision: The central idea is to wrap supervised learning inside an iterative procedure that alternates between inferring supervision for the task (simulate) and training a model (learn). The generated supervised data allows to train the model, which in turn is used to generate more supervised data. We refer to this procedure as *Iterative Learning* (IL), since each iteration gradually expands the scope of training data and refines the model to solve the same task.

into supervised data. The IL procedure varies according to the different cases of limited supervision. In particular, the case of unsupervised, weakly supervised or semi-supervised data are distinguished in the literature. The considered approaches associated with these three cases of limited supervision are *self-supervised learning*, *weakly supervised learning* and *semi-supervised learning*. They all represent specializations of IL as visualized in Fig. 2.4. The three approaches are complementary to each other, which is also evident in the shared learning procedure. Thus, they can be advantageously combined depending on the forms of limited supervision available for the task. Below we describe for each case of limited supervision the corresponding variant of IL:

Learning with Unsupervised Data

Unsupervised learning is a major field of machine learning [25]. While there exist various approaches to this problem such as clustering and density estimation, we focus on self-supervised learning in this thesis. Self-supervised learning aims to learn a meaningful representation for a downstream task by designing an auxiliary task for which the supervised data for training can be automatically generated. Such an *auxiliary task* is also called self-supervised, surrogate or pretext task in the literature, since it only serves as a pretext for learning a useful representation for a downstream task.

From the perspective of the downstream task, self-supervised training on the auxiliary task represents a form of pre-training. Thus it usually operates in two stages: First, IL of a representation by training on a self-supervised task. Second, building a model based on the learned representation and fine-tuning it to solve a downstream task. The general objective is to learn a representation in the first stage that minimizes the required supervision for fine-tuning in the second stage while achieving the best performance on the downstream task. A good representation is usually not too specialized on the auxiliary task, but rather provides a rich representation for different downstream tasks. A key challenge of self-supervised learning is the design of auxiliary tasks that promote learning good representations for downstream tasks while being self-supervised. Many different auxiliary tasks have been proposed in recent years in computer vision and natural language processing [111]. Most tasks are based on reconstructing or predicting a part of unsupervised data

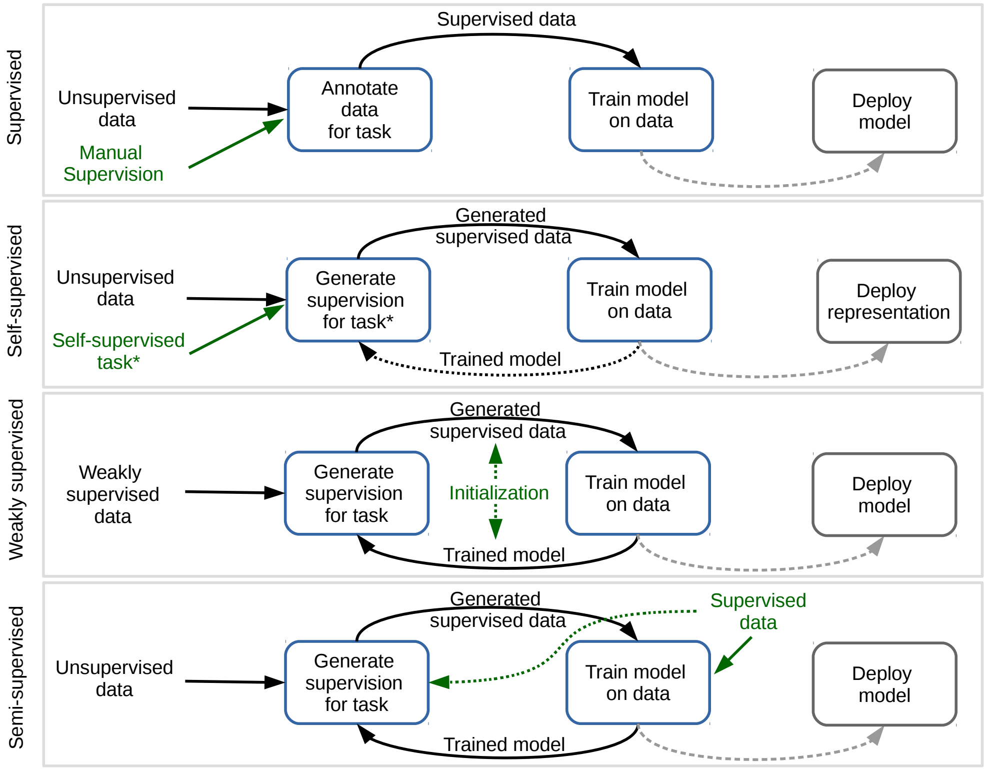


Figure 2.4: Learning approaches mainly vary according to their source of supervision. While the supervised approach relies on manual annotation for training the model, the other approaches represent special cases of IL that alternates between automatic generation of supervision and refining the model. Self-supervised: A auxiliary task (marked by asterisk) is designed for which supervised data can be automatically generated. It is possible to leverage the trained model for IL. Only the representation is used for the target task. Weakly supervised: Weakly supervised data is used to generate training data for a target task. For initialization either a pre-trained model or an initial set of generated annotations must be obtained. Semi-supervised: A little supervised data is available to train an initial model and start IL. The trained model as well as supervised data can be used to optimize the generation of annotations for the unsupervised data.

from a transformed version, which is often obtained by deletion. Recent approaches rely on the efficiency of deep learning to learn meaningful representations from training on various auxiliary tasks. In addition, deep learning supports an easy and flexible transfer of the learned representation to a downstream task [131].

Most approaches for self-supervised learning only train the self-supervised task once and do not implement the feedback loop of the IL procedure as visualized in Fig. 2.4. However, some approaches utilize the potential of IL and iterate the self-supervised training to increase performance. Techniques such as clustering [34], model distillation techniques [159] or optimal transport [14] are employed to improve the generated annotations for continuous learning. Recently self-supervised methods based on contrastive learning show impressive results [41, 89].

Learning with Weakly Supervised Data

Weakly supervised learning means that each training sample comes with coarse supervision, i.e. a lower degree of annotation than required at test time. Weakly supervised learning aims to train a model directly to solve a target task and does not require a subsequent fine-tuning step as in the case of self-supervised learning. Weakly supervised learning is based on a IL procedure that continuously alternates between generating supervised data, by bridging the gap between weak and full, and refining the model. IL enables a gradual refinement of the generated annotations and the trained model.

Weakly supervised learning requires an initialization of IL, since neither an initial model nor an initial set of generated annotations is available in the beginning. A typical initialization by means of the model is to use a deep network pre-trained on a related task. Because a suitable model for initialization is often not available, many approaches focus on producing an initial set of generated annotations using simple heuristics based on task-specific prior knowledge. Such an initialization may result in a first round of very noisy and ambiguous generated annotations for training. Fortunately, the deep learning method is relatively robust to noise and learns useful representations even under adverse conditions [178, 237]. Nevertheless, it is important to consider the sensitivity of weakly supervised learning to the initialization. In the worst case, a bad initialization inhibits further IL.

The supervision generation method typically uses task-specific prior knowledge and the pre-trained model from the previous iteration to transform weak supervision into full supervision. This step represents a form of optimization that checks the consistency between the generated annotations (by the pre-trained model) and the weak supervision based on general assumptions about the task (e.g. the appearance of an object). Depending on a task, approaches differ how they combine weak supervision and task-specific priors (cf. Sect. 2.1.3) to enable weakly supervised learning. Various weakly supervised learning approaches have been proposed for visual recognition, which mostly belong to one of two general problem classes:

- Object detection with weak supervision such as image-level supervision [24, 177, 205].
- Semantic segmentation with weak supervision such as image-level or bounding box supervision [20, 36, 160].

The weak supervision of these tasks is usually associated with missing localization information (e.g. location and extent of objects). To obtain the localization from weak supervision, the use of prior knowledge such as appearance models or region proposal is essential for addressing the inherent ambiguity. Moreover, deep networks provide an valuable tool to implicitly learn the appearance of objects from weak supervision, which can be effectively exploited.

Learning with Semi-Supervised Data

Semi-supervised learning is a common learning problem because it can be created simply by augmenting an existing supervised dataset with unsupervised data. The abundance of unsupervised data together with the capabilities of deep learning has triggered a surge of research in self-supervised learning in recent years. Approaches for semi-supervised learning can also be considered a special case of IL, as illustrated in Fig. 2.4. Like weakly supervised learning, semi-supervised learning directly trains a model for a target task. In contrast to weakly supervised learning, most semi-supervised approaches do not require an initialization method, since at least a few annotations are given at the beginning of the learning process. However, the supervision generation is usually more challenging than in weakly supervised case because there is only unsupervised data without any additional hints. An important goal of semi-supervised learning is to make the best use of the given annotations to optimize the annotation generation on the unsupervised part of the data. For this purpose, several techniques have been proposed that aim to regularize the consistency between the supervised data and the annotations generated for the unsupervised data. Approaches based on self-training with pseudo-labels [129, 228] are among the most influential.

2.5 Background of Language Modeling

Language can provide an alternative form of supervision for many visual recognition tasks such as scene text detection [151] or image captioning [7]. This section provides some background on how the inherent structure of a natural language can be exploited by a computer vision algorithm. For this purpose, we introduce the task of language modeling, which belongs to the domain of Natural Language Processing (NLP) and aims to learn a representation of a natural language. In the following, we describe NLP concepts and algorithms for language modeling.

2.5.1 Language Model

A Language Model (LM) aims to estimate the probability distribution over sequences of tokens, usually words or characters, in a natural language. In a good LM a random sequence of words should have a low probability, whereas a syntactically and semantically correct sentence should have a higher probability. By querying the LM, we can identify very improbable, potentially erroneous sentences. Most LMs are integrated in larger systems for various applications such machine translation or text recognition. LMs generally aim to compute the conditional probability of a word w_t given its $n-1$ previous words, i.e.

$$p(w_t|w_{t-1}, \cdots, w_{t-n+1})\,. \tag{2.5}$$

The probability of a sentence can be approximated with a LM using the Markov assumption (based on the window size $n-1$) and the chain rule. Then, a LM assigns a probability to a sequence of m words by computing

$$p(w_1, \cdots, w_m) = \prod_{t=1}^{m} p(w_t \mid w_{t-1}, \cdots, w_{t-n+1}) \tag{2.6}$$

2.5.2 n-gram Language Model

The n-gram language model is a simple non-parametric approach to model the joint distribution of a sequence of words. It is based on the idea to quantify the probability of fixed-length sequences of words, called n-grams, by counting the number of their occurrences in a training set (i.e. a text corpus). An n-gram is a sequence of n words. It assumes that the probability of any word in this sequence only depends on the $n-1$ previous words as in (2.5). The probability of an n-gram is computed from the relative frequency of the word sequences in the train dataset as follows:

$$p(w_t \mid w_{t-1}, \cdots, w_{t-n+1}) = \frac{count(w_{t-n+1}, \cdots, w_{t-1}, w_t)}{count(w_{t-n+1}, \cdots, w_{t-1})} \,. \tag{2.7}$$

In general, the n-gram model requires the storage of $|V|^n$ parameters, i.e. the counts of all possible combinations. This highlights two important limitations of the n-gram LM: First, the exponential dependency between n and the model parameters prohibits scaling n-gram models to larger n, in particular with large $|V|$. Thus n-gram models cannot be efficiently used to capture long-term dependencies. Second, the probability of an n-gram estimated from the counts in the training data is very likely zero in many cases. If such unseen n-gram occurs in the test set, the model does not provide a useful estimate of the probability. To overcome these limitations, a LM must be able to share statistical information between one word and other semantically similar words.

2.5.3 Tokenization

The tokenization of a text sequence, i.e. how to split it into discrete entities, is an important aspect of a LM, since it determines the granularity of the language statistics used by the model. The two most common approaches are character-level and word-level tokenization. An advantage of character-level tokens is that it is easy to deal with unseen or new words. An advantage of word-level tokens is that they directly operate on a semantically more meaningful level. Depending on the tokenization, we obtain a vocabulary V of all tokens in our text dataset. The size of the vocabulary $|V|$ is usually much larger in the case of word-level tokenization. Of course, this depends on the writing system (cf. Latin alphabet and Chinese traditional writing system). In the case of neural networks, character-based tokenization is now relative common, in particular for morphologically rich languages (i.e. that can form many unknown words by different compositions). Character-based language models based on neural networks still learn semantic rich representations while avoiding the problem of dealing with a very large or even open vocabulary.

2.5.4 Word Representation

Words, characters, or more general tokens need to be represented as vectors of real numbers to enable processing by a machine learning algorithm. A basic approach is to represent each word of a vocabulary V as $|V|$-dimensional one-hot vector, there every pair of words has the same distance to each other. A more useful word representation is provided by *word embeddings*, that embed words in a feature space of lower dimension independent of $|V|$ where semantically related words (i.e. that often appear in the same context) are close to each other. Word representations based on word embeddings provide a similarity measure for words and are employed to boost the performance of LMs and other NLP methods. Word embeddings are usually obtained by training a neural network on large text datasets. A popular method is to train the word embeddings as the weights of a shallow neural network using standard stochastic gradient descent. The embedding training is performed in a self-supervised fashion that automatically creates training samples from a text corpus. The self-supervised tasks as in the case of [149, 150] usually involve predicting a center word given the surrounding context words (skip-gram) or vice versa predicting the surrounding words from a given center word (continuous bag of words). The Global Vectors for word representation (GloVe) [169] method builds on top of the skip-gram approach and additionally makes use of co-occurrence matrices computed on the text dataset.

2.5.5 Neural Network as Language Model

Language models based on neural networks are fix-sized parametric models that are the current standard approach in NLP for language modelling. Such a LM overcomes the limitations of an n-gram LM by learning a representation shared between all words. Words are treated similarly if they share features captured by the representation. Based on this similarity, the LM can generalize from a sequence in a training set to many semantically similar sequences. A neural network-based LM [200] with model parameters θ estimates the conditional probability of the next word (cf. (2.5)) based not only on the previous $n-1$ words, but on the entire word history ($w_{<t}$) as follows:

$$p(w_1, \cdots, w_m) = \prod_{t=1}^{m} p(w_t | w_{<t}; \theta) \tag{2.8}$$

The LM corresponds to a decoder network as introduced in Sect. 2.3.4 that computes the conditional probability in an auto-regressive fashion. Different network architectures for a LM decoder have been proposed such as FCN, CNN, RNN and recently transformers [216]. Similar to training the shallow word embedding model, a LM decoder can be obtained by self-supervised learning, which requires only unsupervised training data. The LM decoder is trained to perform a self-supervised task, such as reconstructing a target sequence from a modified input sequence. The LM decoder learns a representation of a natural language that can be integrated as pre-trained model in many applications. For example, a LM decoder can be used as part of a seq2seq architecture in applications such as machine translation or text recognition.

CHAPTER 3

Self-Supervised Human Pose Analysis

Human pose analysis is presently dominated by deep convolutional networks trained with extensive manual annotations of joint locations and beyond. To avoid the need for expensive labeling, we exploit spatiotemporal relations in training videos for self-supervised learning of pose embeddings. The key idea is to combine temporal ordering and spatial placement estimation as auxiliary tasks for learning pose similarities in a Siamese convolutional network. Since the self-supervised sampling of both tasks from natural videos can result in ambiguous and incorrect training labels, our method employs a curriculum learning idea that starts training with the most reliable data samples and gradually increases the difficulty. To further refine the training process, we mine repetitive poses in individual videos which provide reliable labels while removing inconsistencies. Our pose embeddings capture the visual characteristics of human pose that can boost existing supervised representations in human pose estimation and retrieval. We report quantitative and qualitative results on these tasks in Olympic Sports, Leeds Pose Sports and MPII Human Pose datasets.

3.1 Introduction

The ability to recognize human posture is essential for describing actions and comes naturally to a human being. Different poses in a video form a visual vocabulary similar to words in text. An important objective of computer vision is to bring this ability to the computer. Finding similar postures across different videos automatically enables a lot of different applications like action recognition [11, 12] or video content retrieval.

So what makes two postures look similar? More formally, a similarity function, which is entailed by a pose embedding, needs to capture the characteristics of different postures, while exhibiting the necessary invariance to strong intra-class variations. In particular, it should be sensitive to the articulation of body parts while being invariant to illumination, background, clutter, deformations (e.g. facial expressions) or occlusions. Often human joints are used as a surrogate for describing similarity, but there are several issues: First, measuring distances in pose space accurately and coming up with a non-ambiguous Euclidean embedding is already a challenging problem. Second, the manual annotation of human joints in larger datasets is expensive and time-consuming.

Convolutional networks have recently been immensely helpful to computer vision. The feature hierarchy of such a network is effectively defined by a cascade of filter banks that are recursively applied to extract discriminative features for the given task. In this chapter we take advantage of convolutional networks to learn pose embeddings from videos.

In supervised similarity learning, we assume that we are given positive and negative pairs of postures for training. In this supervised setting, convolutional networks excel and have recently surpassed human performance in some basic tasks. In contrast, unsupervised training of convolutional networks is still an open problem and currently the focus of the research community. In this chapter we investigate how to learn a pose representation without labels.

A solution for the problem of missing supervision is to switch to a related auxiliary task for which label information is available. For this self-supervised strategy, several well-known sources of weak supervision have recently been utilized as auxiliary tasks: including spatial configuration of natural scenes, inpainting, super-resolution, image colorization, tracking, ego-motion and even audio. Although there are many sources available, not all are suitable for application in pose analysis. We exploit human motion in videos to make pose similarity apparent and learnable without labels. Given the almost infinite supply of video data on the web, exploiting this idea is very attractive.

We propose learning spatiotemporal relations in videos by means of two complementary auxiliary tasks: a *temporal ordering* task which learns whether two given person images are temporally close (similar) and a *spatial placement* task which discovers randomly extracted crops from the spatial neighborhood of persons, and learns whether the given patches are a person or not. Learning the spatial and temporal relations of human movement simultaneously provides us information of “what” we are looking at (person/ not person) and “how” the instances differ (similar/dissimilar poses). *Curriculum-based* learning and *repetition mining* arrange the training set by starting from only the easy samples and then iteratively extend to harder ones, while also eliminating inactive video parts. Then our spatiotemporal embeddings successfully learn representative features of human pose in a self-supervised manner.

3.2 Related Work

Human pose analysis deals with problems such as pose retrieval, similarity learning and pose estimation. Most approaches in pose analysis rely on supervised data and there exist only a few unsupervised approaches. Here, we summarize significant examples of pose analysis and related unsupervised learning approaches:

Pose estimation Pose estimation aims at finding the locations of body joints, whereas pose retrieval or embedding finds a metric that can retrieve the most similar poses and can discriminate samples according to their pose information, without localizing joints directly. With the advancements in convolutional neural networks [120], pose estimation is also dominated by deep learning-based methods. Toshev and Szegedy [211] estimated joint locations directly regressing in a CNN architecture. Instead of simply regressing joint locations, Chen and Yuille [42] learned pairwise part relations combining CNN with graphical models. Tompson *et al.* [209] exploited CNNs for relationship between body parts with a cascade refinement. A recent work by Newell *et al.* [156] used fully convolutional networks in a bottom-up top-down manner to predict heatmaps for joint locations.

Similarity learning The first *Siamese*-type architecture [30] was proposed to learn a similarity metric for signature verification. Similarity learning was also applied in human pose analysis. In [155] and [124], body joint locations are used to create similar and dissimilar pairs of instances from annotated human pose datasets. [124] also transferred a learned pose embedding to action recognition.

These works in pose estimation and similarity learning exploited large amounts of annotations (body joints or labeling of similar/dissimilar postures). However, unsupervised learning methods without using labels showed promising performance in various learning tasks in the last decade. Self-supervised learning is very popular similar to classical unsupervised methods such as clustering, autoencoders [95], restricted Boltzman machines [182].

Self-supervised learning The availability of big data motivated the community to investigate alternative sources of supervision such as ego-motion [3, 235], colorization [238], image generation [173], spatial [57, 158] or temporal clues [153, 220]. As our approach belongs to the class of self-supervised methods using spatial and temporal information, we describe these methods in detail.

Doersch *et al.* [57] trained convolutional neural networks to take image patches from a 3×3 grid and classify the relative location of 8 patches with respect to a center patch. Norozzi and Favaro [158] argued that solving locations of relative patches could introduce ambiguities and proposed a localization problem given all 9 patches at once. Also, they used 100 relative locations as class labels out of 9! permutations using a Hamming distance-based selection.

Wang and Gupta [220] exploited videos by detecting interesting regions with SURF keypoints and tracking them. Then, they used a Siamese-triplet architecture with a ranking loss together with random negative selection and hard negative mining. However, tracking is not the best solution in the challenging context of pose analysis due to the non-rigid deformations of person patches which are in low resolution and contain too few keypoints to detect parts and track them precisely.

Misra *et al.* [153] defined a temporal order verification task, which classifies whether given 3-frame sequences are temporally ordered or not by altering the middle frame. In action/pose benchmarks or internet videos, there are a lot of cyclic human actions (*e.g.* running based sports, dancing), which often produce confusing samples and interfere with representation learning.

To learn a better representation, we argue that temporal cues aimed at learning whether given inputs come from temporally close windows or not are a more effective approach. The use of temporal cues to learn whether given inputs are from temporally close windows or not is an effective approach for representation learning. Local proximity in data (slow feature analysis, SFA) has first been proposed by Becker and Hinton [21]. The most recent spatial and temporal self-supervised learning methods are inspired from SFA. Goroshin *et al.* [84] created a connection between slowness and metric learning by temporal coherence. Motivated by temporal smoothness in feature space, Jayaraman and Grauman [109] exploited higher order coherence, which they referred to as steadiness, in various tasks. Slowness or steadiness criterion can introduce significant drawbacks mostly because of limited motion and the repetitive nature of human actions. Thus, we learn auxiliary tasks in relatively small temporal windows which do not contain more than a single cycle of action. Moreover, the use of curriculum learning [22] and repetition mining refine and

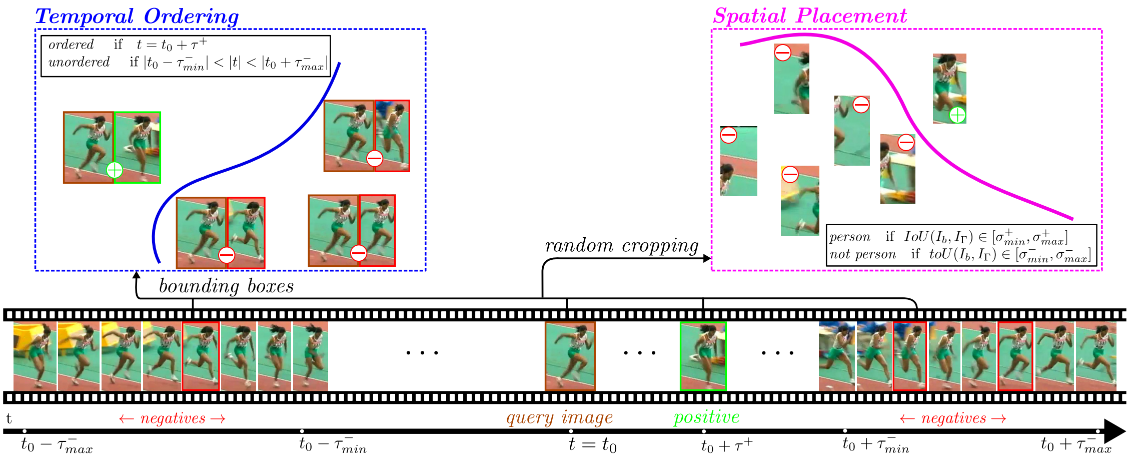

Temporal Ordering
ordered if $t = t_0 + \tau^+$
unordered if $|t_0 - \tau^-_{min}| < |t| < |t_0 + \tau^-_{max}|$
Spatial Placement
person if $IoU(I_b, I_\Gamma) \in [\sigma^+_{min}, \sigma^+_{max}]$
not person if $toU(I_b, I_\Gamma) \in [\sigma^-_{min}, \sigma^-_{max}]$
random cropping
bounding boxes
t
← negatives →
query image
positive
← negatives →
$t_0 - \tau^-_{max}$
$t_0 - \tau^-_{min}$
$t = t_0$
$t_0 + \tau^+$
$t_0 + \tau^-_{min}$
$t_0 + \tau^-_{max}$

Figure 3.1: Sampling procedure for training self-supervised pose embeddings. For each query image in a video, positive and negative pairs of temporal ordering are collected from specific temporal ranges *(left)*. In spatial placement, samples are cropped using the IoU criterion *(right)*.

guide our self-supervised tasks to learn stronger temporal features.

Curriculum learning has been proposed by Bengio *et al.* [22] and it speeds up training and improves test performance by using samples whose difficulties are gradually increasing in shape recognition and language modeling. To the best of our knowledge, the potential of a curriculum has not been studied in the self-supervised setting, where we associate the difficulty of training samples with their inherent motion.

3.3 Approach

Our motivation is to learn pose embeddings from videos without labels. We follow the insight that spatiotemporal relations in videos provide sufficient information for learning. For this purpose, we propose a self-supervised pipeline that creates training data for two auxiliary tasks: 1) temporal ordering and 2) spatial placement. Since the raw self-supervised output needs refinement, we introduce curriculum learning and repetition mining as key ingredients for successful learning. The two auxiliary tasks are trained in a Siamese CNN architecture and the learned features are eventually used as pose embeddings in order to retrieve similar postures and estimate pose.

3.3.1 Self-Supervised Pose Embeddings: Temporal Ordering and Spatial Placement

We consider a temporal and a spatial auxiliary task which are automatically sampled from videos as described in Fig. 3.1. Both tasks capture complementary information from inside videos essential for learning a pose embedding. The temporal task teaches the pose embedding to become more sensitive to body movements and more invariant to camera motion (*i.e.* panning, zoom in/out, jittering), while the spatial task relies on the spatial configuration of a single frame and focuses on learning a human appearance model which strengthens the ability to separate posture from background.

For the temporal ordering task, a tuple of two frames is sampled from the same video together with a binary label which indicates whether the first frame (anchor) is closely followed in time by the second frame (candidate). In order to focus on learning human posture, we do not sample the full frames, but instead crop bounding box estimates of the person of interest. Thus, the training input for the temporal ordering task consists of two cropped boxes and a binary label indicating whether the two boxes are temporally ordered.

For a frame I_{t_0} sampled at time point t_0, we sample a candidate frame I_t with a temporal offset of $\Delta_t = t - t_0$. In order to sample a positive candidate, the offset needs to be $\Delta_t = \tau^+$, while a negative candidate is sampled if

$$\Delta_t \in \tau^- = [\tau^-_{min}, \tau^-_{max}] \cup [-\tau^-_{max}, -\tau^-_{min}]$$

holds. $\tau^-_{min}, \tau^-_{max}$ are the range limits of the negative candidates. In other words, a positive candidate comes exactly from τ^+ frames in the future, while negative candidates come from ranges before or after the anchor frame.

The temporal ordering task relies on the assumption of temporal coherence that frames in a small temporal neighborhood are more similar than distant frames. We add the constraint that positive candidates can only come from the future. Since the self-supervised sampling from videos already introduces large amounts of variation, we want the positive class to be as homogeneous as possible in order to facilitate training. In contrast the negative class is sampled from a larger range that allows more variation, but is still close enough to the positive class to provide challenging similarities for discriminative learning.

For the spatial placement task, a box is randomly cropped from a single frame together with a binary label that indicates whether the cropped box overlaps with the estimated bounding box of a person in this frame. The overlap is measured with the Intersection-over-Union (IoU) criterion [63]. For the estimated bounding box I_b and a randomly cropped box I_r, the binary label y_S is defined as

$$y_S(I_b, I_r) = \begin{cases} 1, & \text{if } IoU(I_b, I_r) \in [\sigma^+_{min}, \sigma^+_{max}] \\ 0, & \text{if } IoU(I_b, I_r) \in [\sigma^-_{min}, \sigma^-_{max}] \end{cases} \tag{3.1}$$

where $IoU(\cdot, \cdot)$ computes the IoU and $[\sigma^+_{min}, \sigma^+_{max}]$ defines the positive range of overlap while $[\sigma^-_{min}, \sigma^-_{max}]$ defines the negative. Since the estimated bounding boxes are not completely reliable, the positive and negative IoU ranges are usually selected with a gap between them to help the separation of the classes.

In both auxiliary tasks, three negative samples are used for each positive posture, because sampling of negatives (what it is not) from larger ranges helps with learning positive similarities (what it is) precisely. The intuition is that the pose embedding learns to discriminate between a homogeneous positive and a more heterogeneous negative class in both tasks. Since both tasks focus on different aspects of human posture, the best pose embedding is obtained by joint training. We investigate the contribution of different configurations in Sect. 3.4.4.

3.3.2 Creating a Curriculum for Training

In supervised training with human annotations, it is often beneficial to avoid difficult samples with ambiguous or even incorrect labels, because this kind of data can inhibit convergence and lead to inferior results. In the self-supervised case, we find that the data quality fluctuates even more and needs to be taken into account. On the other hand, skipping too many difficult training samples can result in overfitting on a small subset of easy samples and hurts generalization to unseen datasets. We propose to strike a balance by using a curriculum of training data that gradually increases in difficulty over the course of training. We create the curriculum with regard to the temporal ordering task which produces far more inconsistent samples than spatial placement.

In order to determine the difficulty of temporal ordering for a particular training sample, we look into the motion characteristics of the respective video. For instance, a clean-and-jerk video mainly consists of inactive parts with little motion, whereas a long-jump video is dominated by a highly repetitive structure with fast moving, deforming postures. Training samples from video sequences with clear foreground motion (e.g. a long-jump video) are preferable for learning temporal ordering, because their negative candidates, which are sampled from the range of τ^-, are easier to distinguish from the positive ones from τ^+. Therefore, we determine the difficulty of a training sample by estimating the motion in videos and sample training frames with sufficient action.

When creating a curriculum, we use an optical flow based criterion that computes the ratio of the optical flow in the foreground and background of the frame. To compute the *fg/bg ratio* the mean magnitude of optical flow in the foreground bounding box is divided by mean magnitude of optical flow of the background. We use the method from [31] to estimate the optical flow between two frames. The fg/bg ratio acts as a proxy of a *signal-to-noise* ratio, as examples with higher values are more easily separated from the background.

The curriculum is assembled by sorting the training samples according to their flow ratio and splitting them in discrete blocks to form curriculum updates with increasing difficulty (decreasing flow ratio). We analyze the impact of the curriculum in an ablation experiment in Table 3.1 where we train the network with and without a curriculum using the same subset of self-supervised training data. Details of ablation experiments and the effect of the curriculum will be explained in Sect. 3.4.1 and Sect. 3.4.4.

3.3.3 Mining Repetitive Poses

There are two reasons why we pay special attention to repetitive poses in video sequences: First, they impair the training of the temporal ordering task. Second, if the location of repetitions were known, they could be extracted and used as valuable training data, which we refer to as *repetition mining*. The mined repetitions augment temporal ordering by providing a new similarity learning task.

While the proposed curriculum avoids difficult samples in the early stages of self-supervised training, repetitive poses in videos are not filtered by the motion-based curriculum. The training of the temporal ordering task suffers from repetitions which can cause incorrectly labeled image pairs by violating the assumption of temporal coherence. For instance, if a negative frame is sampled from a video with a repetitive action like running or walking, it might be more similar to the anchor frame than the positive candidate.

After an initial training of the temporal ordering task, we use the learned pose embeddings to detect repetitive poses in the training data. For each video, we obtain a self-similarity matrix by computing all pairwise distances between frames. As distance measure, we use the Euclidean norm of the normalized `pool5` features. In order to extract reliable and strong repetitions, we convolve the self-similarity matrix with a 5x5 circulant filter matrix

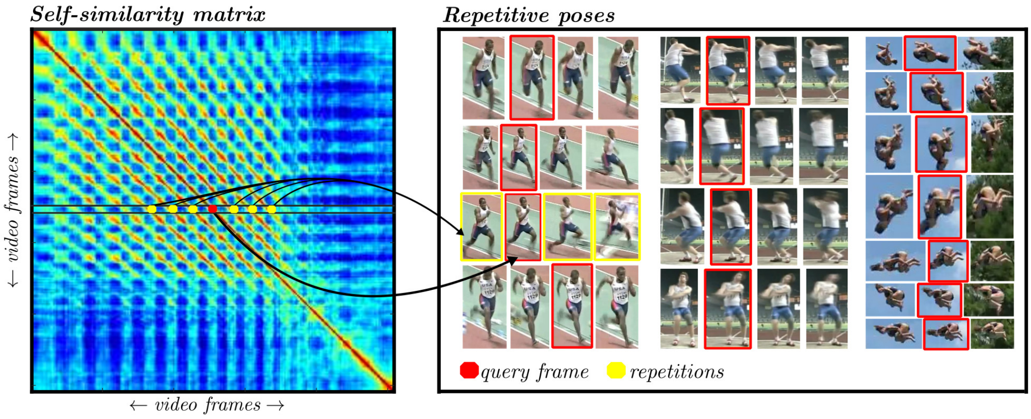


Figure 3.2: Mining repetitive poses. Off-diagonal structures of the self-similarity matrix on the left indicate repetitions in a video. For each row, repetitions are mined using a query frame. Repetitive poses from three videos are shown on the right.

to suppress potential outliers that are not aligned with the off-diagonals by thresholding. The maxima of each row indicate the fine-scaled repetitions of the respective query frame. Fig. 3.2 shows an example self-similarity matrix and repetitions which are mined using this approach.

Repetitive poses form groups of very similar but not identical images due to small variations over time caused by the persons movement, changes in the camera viewpoint, or even the frame rate of the video camera. These groups of highly similar images help to learn the finer details of human posture. They can be used to create a new type of similar-dissimilar problem. Similar pairs are chosen among repetition groups, negative candidates are picked from regions between the repetitions.

As repetitions occur only in a subset of the available video data, they are combined with samples from non-repetitive videos and added to the first stages of the curriculum. The mined repetitive poses are in quality close to human annotated similarities and provide a stabilizing effect on the whole training procedure.

Our method can be employed in a bootstrapping fashion, by repeatedly training the temporal ordering task and mining repetitions which provide better training samples without additional supervision.

3.3.4 Network Architecture

For the two self-supervised tasks we train two convolutional neural networks which differ in the number of images they process as shown in Fig. 3.3. The temporal ordering task is trained using a Siamese architecture [30] that takes a pair of images as input while the spatial placement task is trained on single images using a common single stream architecture.

We adopt the well-known Alexnet architecture [120] for both tasks. In the temporal task the two Siamese streams consist of convolutional layers. After the last pooling layer the output from the two streams is concatenated. The fully-connected layers compute a binary output probability for testing. The convolutional networks are trained by minimizing

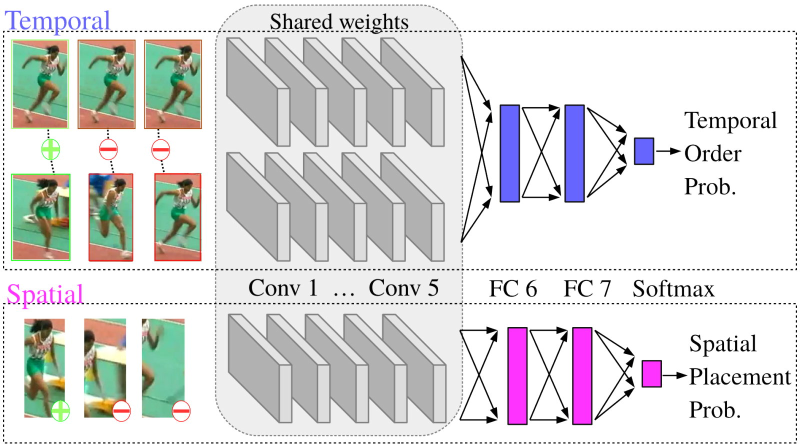


Figure 3.3: Network architecture for temporal ordering and spatial placement.

binary cross-entropy loss functions. For joint training of both tasks the weights in the convolutional layers are not only shared between the Siamese parts but are also shared with the convolutional layers of the spatial placement task. Moreover, the joint loss of the two auxiliary tasks is computed in a weighted sum.

After training the network, we use the feature representation from the last shared layer `Pool5` as pose embeddings. Features of this layer provide good localization which is important for pose retrieval and estimation.

We make several modifications to the Alexnet architecture: 1) Because we want to avoid overfitting and our binary tasks do not require a large number of parameters, both networks have a reduced number of neurons in the fully connected layers compared to the original Alexnet (namely 2048/1024 vs 4096/4096). 2) To improve training of the temporal task, we replace the regular rectified linear unit in the last convolutional layer with a non-linearity that has a negative slope. We find that this modification is critical for performance. 3) The use of batch normalization in the fully connected layers is an important regularizer in our training that helps with generalization to other datasets.

3.4 Experiments

We present experiments on posture analysis, pose estimation and pose retrieval. The training of our method is demonstrated on the Olympic Sports dataset (OSD) [157]. We provide details about the used curriculum and examples of mined repetitions. In different ablation experiments we highlight the design decisions in our proposed method. To study the ability of our approach to generalize to unseen datasets we include experiments on Leeds Sports Pose (LSP) [112] and the challenging and unconstrained MPII Human Pose [10]. Additionally in a supervised pose estimation setting [211], we report performance of our method in comparison with other initialization approaches.

3.4.1 Training and Testing Details

From 680 videos in Olympic Sports dataset, we extract approximately 140,000 frames for which we obtain bounding box estimates using the method in [64]. Our training curriculum

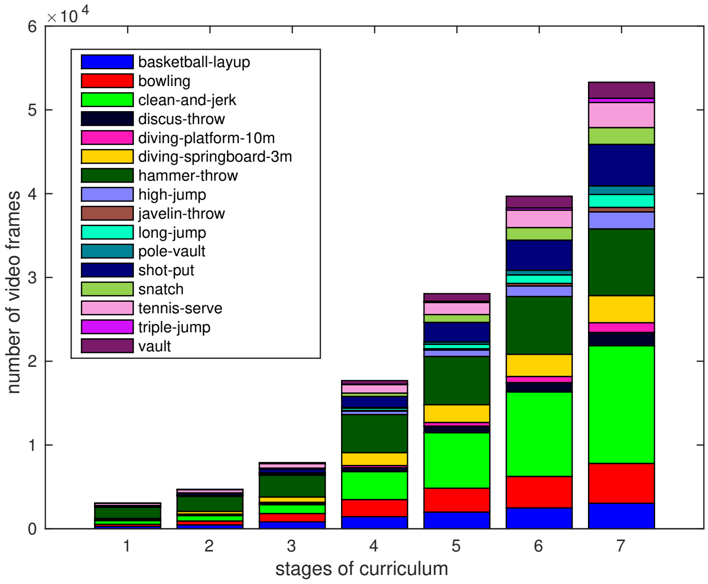


Figure 3.4: A histogram describing the curriculum used for training on Olympic Sports dataset. The curriculum contains seven stages with training samples of increasing difficulty.

uses about 80,000 frames which are ordered using the flow ratio criterion described in Sect. 3.3.2. It starts out with about five percent of the easiest training samples and increases the amount of training data in seven steps every 2.5K iterations. The amount of training data grows exponentially during the first few curriculum updates, but does not surpass 25 percent of training data for a single update.

For training of the convolutional networks, we use the *Caffe* framework [110]. We optimize our model using the Adam solver for stochastic batch gradient descent with batch size of 48 and a fixed learning rate of 10^{-4}. In the convolutional layers, we use a reduced learning rate of 10^{-5}. The training is stopped after 40K iterations. For joint training we reduce the loss weight for the spatial task by a factor of 0.1. In the auxiliary tasks, we use $\tau^+ = 4$ and $\tau^- = [8, 16]$ as well as $\sigma^+ = [0.65, 0.95]$ and $\sigma^- = [0.25, 0.55]$. For mining repetitions, we follow the procedure described in Sect. 3.3.3 and iterate it two times to collect about 15000 frames with repetitive poses. We find that two iterations are sufficient, since our method has found most of the repetitions by this time. For testing, we use the pairwise Euclidean distance of `Pool5` features as a similarity measure between images.

3.4.2 Curriculum Details

The curriculum, that we employ for training the spatial and temporal tasks on the Olympic Sports dataset, is visualized as a histogram in Figure 3.4. The histogram bins correspond to the training data which is used at different stages of the curriculum. As described in Sect. 3.3.2, the difficulty of training data increases from stage to stage. It is measured by the fg/bg ratio of estimated optical flow. The figure visualizes the number of data samples as well as the absolute frequencies of the sixteen Olympic Sports categories in each stage.

The gradual augmentation of training data according to the curriculum visibly changes

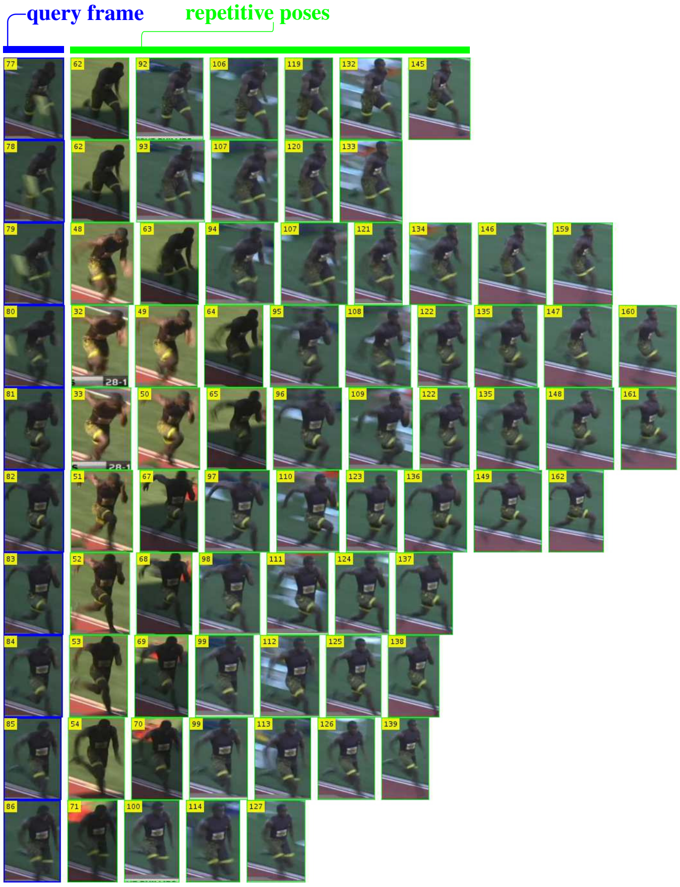


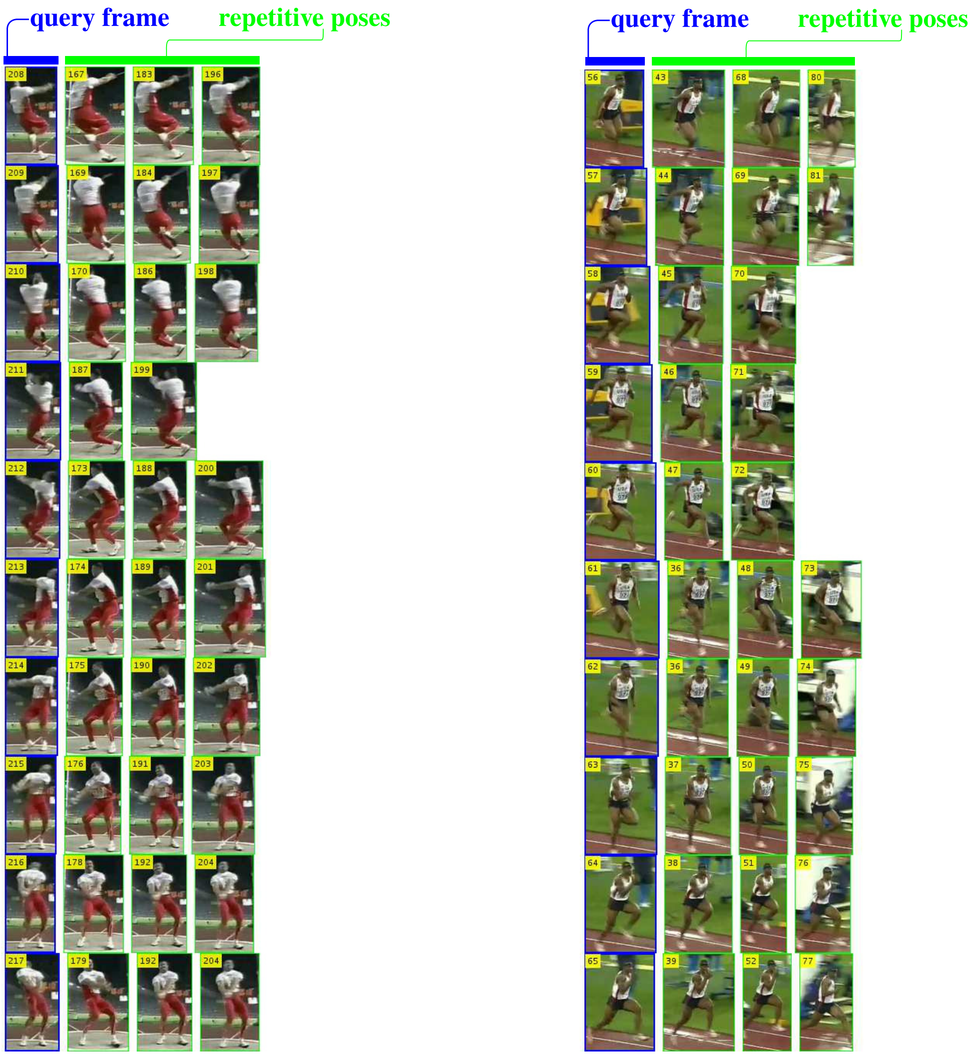
Figure 3.5: Groups of repetitive poses mined from a single video in the Olympic Sports dataset.

the distribution of the categories across stages. This is due to the varying difficulty of the categories: 1) Videos in categories like hammer throw usually contain large clips with clearly visible motion which are already included in the early stages of the curriculum. 2) Videos in categories like clean-and-jerk as well as shot-put largely consist of slow motion parts with less obvious differences between nearby frames and thus are more prominent in the later stages of the curriculum.

3.4.3 Examples of Mined Repetitions

In Figures 3.5, 3.6a and 3.6b, we show examples of repetitive poses that our repetition mining procedure is able to locate inside individual videos as described in Sect. 3.3.3. Each row of images shows a query frame and its repetitions, which are retrieved by our method. The column of query frames and each row of repetitive poses are sorted according to frame numbers that are shown inside the yellow boxes.

Our self-supervised embeddings detect groups of repetitions in spite of changes in lightening, camera angle and background. In the fourth and fifth row of Figure 3.5, it is apparent how the camera angle and background can significantly change over time. While our method sometimes detects up to ten repetitions, usually there are only two to three repetitions.



(a) Hammer throw. **(b)** Long jump.

Figure 3.6: Groups of repetitive poses for hammer throw (a) and long jump (b). Each subplot is mined from a single video in the Olympic Sports dataset.

3.4.4 Ablation Experiments on Posture Analysis in Olympic Sports Dataset

We demonstrate on the Olympic Sports dataset, how different configurations of our method affect the performance of the learned pose embeddings. For the evaluation on the the Olympic Sports dataset, we adopt the posture analysis benchmark proposed in [19]. It consists of 1200 exemplar postures for each of which ten positive (similar) and ten negative (dissimilar) poses are defined. The performance is determined by the ability of the pose embeddings to separate positives from negatives and measured in terms of the area under the curve (AuC) of a ROC.

Task	without curriculum	with curriculum
temporal(T)	0.592	0.630
temporal& spatial	0.664	0.679
temporal(T)$*$	0.762	0.781
temporal& spatial$*$	0.767	0.784

Table 3.1: Average AUC in Olympic Sports benchmark shows effect of curriculum training. Methods with ($*$) are initialized with Imagenet pre-trained weights.

Positive τ^+	Negative $[\tau^-_{min}, \tau^-_{max}]$	Avg. AUC
3	[8, 16]	0.768
4	[8, 16]	**0.781**
5	[8, 16]	0.769
4	[6, 12]	0.776
4	[10,18]	0.778

Table 3.2: Positive and negative sampling ranges in temporal ordering task.

Curriculum Learning

First, we study the impact of curriculum learning. We train our temporal (T) and temporal & spatial (ST) tasks once with and once without a curriculum, but using the same amount of training data. The experiments in Table 3.1 show that the curriculum as proposed in Sect. 3.3.2 improves the performance of our method by 5% in mean AUC in a randomly initialized temporal task. When the temporal task is initialized with Imagenet pre-trained weights, it improves by 2%, and this improvement is preserved even in joint learning of temporal ordering and spatial placement. Temporal ordering itself seems less powerful than spatial placement and cannot be learned without curriculum learning. However, our *fg/bg ratio* based curriculum significantly increases its performance in posture analysis.

Temporal Ordering Task

Second, we study the sensitivity of the temporal ordering task to different configurations. In particular, we vary the ranges τ^+, τ^- for sampling positive and negative pairs as describe in Sect. 3.3.1 and train a network using only the modified temporal ordering task. Table 3.2 shows that the best setting for the two sampling ranges is $\tau^+ = 4$ and $\tau^- = [8, 16]$.

Our understanding is that the difficulty of the tasks is influenced by τ^+. The larger this parameter, the more variation in the sampled positive class has to be accounted for. The selection is therefore a trade-off: A small τ^+ results in little variation and potentially overfitting, a large τ^+ results in more variation and training might show little or no convergence. The range τ^- behaves similarly, but has a smaller impact on the performance.

Auxiliary Tasks and Repetition Mining

Third, we analyze the contributions of repetition mining and the two individual auxiliary tasks in Table 3.3. Temporal ordering underperforms with respect to spatial placement when initialized with random weights. We argue that temporal ordering is a more challenging

Method	Avg. AUC
temporal(T)	0.630
spatial(S)	0.668
temporal & spatial	0.679
T with repetitions	0.658
S&T with repetitions	**0.701**
HOG-LDA	0.580
Doersch *et al.* [57]	0.580
Jigsaw puzzles [158] (Imagenet)	0.653
Jigsaw puzzles [158] (OSD)	0.646
Shuffle&Learn [153]	0.646
Video triplet [220]	0.598
Alexnet [120]	0.722
temporal*	0.781
spatial*	0.756
temporal & spatial*	0.784
T with repetitions*	0.794
S&T with repetitions*	**0.804**
CliqueCNN* [19]	0.790

Table 3.3: Comparative posture analysis performance of auxiliary tasks in Olympic Sports dataset. Methods with $(*)$ are initialized with Imagenet pre-trained weights of Alexnet.

task, since the temporal nature of actions has to be learned by the network. When the network is initialized from Imagenet, temporal ordering performs well. It has already learned to filter relevant visual information and improves with additional temporal cues from videos. On the other hand, spatial placement does not improve on Alexnet by such a large margin, because pre-trained Alexnet already comes with a good localization ability. In both settings (initialized randomly or from Imagenet pre-trained weights), repetition mining further boosts performance. This improvement highlights the benefit of the usage of repetitions.

Additionally in Table 3.3, we compare our best performing method with related work. When randomly initialized, our method performs better than several different self-supervised methods [57, 158, 220] and surpasses the best competitor by nearly 5 points. It even approaches the performance of the Imagenet pre-trained Alexnet, which is impressive considering that our training leveraged 680 sport videos (approx. 80K frames used) without labels, whereas Imagenet contains 1.2M labeled images. In the case of fine-tuning, our model improves about 8 points on ImageNet pre-trained Alexnet and surpasses CliqueCNN [19] which is trained in the same setting.

3.4.5 Pose Retrieval

In this set of experiments, we want to study the ability of our trained pose embeddings to generalize to unseen datasets. For this purpose, we evaluate our methods in the task of pose retrieval on the challenging MPII Human Pose dataset. We adopt the same procedure as described in [124], and split the fully annotated MPII training set into a train and

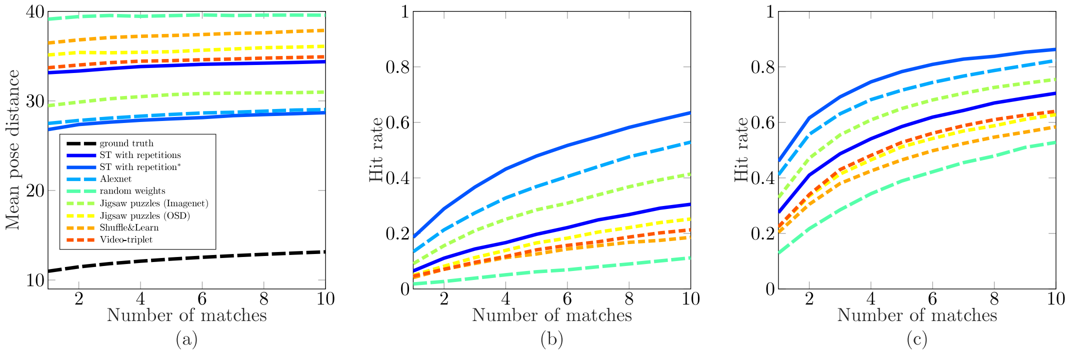


Figure 3.7: Pose retrieval results on MPII validation set: (a) Mean pose distance, (b) Hit rate@K using nearest neighbor criterion, (c) Hit rate@K using relative distance criterion. Model with $(*)$ initialized with Imagenet pre-trained weights.

validation set. The validation set is further split in 1919 images for query and 8000 images for test purposes. The input images and pose annotations are normalized with respect to the smallest square patch tightly enclosing all body part locations, and normalized to the input size of our network.

According to [124] three performance metrics are used: *mean pose distance*, *hit rate using nearest neighbor* and *relative distance criterion*. The pose distance is the mean of Euclidean distances between normalized pose vectors. The mean pose distance is computed across the first K nearest neighbors. The hit rate measures the correctness of retrieval and is defined in two different ways: 1) *nearest neighbor criterion* determines whether at least one retrieval among the K nearest neighbors belongs to the first fifty nearest neighbors in the pose space. 2) *relative distance criterion* uses a +10 margin of minimum pose distance between query and test set.

The pose retrieval results evaluated on the three performance metrics on MPII are shown in Fig. 3.7. Here, we trained our method on the spatial&temporal (ST) tasks with repetition mining using OSD only. It successfully generalizes to the challenging MPII dataset. When randomly initialized, it shows better mean pose distance and hit rate performance than previous methods, which are also trained on videos [153, 220].

When the jigsaw puzzle method [158] is trained on the larger Imagenet dataset, they clearly outperform our method. We argue that this performance gap is due to different training data. To support this assumption, we re-train their method on OSD person boxes using their official implementation [1], and find it to perform worse than our self-supervised method across all measures.

When initialized from Imagenet pre-trained weights, our method outperforms Alexnet across all measures, particularly in terms of hit rate.

In Figure 3.8 we show qualitative results for the task of pose retrieval on the MPII Human Pose dataset. We compare Alexnet with our best performing model. Even though our method is only trained on Olympic Sports videos (not MPII images) and without any pose labels, our method shows favorable retrievals. The last row shows a typical failure cases due to occlusions.

[1] https://github.com/MehdiNoroozi/JigsawPuzzleSolver

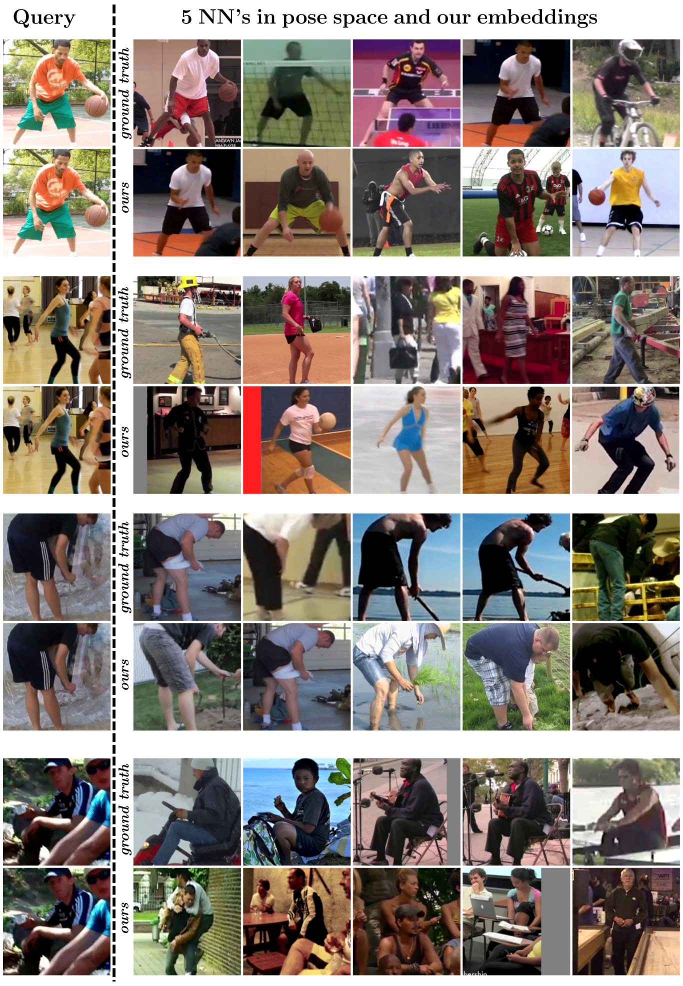


Figure 3.8: Pose retrievals for four query images on the MPII Human Pose dataset. We retrieve five nearest neighbors using either the ground truth by computing Euclidean distances in pose space, or using our self-supervised pose embeddings. The last row illustrates a failure case related to occlusions.

3.4.6 Pose Estimation

For pose estimation we evaluate on the Leeds Sports Pose dataset [112]. We follow the procedure described in [19] and use the 1000 training images and 3938 (fully annotated) images from the extended training set as test set for retrieval while the original 1000 test images are used as query. In both query and test images, joint locations are normalized into our network's input size.

We report the Percentage of Correct Parts (PCP) measure [68] on 14 body joints for different methods. According to PCP a part is considered correct, if its endpoints are within 50% part length of the corresponding ground truth endpoints.

Unsupervised pose estimation results of LSP in Table 3.4 show that our method, when initialized randomly, performs better than other self-supervised methods except for jigsaw puzzles trained on Imagenet. As in the case of pose retrieval, we argue that it is due

Method	*Head*	*Torso*	*U.arms*	*L.arms*	*U.legs*	*L.legs*	*Mean*
random weights	19.3	45.2	9.6	4.1	21.1	20.3	19.9
ground truth	72.4	93.7	58.7	36.4	78.8	74.9	69.2
Chu *et al.*[46]	89.6	95.4	76.9	65.2	87.6	83.2	81.1
Shuffle&Learn [153]	36.7	66.6	20.1	8.3	37.5	35.0	34.0
Video triplet [220]	40.5	76.6	23.9	10.0	46.1	39.6	39.4
Jigsaw puzzles [158] (Imagenet)	49.3	80.1	27.5	11.9	50.5	47.4	**44.4**
Jigsaw puzzles [158] (OSD)	41.0	72.8	23.8	12.2	43.0	39.8	38.7
S&T with repetitions	40.3	74.7	23.8	11.5	45.8	42.8	39.8
Alexnet [120]	42.4	76.9	47.8	41.8	26.7	11.2	41.1
CliqueCNN [19] *	45.5	80.1	27.2	12.6	50.1	45.7	43.5
S&T with repetitions*	55.8	86.5	35.0	18.9	58.7	53.8	**51.5**

Table 3.4: Pose estimation results in Leeds Sports Pose dataset with PCP measures for each method. Methods with (∗) are initialized with Imagenet pre-trained weights.

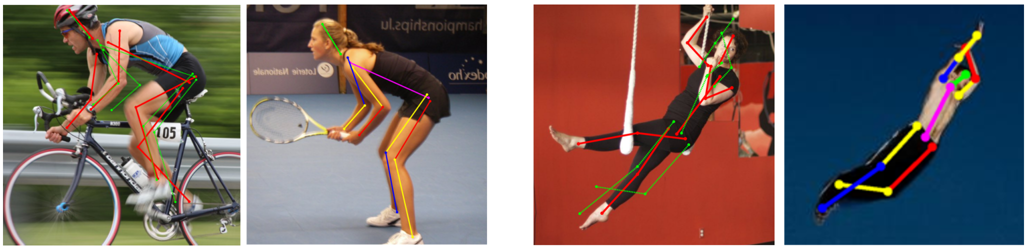


Figure 3.9: Pose estimation results in Leeds Sports Pose dataset. First images are from test set with the superimposed ground truth skeleton depicted in red and the predicted skeleton in green. Second images are corresponding nearest neighbors.

to the size of Imagenet. When initialized from pre-trained weights, our method clearly outperforms [19, 120].

Some qualitative samples from the query set together with their nearest neighbors are shown in Fig. 3.9. Our method is able to retrieve similar poses even if the query is very different from our training data.

In addition to our unsupervised experiments, we use our pose embeddings as an initialization of the supervised DeepPose [211] method. In total, we evaluate four different initializations of [211] on the MPII dataset: (i) our randomly initialized spatial&temporal (ST) with repetitions model, (ii) Shuffle&Learn [153], (iii) random initialization, and (iv) Imagenet pre-trained Alexnet [120].

For all initializations, we train the DeepPose method using the same setup and evaluate using PCKh@0.5 metric as shown in Table 4. Our method shows an improvement of 7.9% and 4% compared with random initialization and Shuffle&Learn, respectively. It is only 4.7% below Alexnet, which is learned using the labels of 1.2 million images.

	Ours	Shuffle&Learn [153]	Random init.	Alexnet[120]
Head	82.6	75.8	79.4	87.2
Neck	90.3	86.3	87.1	93.2
LR Shoulder	79.5	75.0	71.6	85.2
LR Elbow	62.8	59.2	52.1	69.6
LR Wrist	47.1	42.2	34.6	52.0
LR Hip	75.5	73.3	64.1	81.3
LR Knee	65.3	63.1	58.3	69.7
LR Ankle	59.5	51.7	51.2	62.0
Thorax	90.1	87.1	85.5	93.4
Pelvis	80.3	79.5	70.1	86.6
Total	73.3	69.3	65.4	78.0

Table 3.5: PCKh@0.5 measure for DeepPose method [211] on MPII Pose benchmark dataset comparing different initialization approaches.

3.5 Discussion

In this chapter, we have proposed two complementary self-supervised tasks, temporal ordering and spatial placement, which are trained jointly on unlabeled video data. To boost self-supervised training, we have introduced a motion-based curriculum and a procedure for mining repetitive poses and using them as valuable training data. Our pose embeddings capture the characteristics of human posture, which we have demonstrated in experiments on pose analysis. In the Olympics Sports dataset, the learned representation decreases the gap between self-supervised methods and Imagenet supervision, and fine-tuning with our self-supervised approach significantly improves the performance of models pre-trained on Imagenet. Finally, we have shown that the trained embeddings are able to generalize to unseen datasets in pose analysis without fine-tuning.

While our approach learns a representation of human pose without requiring body part annotations, it relies on videos where the actions of a single person are closely followed by a camera (i.e. sports videos). Strictly speaking, this amounts to a form of weak supervision that is necessary for our approach to work properly. One way to overcome this limitation would be the application of an off-the-shelf object detector or person tracking algorithm, that allows to extract the bounding boxes of a person across video frames, largely independent of the camera motion and focus.

Repetitions in videos are a rich source of supervision for learning that can be exploited through self-similarity, as we have demonstrated with our repetition mining method. However, spotting repetitions in videos is not always easy. To deal with this problem, one option is to specifically collect videos for learning that contain many easy-to-spot repetitions. While this might require additional weak supervision, it is still preferable to the time-consuming manual annotation. Another option is to increase the chance of finding a repetition across video frames by searching for self-similarity in the parts rather than the whole object. Repetitions in the parts of an object occur more frequently and still provide rich information for learning.

While self-supervised learning automatically generates supervision, the quality of the supervised data is often fluctuating. To separate valuable from ambiguous training samples, we have proposed a curriculum learning strategy as well as a repetition mining method in this chapter. We believe that such a mechanism for self-correction during the learning process is an important part of self-supervised learning that deserves further research. Ultimately, the knowledge about the quality of self-supervised data also supports the construction of a more balanced curriculum that includes difficult or potentially ambiguous samples (e.g. not only sports categories with many repetitions) that only turn into valuable samples over the course of training.

CHAPTER 4

Reading Cuneiform Script: A Challenging Visual Recognition Problem for Human and Machine

Cuneiform clay tablets are the oldest written records of humankind and thanks to their robustness still prevalent. Reading cuneiform script is time-consuming and difficult, even for experts in the field of Assyriology. The goal of the following chapters is to leverage the recent advances in the field of computer vision and machine learning to support Assyriologist in their analysis of cuneiform script. In this chapter we provide an overview of cuneiform script, its historical background, and the challenges of its scientific analysis today. We conclude this chapter with an analysis of the problem of reading cuneiform script from a computer vision perspective. For a more comprehensive introduction to the topic of cuneiform script, we would like to point the reader to the relevant Assyriology literature [38, 69, 174].

4.1 Historical Background

4.1.1 Writing System

Used in most of the ancient Near East for a period of over three thousand years until the beginning of the Current Era, the cuneiform script is the oldest known writing system in the world, providing us with invaluable records of early human history across all spheres of life.

Most cuneiform texts were written on palm-sized clay tablets by impressing an angular stylus that left a wedge-shaped mark on them [33]. Groups of wedges (*cunei* in Latin, hence the name cuneiform) formed *cuneiform signs*[1], of which several hundred have been documented in cuneiform sign lists (e.g. [29]). The wedge-shaped cuneiform signs were optimized for writing in soft clay which was a readily available material in the ancient Near East. To save space on the mostly palm-sized clay tablets, signs were written tightly packed together without word separation (no white-space).

[1] In the context of cuneiform script, it is common to speak of cuneiform signs rather than symbols or characters.

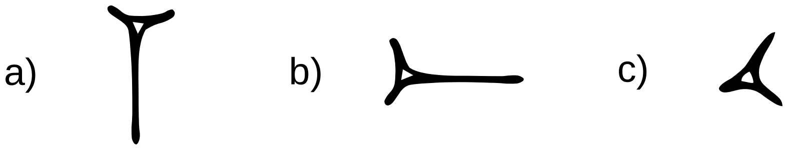


Figure 4.1: Drawing of the three types of wedges used in Neo-Assyrian cuneiform: (a) standing , (b) lying, and (c) oblique (Winkelhaken).

Cuneiform script was used as a writing system for several languages, most importantly Sumerian and later Akkadian, which has two main dialects: Assyrian and Babylonian. As cuneiform script was invented for Sumerian and then later adopted for Akkadian, cuneiform script is often written as mixture of these two languages. From a linguistic perspective, cuneiform script is characterized by the following defining features:

1. Cuneiform script is a logo-syllabic writing system, i.e. signs are used to express logograms (words) or syllables (parts of words). In contrast to an alphabetic writing system, over 900 different signs have been documented for cuneiform script.
2. A sign can have multiple readings, i.e. multiple syllabic or logographic meanings (polyvalence).
3. A sound (syllable) can be written with different signs (homophony).

Since one sign can have multiple readings and different signs can have the same reading, cuneiform script might appear impossible to read. Fortunately, the reading of cuneiform script is significantly simplified due to the presence of determinatives (signs that indicate the meaning of other signs), other writing conventions, and most importantly due to the context of the writing.

4.1.2 Development and Usage

Invented over 5000 years ago in the late 4th millennium BCE as a bookkeeping tool, cuneiform script continued to be used into the first century CE. Experts distinguish several important periods of its use like Early Dynastic, Old Babylonian and Neo-Assyrian [69]. Over the course of millennia cuneiform script evolved from a script with pictographic signs that are more drawn than written to a mature script with standardized wedge-shaped signs for efficient writing on clay tablets. Over time the standard direction of writing became left-to-right and top-to-bottom, i.e. text written in lines as common today. As the standard tool for writing an angular stylus made from reed (sometimes bone) was developed. In 2nd millennium BCE Sumero-Akkadian cuneiform (including Old Babylonian cuneiform) only five basic wedges remained in use for assembling cuneiform signs: horizontal (lying), vertical (standing), left falling, right falling, and Winkelhaken. Later in the early 1st millennium BCE the cuneiform script was even further simplified so that Neo-Assyrian cuneiform signs only consisted of three basic wedges: horizontal (lying), vertical (standing), and Winkelhaken as shown in Fig. 4.1. Due to the development of the writing system, each sign may have multiple variants which need to be distinguished. In this thesis we will focus on the cuneiform script used in the Old-Babylonian (c.2000 – c.1600 BCE) and the Neo-Assyrian period (c.900 – c.600 BCE).

The usage of cuneiform script expanded over time from purely administrative purposes to many applications like writing personal letters, religious stories, mathematical equations or medical procedures. Cuneiform literacy became more widespread in the 2nd and 1st millennia, with the ability to read not only reserved to professional scribes and wealthy families [38, 174].

4.2 Scientific Study

Today reading cuneiform script on age-old clay tablets is a scientific study, which requires years of training in the field of Assyriology. While there is only a small number of experts, cuneiform clay tablets represent the most numerous written resource for the study of ancient history, only surpassed by ancient Greek which of course is far younger. In contrast to script written on papyrus or paper, clay tablets are more resilient to the passing of time and so far over 500'000 such tablets have been found [5]. Only about half of the discovered tablets have been thoroughly studied (amounting to a text corpus of 14 million words) and many more still remain in the ground to be recovered in the future.

4.2.1 Assyriology

Assyriology is the archaeological, historical and linguistic study of ancient Mesopotamia. A central task of Assyriologists is the study of cuneiform clay tablets, as they represent an unfiltered record of the activities, thoughts and stories of people of this time. This has lead to numerous remarkable findings that have changed our perception of the ancient history (e.g. the Gilgamesh flood myth [50, 147]). While clay tablets provide an invaluable information source, reading cuneiform requires time and effort, also after years of practice.

4.2.2 Reading Comprehension

Decipherment As the knowledge of cuneiform was lost for almost two thousand years, its decipherment[2] in 19th century is an impressive achievement. It involved the deciphering of the writing system (re-discovery of the meaning of the cuneiform signs) as well as the reconstruction of two dead languages [69, 78]. Despite its decipherment, reading cuneiform script is challenging for several reasons which we will explain in the following.

Challenges of Reading Back in the day, the ability to read basic cuneiform texts like letters only required a working knowledge of less than 200 cuneiform signs. Nowadays Assyriologists, the cuneiform specialists of our time, have to learn two dead languages (Sumerian and Akkadian) in addition to familiarizing themselves with the cultural and historical background of the relevant period.

[2] Deciphering ancient scripts is often referred to as codebreaking. However, there are important differences in deciphering ancient writing and secret writing (cryptography). Most methods of decipherment exploit the redundancy of text, i.e. certain combinations of signs are more frequent than others. To increase the security of a code, linguistic redundancy is deliberately reduced in secret writing . In contrast, redundancy in written or spoken language increases the likelihood that the message content can be fully recovered (since deciphering the message does not rely on receiving every utterance or sign), e.g. sometimes the first or last letter of a word and its length are sufficient. Clearly, different intentions shape the design of languages and cryptographic codes or ciphers. For a more detailed exploration of this topic, Gelb [78] provides a good introduction.

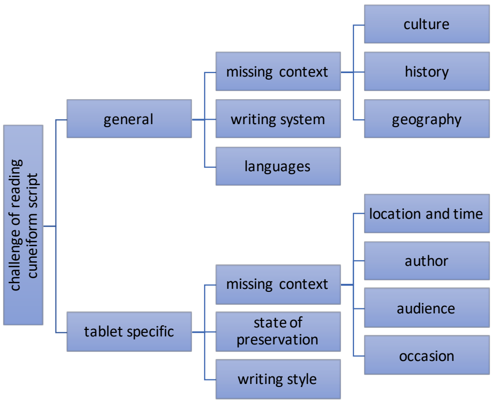


Figure 4.2: Reading cuneiform script is a challenging task requiring both general knowledge and practical experience when dealing with a specific clay tablet.

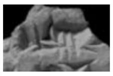
ID	Thumbnail	Sign	Name	Readings										
001		𒀸	asz	aš	as$_3$	rum	ru$_3$	rim$_5$	ina	dil	dili	ṭil	dal$_3$	in$_6$
003		𒄬	hal	ḫal										
005		𒁄	bal	bal	pal	bul$_3$	pul$_3$	bala						
006		𒄈	gir2	gir$_2$										

Figure 4.3: Excerpt of a sign list based on the MZL by Borger [29] with four different cuneiform sign code classes. For each sign code class, we report its MZL number (ID), an example image (Thumbnail), its standardized shape according to an Unicode font (Sign), and its recorded readings.

Reading cuneiform is a difficult task for a number of reasons as visualized in Fig. 4.2 and requires both general knowledge as well as practical experience. An obvious prerequisite is the knowledge of the writing system in its various stages of development and language proficiency of the encoded languages. Additionally, knowledge about the history, geography, and culture of the relevant periods is essential, as it allows to identify names (e.g. of kings, places, or events). While the study of Assyriology prepares a reader, practical experience is as important when dealing with a specific tablet. In particular, knowledge about similar texts, genres, and writing styles can be essential, as it provides cues about the correct reading, resolves ambiguities, fills in gaps, or verifies specific interpretations.

Sign Lists Sign lists are an essential tool of Assyriologists in their analysis of cuneiform texts. Similar to a dictionary, a sign list records for each sign all known readings which is very helpful due to the extensive polyvalence of cuneiform script. The "Mesopotamisches Zeichenlexikon" (MZL) by Borger [29], a modern sign list, comprises over 900 different signs, where each sign is uniquely identified by *sign code* also known as *MZL number* that corresponds to a *sign code class* as shown in Fig. 4.3. This index of sign code classes simplifies the exchange of cuneiform texts especially in the digitized version, since the content of a cuneiform tablet can be uniquely encoded with the help of sign codes.

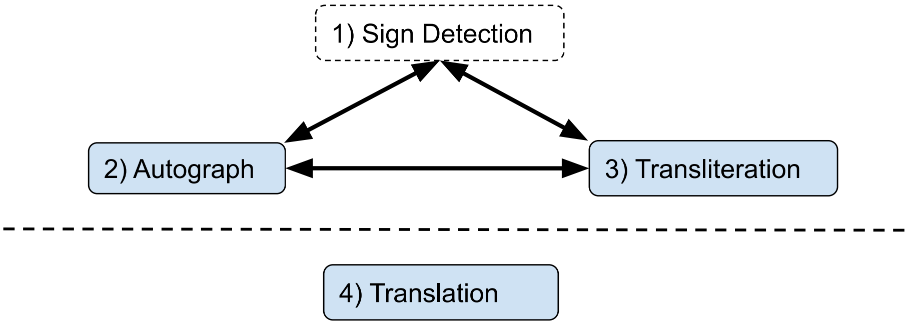


Figure 4.4: Standard process of Assyriology to study a cuneiform tablet. Standardized outputs in blue.

Besides categorizing cuneiform signs, sign lists also establish a standard for the readings of cuneiform signs by defining a system of subscripts and accents in order to distinguish homophone readings.

Assyriological Process The field of Assyriology has established a general process for the study of cuneiform tablets as visualized in Fig. 4.4. This process produces three standardized outputs for each clay tablet (highlighted as blue boxes): Autograph, transliteration, and translation. This common process and its standardized outputs are essential for the dissemination of new findings in the field of Assyriology.

During the main phase of the analysis (visualized as the process above the dashed line in Fig. 4.4), an Assyriologist leverages all information available in the clay tablet to recover the meaning of the cuneiform text. In an initial reading, an Assyriologist first identifies groups of wedges as individual cuneiform signs and determines for each identified sign its sign code class, a classification according to a common sign list. This amounts to an initial sign detection, i.e. localization and classification of cuneiform signs. To better understand the cuneiform text and update the initial reading accordingly, the Assyriologist then iterates between two steps combining epigraphic and linguistic findings: First, in an *autograph* the cuneiform signs are mapped from 3D to a 2D drawing in a wedge-by-wedge manner. Second, in a *transliteration* for each cuneiform sign one of several possible readings (in the original language) is recorded in Latin script. Due to the special design of the transliteration system, it is possible to retrieve the original sign code class of each sign from its reading. If we mention transliterations in the remainder of this article, we will refer to this sign-by-sign representation in terms of sign code classes rather than readings. In the secondary phase, the creation of a translation adds further details to the interpretation of the text. In contrast, the sign detection is not preserved, as a scholar can reconstruct it from transliteration and autograph.

4.3 Reading Cuneiform Script from a Computer Vision Perspective

As we have discussed in the previous section, reading cuneiform script is a difficult task that involves a time-consuming analysis with several steps. In the following, we want to

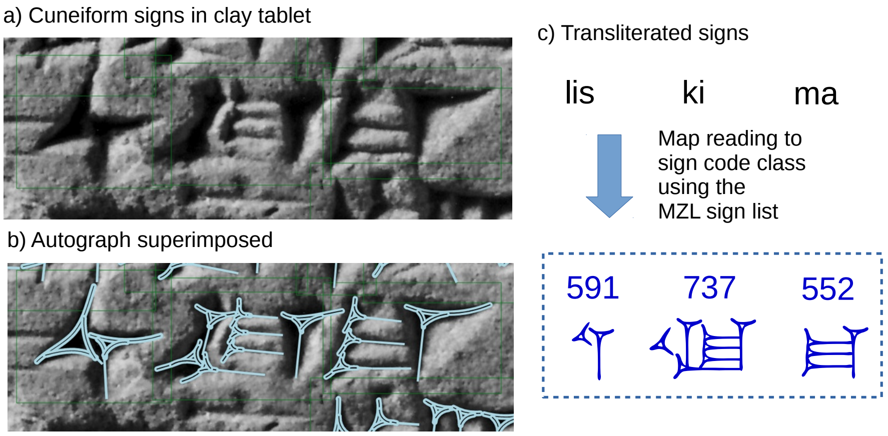

a) Cuneiform signs in clay tablet
b) Autograph superimposed
c) Transliterated signs
lis
ki
ma
Map reading to sign code class using the MZL sign list
591
737
552

Figure 4.5: The Assyriological process takes an image of cuneiform writing (a) and produces an autograph (b) as well as a transliteration (c). The transliteration provides the reading of the cuneiform writing (first row) and implicitly encodes which signs were used in the original clay tablet (dashed box). The two blue rows are derived from the transliteration by relying on the MZL by Borger that maps the reading of a sign to its sign code class.

study the problem of reading cuneiform script from a computer vision perspective. While computer vision might be able to deal with all steps in the process eventually, in this work we want to focus our efforts on the first step of sign detection.

4.3.1 Cuneiform Sign Detection

From a computer vision perspective, cuneiform sign detection is a fine-grained object detection task with several hundred object classes, i.e. sign code classes. Sign detection consists of two subproblems: 1) localization (i.e. draw a bounding box around a sign) and 2) classification (i.e. assign a sign code class according to a sign list). Even though cuneiform sign detection is only conducted implicitly by Assyriologists, it is a fundamental step in reading cuneiform script that immediately supports all other steps of the process.

4.3.2 Challenges of Cuneiform Sign Detection

While in many writing systems the recognition of individual characters is not an issue, detecting cuneiform signs is difficult for several reasons:

- In many scripts white-space between characters simplifies detection significantly, because it allows to separate localization and classification in two consecutive steps. However, cuneiform signs are often inscribed without space between neighboring signs, making their localization and classification interdependent [99].
- Sign similarity of over 900 different sign code classes increases the difficulty further as shown for two classes in Fig. 4.6: Since most cuneiform signs are constructed from very few types of wedges (in Neo-Assyrian: lying, standing, and diagonal) that are combined in various spatial configurations, different sign code classes are often very similar (high inter-class similarity). Meanwhile cuneiform signs of the same sign code

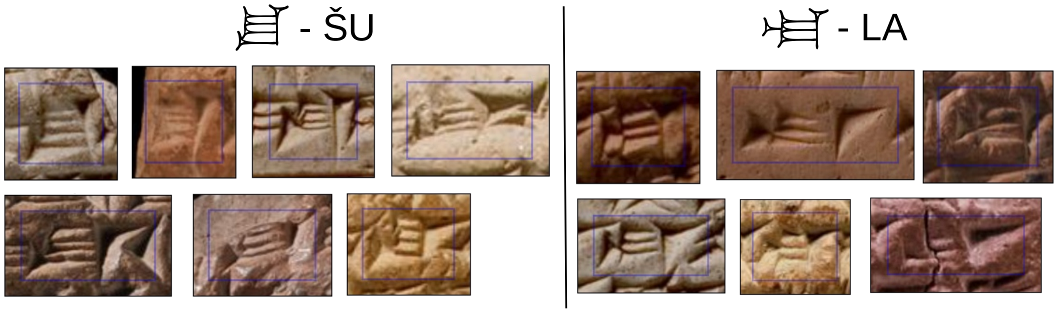


Figure 4.6: The problem of fine-grained sign similarity: Detecting individual signs on a cuneiform tablet usually requires the expertise of an Assyriologist. Cuneiform signs of different sign code classes may look very similar, while signs of the same sign code class may look very different.

class can be very dissimilar in their appearance if written by different scribes (high intra-class variance).

- The frequency of cuneiform signs follows a Zipfian distribution (a discrete Pareto distribution) as shown in Sect. 5.6.2. This exponential distribution causes a strong imbalance in the sign code class frequency, where only a few sign code classes make up the majority of cuneiform signs in clay tablets, while the other sign code classes are rare. Thus it is difficult to obtain sufficient training data for all sign code classes.
- Many tablets have been damaged over the course of millennia and thus complicate sign detection further. The damage appears in various forms: surface damage, cracks, holes, and missing pieces. Often tablets have to be reconstructed from broken pieces.
- Cuneiform script consists of three-dimensional wedges written on the surface of three-dimensional clay tablets. As we usually have to rely on two-dimensional photographs of a clay tablet, lighting and visual distortions impact the detection of individual signs.
- The process of writing into soft clay tablets causes already inscribed signs to be deformed by the following signs in their proximity. This plastic deformation is unique to cuneiform script and further increases the difficulty of distinguishing signs, as wedges might get lost in the process of writing.
- Finally, standard computer vision and machine learning approaches for object detection require large amounts of supervised data. In the case of sign detections, this amounts to ten thousands of bounding box annotations of cuneiform signs. As Assyriologists only perform sign detection implicitly, there are no bounding box annotations available.

4.3.3 Potential Sources of Inductive Bias

As with any writing system, cuneiform script is governed by a set of rules that each scribe needs to learn. This task-specific prior knowledge helps a scribe to produce valid cuneiform script, that can be understood by other cuneiform readers. These rules literally serve as a regularizer for the process of writing. Similarly, we can exploit this prior knowledge of the rules as inductive bias for a machine learning algorithm to tackle the problem of reading cuneiform script. In the following we provide a list of potential sources of inductive bias for reading cuneiform script:

- The writing system provides a rich structure: Cuneiform text is mostly organized in parallel lines (from left to right), signs all face in the same direction (there is one ”up”), the rectangular surface of a tablet provides page boundaries, and all signs on a tablet have roughly the same scale.
- The language of the information encoded in cuneiform script is highly structured itself. Constraints derived from the syntax, semantics and grammar of the encoded language can be potentially exploited to support sign detection. Thus only a small subset of all possible combinations of cuneiform signs actually encodes meaningful information.
- Assyriologists have already analysed a large corpus of cuneiform tablets for which images and transliterations are available in digitized form. While this dataset does not support standard supervised learning, its size has the potential to enable other forms of learning.
- Transliterations as part of the standardized output of the Assyriological process are readily available. While they do not provide the precise location of cuneiform signs in a tablet image, they at least provide the relative position of cuneiform signs (grouped in lines and sometimes even information about damage).
- Less than 200 cuneiform signs are sufficient to understand most cuneiform texts. Thus, learning to detect the 200 most common sign code classes (instead of over 900) allows us to deal with most texts. This also reduces the negative impact of the sign code class imbalance described earlier.
- While the structure of signs requires fine-grained detection, the variance in appearance is strongly regularized by the fact that all signs are constructed from mostly three types of wedges. Thus learning the ability to recognize a wedge in one sign might be reusable for multiple other signs.

CHAPTER 5

Building a Dataset for Cuneiform Sign Detection

Cuneiform sign detection is the first fundamental step in reading cuneiform script (cf. Sect. 4.3). The development of a computer vision algorithm for cuneiform sign detection requires a task-specific dataset that is sufficient to support its training, validation, and testing. In this chapter we describe the creation of the first large-scale dataset for cuneiform sign detection. Such a dataset is integral for our approach to learn a cuneiform sign detector introduced in the next chapter. Besides enabling our research, we also provide a general benchmark for fine-grained sign detection that facilitates further research in the domain of cuneiform script. Moreover, the described procedure can be applied to create additional datasets for cuneiform script from different periods.

In the following, we first explain the major design decisions with respect to the cuneiform sign detection task that are fundamental to the creation of this dataset. Then, we describe the data collection from different online sources and the comprehensive data preparation to extract pairs of tablet images and transliterations. In a third step, we describe the manual annotation of a small subset of the dataset, essential for quantitative validation and testing. Finally, we report some general statistics of the new dataset for cuneiform sign detection and describe how the dataset is split to form the cuneiform sign detection dataset used in the next chapters.

5.1 Design Decisions

While the design of a cuneiform sign detection task is primarily driven by the (un)availability of data, we highlight three important design decision that have a strong impact on the creation of the dataset.

5.1.1 Neo-Assyrian Cuneiform

Cuneiform script has evolved over a period of three thousand years, passing through various stages of development as we describe in the previous chapter. We concentrate the scope of our dataset for cuneiform sign detection on the Neo-Assyrian stage of cuneiform writing (c. 900 BCE - 600 BCE).

By focusing on a single cuneiform script, we avoid an additional source of ambiguity during cuneiform sign detection which arises from the various variants of cuneiform signs that evolved over time. Additionally, Neo-Assyrian cuneiform represents a stable and advanced version of cuneiform script with a standardized form of writing, which appears especially suitable for a machine learning approach.

During the Neo-Assyrian period, the cuneiform script was widely used in Assyria and texts were collected in various libraries, most importantly Assurbanipal's library, which comprised most of the canonical Mesopotamian literature. Fortunately, many texts have been recovered and analyzed by Assyriologist in the last 150 years providing a rich corpus of texts for the creation of a dataset.

A more practical reason for the focus on Neo-Assyrian script is our collaboration with Prof. Stefan Maul from the institute of Assyriology at Heidelberg University, who is also an expert for the Neo-Assyrian period. Besides advising us from an Assyriologist perspective, he kindly granted us access to Neo-Assyrian cuneiform texts and provided support for the manual annotation of cuneiform tablets which we will describe later.

5.1.2 2D Images of 3D Clay Tablets

Another important design decision in the creation of the cuneiform sign detection dataset is the usage of 2D tablet images as input data, that are based on photographs of clay tablets. Cuneiform tablets have been digitized in various forms that strongly differ in their availability and production cost. Cuneiform tablets are best studied in their original form as 3D objects: signs are often found not only on the front and back but also along the edges, and their visibility depends on lighting and viewpoint. Similarly, autographs can be used as input for a detector. However, the creation of 3D scans or autographs is expensive and time-consuming. In contrast, we rely on 2D images of tablets (for each 3D tablet a 2D composite image of the different tablet sides) which are readily available online [49].

5.1.3 Transliteration in place of Bounding Box Annotations

Instead of manually creating new annotations for each tablet image, we rely on the existing transliterations that are an essential part of Assyriologist's work (as mentioned in the previous chapter) and available for thousands of tablets via the *Open Richly Annotated Cuneiform Corpus* (ORACC) websites [161]. While the transliteration provides a sign-by-sign representation of the tablet content, it does not explicitly localize each sign in the 2D tablet image like in the case of bounding box annotations. For evaluation purposes, we also provide manual annotations for a small subset of tablet images which we describe in Sect. 5.3.

5.2 Data Preparation

Guided by the stated design decision, the goal of data preparation is to obtain a dataset for cuneiform sign detection that combines 2D images of Neo-Assyrian clay tablets and their corresponding transliterations. Fig. 5.1 visualizes the individual steps of data preparation. First, we collect the image data and transliterations from different online sources. Then, we perform a comprehensive data preprocessing to produce tuples, each of which consists of a tablet image and a corresponding transliteration. To merge the different data sources into

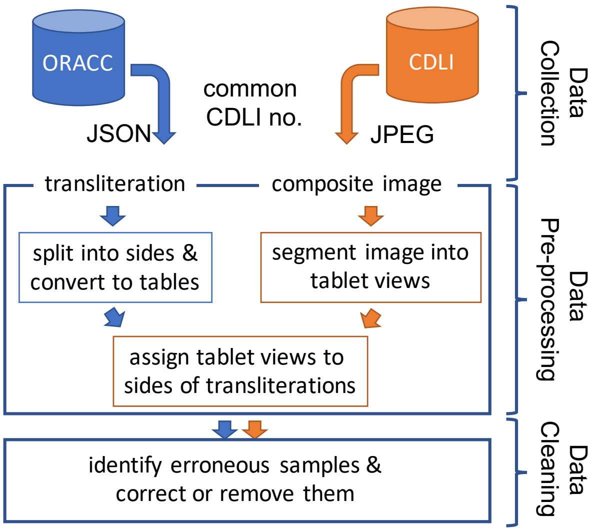

ORACC
CDLI
JSON
common CDLI no.
JPEG
Data Collection
transliteration
composite image
split into sides & convert to tables
segment image into tablet views
assign tablet views to sides of transliterations
Data Pre-processing
identify erroneous samples & correct or remove them
Data Cleaning

Figure 5.1: Data preparation pipeline for the cuneiform sign detection dataset. It comprises three consecutive steps: data collection, pre-processing and cleaning.

a coherent dataset, we need to solve a segmentation and assignment problem described in the following. Finally, the data cleaning step corrects or removes erroneous data samples, e.g. with inconsistent assignment. To facilitate the creation of a large-scale dataset, we automate as many steps of data preparation as possible, while reducing the number of introduced errors.

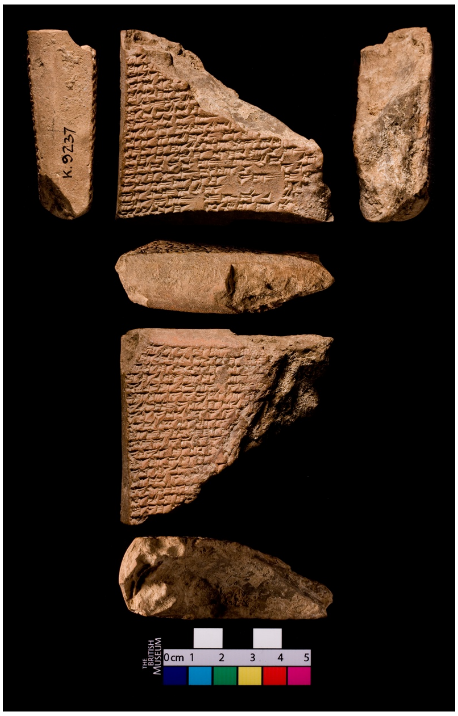
5.2.1 Data Collection

We focus the data collection on the *State Archives of Assyria* (SAA). For the Neo-Assyrian period, the SAA series are invaluable, delivering a diverse corpus of texts, including royal correspondence, divination, and literary texts or international treaties, among other genres. The SAA are organized in 19 collections according to specific genres that are helpful in the creation of a mixed and representative dataset. The dataset of clay tablet images with their transliterations, originates from two different sources:

- Transliterations of Neo-Assyrian tablets found in the *State Archives of Assyria online* (SAAo) [161, 167], as digitized by the ORACC project [208].

- Clay tablet images made available through the *Cuneiform Digital Library Initiative* (CDLI) [49].

We distill our dataset by first extracting the usable transliterations from SAAo and then collating them with the available images in CDLI. Only tablets with an available transliteration and image are included in our corpus. To match images and transliterations, we rely on the CDLI number as a unique identifier that is used to access individual clay tablet entries in both digital libraries. While there exists an open data version of the SAAo project for easy download ([208], the images of clay tablets are individually downloaded from the CDLI server. The CDLI images are identified either by relying on the CDLI numbers retrieved from the SAAo project or by filtering for clay tablets from the SAA



obverse
beginning broken
1'. in#-[...]
2'. si#-im#-[...]
3'. isz#-szak#-[...]
4'. usz-te-s,i# [...]
5'. dal-ha te-re#-[...]
6'. it-ti _{lu2}hal_ u [...]
7'. ina pi-i _e-sir2#_ [...]
8'. at-til ina szat mu-szu [...]
9'. _lugal uzu digir-mesz_ [...]
10'. lib3-bu-usz ik-ka-s,ir-ma# [...]
11'. na-an-za-zi tesz2-lit usz-ta#-[...]
12'. pah-ru-ma ra-man-szu-nu u2-szah#-ha-zu# [...]
13'. szum-ma isz-ten-ma na-pisz-ta-szu2 u2-[...]
14'. i-qab-bi sza2-nu-u2 u2-szat-bi ter-[...]
line preceded by 1(u)
15'. sza2 ki-ma szal-szi qip-ta-szu2 a-tam-ma# [...]
16'. er-ru-ub bit-usz-szu 4(disz){+u2} i-ta# [...]
17'. 5(disz){+szu} pi-i ha-sze-e szu-bal# [...]
18'. 6(disz){+szu} 7{u2} i-rad-du-u sze# [...]
19'. ik#-s,ur-nim-ma ri-kisz se-bet il# [...]

reverse
1. [...] mi#-isz la pa-du-u2 u2# [...]
2. [...] isz-ten _uzu_{mesz}-szu-nu-ma [...]
3. in#?-na-ad-ru#-num-ma na-an# [...]
4. tusz-szu u nap-ra-ku u2 [...]
5. mut2-tal2-lu pi-ia a-pa# [...]
line preceded by 1(u)
6. szap-ta-a-a sza2 it-ta-[x]
7. sza2-pu-tum sza2-gi-ma-ti
8. sza2-qa-a-ti ri-sza-a ik#-[...]
9. lib3-bi kab-ba-ra-a
10. ra-pa-asz2-szi i-ra#-[...]
11. sza-di-ha, a-ha,-a#-a#
12. sza2 e#-!-lisz at-tal-la#
13. szar-ra-ha,-ku-ma
14. a-na rap-szi ki#-[...]
15. su-qu a-ba#-[...]
line preceded by 1(u)
81. e-ru-ub _e2#?-[gal_-...]
82. _iri#-mu_ ki-i a#-[a-bi ...]
83. tu-sza2-ma nak#-[ra-ti ...]
84. a#-na# [...]

Figure 5.2: Composite image of clay tablet and corresponding transliteration with two inscribed sides. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

collection. Fig. 5.2 provides an example of a full clay tablet image and a corresponding transliteration which are linked by their common CDLI number.

5.2.2 Data Preprocessing

The goal of data preprocessing is to obtain a dataset that consists of tuples of tablet image and corresponding transliteration. The main challenge for data preprocessing arises from the fact that clay tablets were often inscribed on both sides and even on the edges. Thus, images in CDLI are composites of different views (inscribed sides of a clay tablet) as shown in Fig. 5.2. Since clay tablets are 3D objects, we can imagine the unfolded surface of a rectangular cuboid in 2D. Similarly, the transliterations retrieved from SAAo are composites that contain transliterations of each inscribed side of the clay tablet. To obtain consistent tuples of tablet image and transliteration, data preprocessing performs three key steps for each clay tablet:

1. Extract the different tablet views from composite image. (Segmentation)

2. Extract the different transliteration sides from composite transliteration file. (Parsing)

3. Find an assignment between the tablet views and transliteration sides. (Assignment)

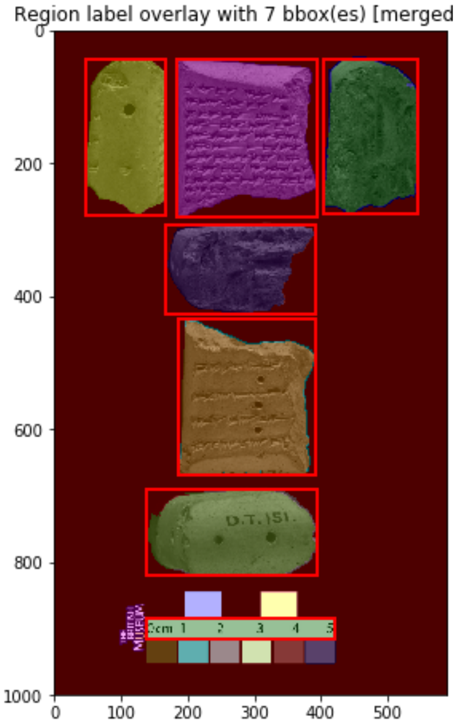


Figure 5.3: Composite image of clay tablet with segmented views. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

Segmentation of Composite Images

We segment the composite images to separate the different tablet views into individual tablet images. As the composite images are mostly placed on a high-contrast background (e.g. black or white), the segmentation of the individual views can be solved with a standard segmentation algorithm. We use the image segmentation algorithm by Felzenschwalb *et al.* [65] as implemented in the scikit-image library [217].

Before segmentation, we rescale all composite images to the same height of one thousand pixels. For the Felzenschwalb segmentation, we set `scale=1600` to capture only large segments, use `sigma=0.8` for smoothing, and set `min_size=1000`. After the segmentation, we merge segments whose intersection-over-union is larger than 0.2. Finally, all segments with an area smaller than 3000 pixels are removed and we obtain a list of candidate segments that most likely correspond to the actual views (sides) of the clay tablet. While the segmentation works well for most tablets, there are difficult edge cases due to partly broken tablets and overlapping views. To catch these outliers, we postprocess the results during the data cleaning step.

Parsing of Transliteration

The transliterations are provided as JSON data files whose format is customized for the ORACC transliteration data. It follows a hierarchical organization where on the highest level metadata about the clay tablet are stored, followed on the next lower level by the transliterations for all inscribed sides. Besides the relative position of cuneiform signs, transliterations contain additional information about the cuneiform signs, in particular their reading, their grammatical function (logogram, syllable, determinative), their state of preservation, and special damage markers. However, the level of detail of the information varies between transliterations due to individual preferences and digitization procedures.

For each clay tablet, we first split the corresponding JSON file into the different sides of transliteration. Then for each transliteration side, we parse the JSON data format and convert it into a data table where each sign in the transliteration corresponds to a row with the metadata[1] in the columns, in particular sign reading, sign code class, line number, line position, word number, state of preservation, tablet view, CDLI number. As a unique identifier of the sign code class, we use the encoding from Borger's MZL [29] (described in Sect. 4.2.2) which covers all signs used in Neo-Assyrian cuneiform.

Assignment of Tablet Views to the corresponding Transliterated Sides

To obtain a consistent dataset, the correct assignment between tablet views and transliteration sides needs to be determined for each clay tablet. The transliteration sides come with a label that indicates the side. However, identifying the tablet views in the composite image is often challenging, as the number of views shown and their layout in the composite image varies between clay tablets. The variation in the composite images is due to the fact that clay tablets vary in their shape and the number of inscribed sides. Nevertheless, composite images follow certain conventions and share certain properties which we leverage in our solution of the assignment problem.

To reduce the complexity of the assignment problem, we only focus on the front and back views (obverse and reverse) of the clay tablets. Since most of the clay tablets are shaped as rectangular cuboids, the obverse and reverse of a tablet (obv. and rev.) are the largest segments in the composite image which contain most of the cuneiform writing and are the easiest to identify. By analysing the collected composite images, we devise several heuristics to reliably identify the obverse and reverse segments in the composite image. Fig. 5.4 provides an overview of the heuristics used.

5.2.3 Data Cleaning

The data pre-processing step significantly adds to the number of errors already present in the collected dataset. The goal of data cleaning is to remove or fix as many errors as possible to ensure the integrity of the dataset. In the case of the cuneiform sign detection dataset, we have identified the following common error sources:

- Error in segmentation (e.g. two views segmented as one, view not segmented at all, damaged version of tablet)
- Error in assignment (e.g. wrong tablet view selected as obverse or reverse, swap of obverse and reverse)
- Deviation from the standard format of a tablet image or transliteration (e.g. two-column clay tablets, multiple tablets in single composite image, difference in transliteration format)

[1] For additional details regarding the data format, please consult the website of the dataset: https://github.com/CompVis/cuneiform-sign-detection-dataset

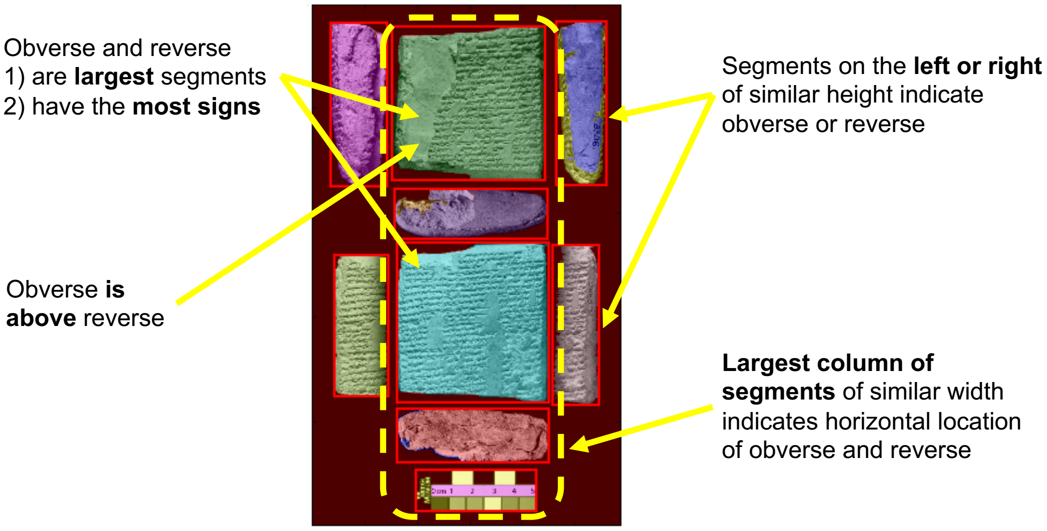

Obverse and reverse
1) are **largest** segments
2) have the **most signs**
Obverse **is above** reverse
Segments on the **left or right** of similar height indicate obverse or reverse
Largest column of segments of similar width indicates horizontal location of obverse and reverse

Figure 5.4: Heuristics for the assignment of sides of transliteration (obverse and reverse) to segmented tablet views. These are heuristics as there is no general rule how composite images are arranged. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

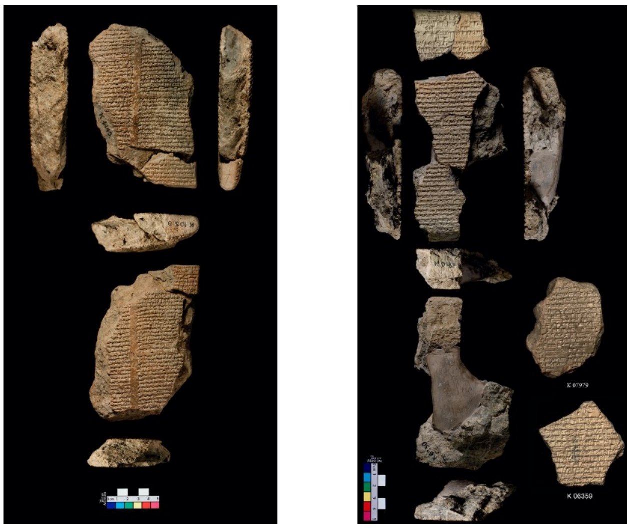
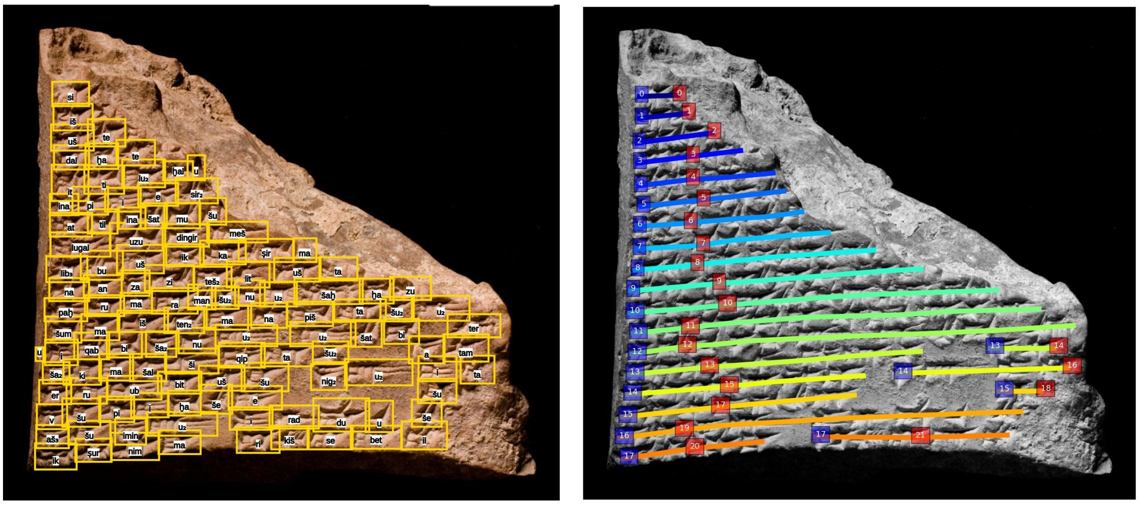


Figure 5.5: Two outlier composite images that cause problems during automatic assignment. Left: Cuneiform tablet in two-column format. Right: Fragments of multiple tablets in one composite image. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

Similar to the segmentation and assignment, we also automate the data cleaning step. For this purpose, we devise a set of handcrafted features that allow to verify the assignments and detect potentially erroneous cases automatically. The following four features are used during data cleaning: 1) line count: compare the number of transliterated and detected lines (requires a line detector); 2) sign count: compare the number of transliterated and detected signs (requires a sign detector); 3) alignment ratio: perform an image-transliteration alignment and check the ratio of aligned signs (requires alignment method); and 4) swap



(a) Sign annotations.

(b) Line annotations.

Figure 5.6: During data annotation, experts produces sign and line annotations for the full segmented tablet view based on existing transliterations. (A) Line annotations for a tablet image consist of piece-wise linear segments (identified by index in red box). A line (identified by index in blue box) can consists of multiple disconnected segments. (B) Sign annotations consist of bounding boxes and corresponding class labels. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

alignment ratio: test swapping obverse and reverse assignment for better match (requires alignment method). For each pair of tablet image and transliteration, we check for strong deviations in any of these features and remove the abnormal samples. In the case of a swapped obverse and reverse, we directly fix the assignment.

With data cleaning features, we face a chicken-and-egg problem, as we rely on line and sign detectors to evaluate the features and obtain a clean dataset, while in turn we need a clean dataset to obtain the detectors. We resolve this problem by bootstrapping: First, we create a preliminary dataset without data cleaning to obtain preliminary detectors. Second, we use the preliminary detectors to perform data cleaning and obtain a clean version of the cuneiform sign detection dataset.

5.3 Data Annotation

To support the quantitative evaluation and testing of a cuneiform sign detector, it is essential to obtain manual annotations (as ground truth) for a small subset of the cuneiform sign detection dataset. We rely on two types of manual annotations that are produced for a segmented view of a clay tablet as shown in Fig. 5.6: 1) A sign annotation which consists of a bounding box enclosing the sign and a label for its sign code class. 2) A line annotation which consists of multiple points marking the center of a cuneiform text line. Manual annotations are always created for the full segmented view of a clay tablet, i.e. annotating all visible signs and lines in a tablet image. Otherwise, if not all visible signs are properly annotated, the evaluation could be distorted.

Sign Annotations

Sign annotations are bounding box annotations that are produced by drawing rectangles around cuneiform signs and labeling them with their respective sign code classes. Localizing and classifying cuneiform signs requires expert knowledge as we described in the previous chapter (Sect. 4.2). The manual annotation of cuneiform script would be even more time-consuming, if not for the availability of transliterations. Transliterations speed up the manual annotation process as they provide a sign-by-sign representation of the tablets content which avoids most of the ambiguity involved in reading cuneiform script. Thus, they additionally reduce the number of errors introduced during the manual annotation. The produced sign annotations are primarily collected for the evaluation of the sign detector (testing), but are also utilized for the comparison of different learning approaches (training) as in Sect. 6.5.6.

Line Annotations

A line of cuneiform text is annotated as a sequence of connected linear segments (piece-wise linear segments) by marking points on the line: the start, end, and points in between to cover curved text. Most line annotations only consist of four linear segments (five points). Since lines are prominent features in cuneiform clay tablets and easily recognizable even for laypeople, their annotation does not require expert knowledge and is fast compared to sign annotations. The produced line annotations are primarily used for training and testing of a line detector (Sect. 6.4.2). Additionally, they help evaluating the quality of the sign annotations (Sect. 7.4.3).

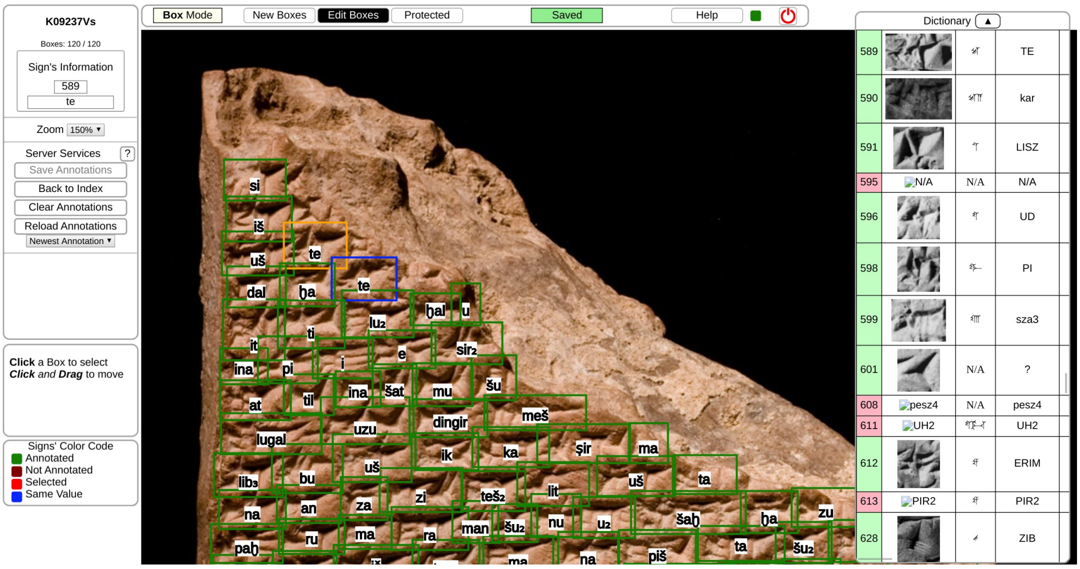
Expert Annotators

Five Assyriology researchers (master and PhD students) assisted in the manual annotation of clay tablet images. Besides producing sign and line annotations for the segmented tablet images, the expert annotators also performed the following important tasks: First, they verified the existing annotations of a smaller dataset provided by the preliminary work of Klinkisch [125]. Second, for each annotated tablet view, the annotators verified the assignment between segmented view and transliteration side, and corrected it if necessary.

Web Application

We employed a custom web application implemented in PHP and JavaScript to facilitate the manual annotation of tablet images. It is specifically designed for cuneiform sign annotation and detection. The integration of the sign detector is described in Sect. 6.6.8. Fig. 5.7 shows a screenshot of the web application during editing the sign annotations of a tablet image. We built on the web application originally developed as part of the work by Klinkisch [125] and significantly adapted it in close exchange with expert annotators to simplify the annotation process further. The web application offers the following core functionalities: 1) create collections of tablet images, 2) upload tablet images, 3) apply a cuneiform sign detector, 4) visualize cuneiform sign detections, and 5) annotate cuneiform signs and lines.

An important feature of the web application is its internal dictionary of cuneiform sign code classes that is available as a dropdown window during the annotation process. It



589		𒋼	TE
590		𒋼𒀀	kar
591		𒇺	LISZ
595	N/A	N/A	N/A
596		𒌓	UD
598		𒉿	PI
599		𒊮	sza3
601		N/A	?
608	pesz4	N/A	pesz4
611	UH2	𒌔	UH2
612		𒂟	ERIM
613	PIR2	𒂟	PIR2
628		𒍦	ZIB

Figure 5.7: The screenshot shows the web application, while a user edits sign annotations of a tablet image. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

provides a cuneiform sign list in the sense of [29] with readings, sign visualization and an example for each sign code class.

Human-in-the-loop

Additionally, to further reduce the burden of manual annotation, a sign detector (as proposed in Chapter 6) can be leveraged inside the web application to assist the annotator with its sign detections in a human-in-the-loop manner [185]. Instead of annotating tablet images from scratch, the annotation process is simplified to correcting the erroneous sign detections, i.e. changing labels and fixing bounding boxes. While the provided web application offers a basic implementation of this approach, it could be further refined using active learning strategies [164].

5.4 Results

Table 5.1 summarizes the results from data collection, preparation, and annotation. In the following we describe the results of the individual steps in more detail.

Data Collection

Data collection yields over four thousand data samples of clay tablets which come with a composite image and a corresponding transliteration. While there would be about six hundred more transliterations available, the composite images are the limiting factor here. The data collection step is significantly simplified due to the cooperation between CDLI and ORACC project and their adoption of CDLI numbers as an unique identifier of clay tablets. The SAAo project also proved to be a rich data source, which is apparent in the number of jointly available tablet images and transliterations.

Data Preparation

Data preparation is not able to assign all available tablet views with their respective transliteration sides, but still yields over six thousand pairs of tablet image and transliteration side. These tablet images contain over 500k visible signs and 80k lines. During data preprocessing, the segmentation of the composite images works well for the dataset except for a few edge cases. In contrast, the assignment step of data preprocessing is far more error-prone. Despite data cleaning, a substantial number of errors remain in the prepared dataset. We estimate that about five to ten percent of the assignments of tablet image to transliteration side may be erroneous (see Sect. 7.4.3 for details on this estimate). Since expert time is limited, only a little time was available for the correction of the assignments in the dataset, while most of it was spent on the manual annotation of cuneiform signs described in the next section. Thus, any learning approach based on this dataset has to take into account this level of noise in the training data.

Data Annotation

Data annotation yields annotations for over 455 tablet views with over 25k annotated signs and over 5k annotated lines. In total, about 200 person-hours were invested in sign annotation and about 10 person-hours in line annotations. Using the web application in a human-in-the-loop manner (based on a preliminary cuneiform sign detector), the fastest expert annotators were able to produce about 180 bounding box annotations per hour. However, the annotation speed varied significantly not only between annotators, but also between particular clay tablets due to changing levels of difficulty in reading the texts. Overall, the annotators averaged about 125 signs per hour. While less than five percent of the available signs were annotated, the amount of manual annotations is more than sufficient for thorough evaluation and testing of a machine learning approach.

Collection-specific Results

The data retrieved from the SAAo project is organized in collections according to text genre. Table 5.2 provides the collection-specific results of data collection, preparation, and annotation. The number of clay tablets varies across collections, with the smallest consisting of less than ten and the largest having more than five hundred clay tablets. Data preparation performs similarly across the collection, assigning on average over 90% of the available composite images to their transliterations. For most collections, we at least obtained some manual annotations to have the flexibility to evaluate sign detection across different genres.

5.5 Data Splits

From the prepared data, we build multiple datasets that are dedicated to different subproblems of reading cuneiform script. In particular, we build a dataset for line detection (see Sect. 6.4.2), a dataset for sign detection, and a dataset for linguistic refinement (see Sect. 7.4.1). The scope of each dataset depends on the requirements of the machine learning subproblem. In the case of linguistic refinement, a NLP-based approach, we use all available transliterations. While for line and sign detection, we only rely on subsets of the available

Table 5.1: Aggregated statistics of SAAo after performing data collection, preparation and annotation. The term visible signs refers to transliterated signs that are not marked as broken.

Step	Parameter	Count
data collection	tablets w/ transliteration	4770
	tablets w/ composite image	4181
	tablets w/ both	4172
data preparation	assigned tablets	3829
	assigned views	6505
	lines w/ visible signs	87520
	visible signs	574756
data annotation	views w/ annotations	455
	annotated lines	5482
	bounding box annotated signs	25578

Table 5.2: Statistics of SAAo collections after performing data collection, preparation and annotation.

	Data collection (w/ image)	Data preparation (w/ assigned TL)		Data annotation (w/ bbox annos)	
Collection	Tablets	Tablets	Views	Views	Signs
saa01	249	230	385	30	1720
saa02	7	5	9	9	1661
saa03	31	28	50	50	6888
saa04	280	260	442	59	2631
saa05	287	259	419	19	563
saa06	285	277	521	41	1874
saa07	196	177	293	17	473
saa08	545	501	864	33	1114
saa09	10	7	12	12	1190
saa10	365	355	637	19	1626
saa11	195	170	257	10	133
saa12	69	58	80	7	340
saa13	194	187	328	0	0
saa14	354	318	538	6	141
saa15	381	344	545	44	1750
saa16	231	213	350	0	0
saa17	198	175	298	34	1500
saa18	187	169	302	27	766
saa19	117	96	175	38	1208
Sum	4181	3829	6505	455	25578

Table 5.3: Collection sources of the two train sets and the test set.

Set	BB annotations	Collection sources
Train-TL	X	saa01, saa05, saa08, saa10, saa13, saa16
Train-BB	✓	TrainPK, saa05, saa09
Test	✓	TestPK, saa03*, saa06

* only a subset of all annotated tablet images in saa03 is used

Table 5.4: Composition of the two train sets and the test set.

Set	BB annotations	Signs	Views	Tablets	Weakly sup.	Semi-sup.
Train-TL	X	185399	2983	1745	✓	✓
Train-BB	✓	4663	67	47	X	✓
Test	✓	3446	57	34	-	-

data. When selecting subsets of the dataset, we follow the organization of SAAo and combine data from individual collections. Since each collection covers different text genres, the combination of multiple collections ensures a representative and diverse dataset.

5.5.1 Cuneiform Sign Detection Dataset

The cuneiform sign detection dataset is built for the training and evaluation of a cuneiform sign detector. Following standard practice, we also create training and test splits for the new dataset. Furthermore, a standard dataset for object detection usually consists of images where all objects of interest have been annotated with bounding boxes to support not only evaluation but also supervised learning of the object detector. Similarly, the cuneiform sign detection dataset also contains a subset of fully annotated tablet images that are split into a train (Train-BB) and test set (Test). Unlike standard datasets, the cuneiform sign detection dataset also includes a large subset comprising tablet images without bounding box annotations, since most of the available tablet images are not annotated. This subset forms another training set (Train-TL) that supports weakly supervised learning.

Table 5.3 lists the splits of the cuneiform sign detection dataset and the corresponding collection sources. Besides the collections listed in Table 5.2, we also rely on two collections (TrainPK and TestPK) that are based on sign annotations produced during the work by Klinikisch [125]. In Sect. 5.5.2 you find details on the collections of this legacy dataset. In the case of the Train-BB and Test splits, we only use the annotated subset of tablet images from each collection. The only exception is the saa03 collection, where we only use a subset of all annotated tablet images, since most of the sign annotations were created, after the conclusion of our experiments.

Table 5.4 provides an overview of the cuneiform sign detection dataset. For training, the number of available segmented views (images) depends on the form of training (i.e. weakly supervised, or semi-supervised): For purely weakly supervised training, we use 2983 segmented views with their associated transliterations (Train-TL). For semi-supervised training, we additionally use up to 67 views (Train-BB) where all visible signs are annotated with bounding boxes (BB). For validation, we use 31 annotated views from Train-BB as a hold-out validation set (with 1753 sign annotations) to optimize hyperparameters. For testing, we use another 57 annotated views (Test).

Table 5.5: Statistics on data collection, preparation and annotation for legacy collections.

	Data collection (w/ image)	Data preparation (w/ assigned TL)		Data annotation (w/ bbox annos)	
Collection	Tablets	Tablets	Views	Views	Signs
Train-PK [train]	27	27	38	36	2910
Test-PK [testEXT]	11	9	16	16	1572
TrainTL-PK [ransac]	17	17	17	1	116
Sum	55	53	71	53	4598

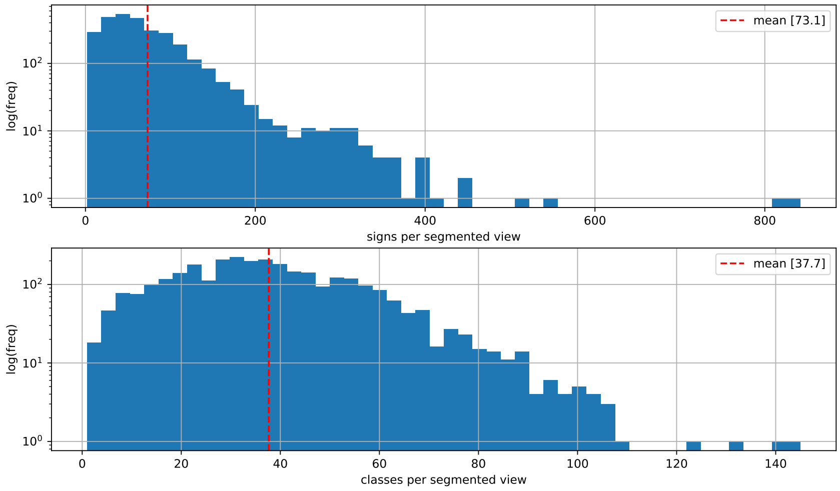


Figure 5.8: Frequency distribution of segmented views with respect to the number of signs (upper plot) or sign code classes (lower plot).

5.5.2 Legacy Dataset

As part of the work by Klinkisch [125], an initial dataset for cuneiform sign detection was built, which we adapted and extended with additional annotations. In Table 5.5 we show the structure and properties of the legacy dataset. The dataset is organized in three disjoint collections: Train-PK, Test-PK, TrainTL-PK. Two collections are used in the larger cuneiform sign detection dataset described in Sect. 5.5.1.

5.6 Data Understanding

In the following, we analyse the statistics for tablet images and sign code classes of the cuneiform sign detection dataset. Since cuneiform tablets were used for various purposes comparable to a piece of paper (cf. Chapter 4), it is interesting to study how the quantity of written text varies between tablets. Similarly, it is interesting to study how the usage of many cuneiform signs varies across the dataset.

5.6.1 Tablet Image Statistics

In Fig. 5.8 we study the size of the tablet images (segmented views) in terms of the number of visible signs and the number of distinct sign code classes. We find that the dataset is skewed towards smaller tablet images that on average contain about 73 visible signs and have about 38 distinct sign code classes. Nevertheless, there is also a substantial amount of medium-sized to large tablet images (100 to 400 signs). Interestingly, the majority of tablet images features less than 100 different sign code classes, which supports the finding that back in the days the knowledge of less than 200 sign code classes was sufficient for everyday usage of cuneiform script.

5.6.2 Sign Code Classes

We provide an overview of the different cuneiform sign code classes in the cuneiform sign detection dataset. In this analysis, we only consider the 186 different cuneiform sign code classes that are present in the train set and have at least one annotated example in the test set. In Sect. 7.4.1 we provide an analysis of all 342 cuneiform sign code classes that are found in the transliterations of the SAAo dataset.

Sign Code Classes used for Detection

Fig. 5.9 illustrates the fine-grained nature of the sign detection task. The diversity as well as the similarity between sign code classes is apparent by just comparing the characters printed in a Unicode cuneiform font [215]. We show the cuneiform signs sorted according to their sign code as introduced in Borger's sign listing [29], which groups signs according to common substructures and similarity. Often cuneiform signs are composites of several smaller signs, where the only difference between two classes is a single wedge. Besides the high inter-class similarity, the variation between signs of the same class in real tablet images is high, which we illustrate in Fig. 6.21.

Frequency of Sign Code Classes

Fig. 5.10 visualizes the frequency of the 186 sign code classes in Train TL set (unlabeled tablet images with transliteration). We only count an occurrence of a sign in Train TL, if it is not labeled as broken in the transliterations. The resulting frequency distribution roughly follows a Zipfian distribution (discrete Pareto distribution). When visualized in log-scale, the exponential distribution becomes a linear trend. Further, we can observe that the least frequent 25 sign code classes have less than 100 occurrences, whereas the most frequent 25 have more than 2000.

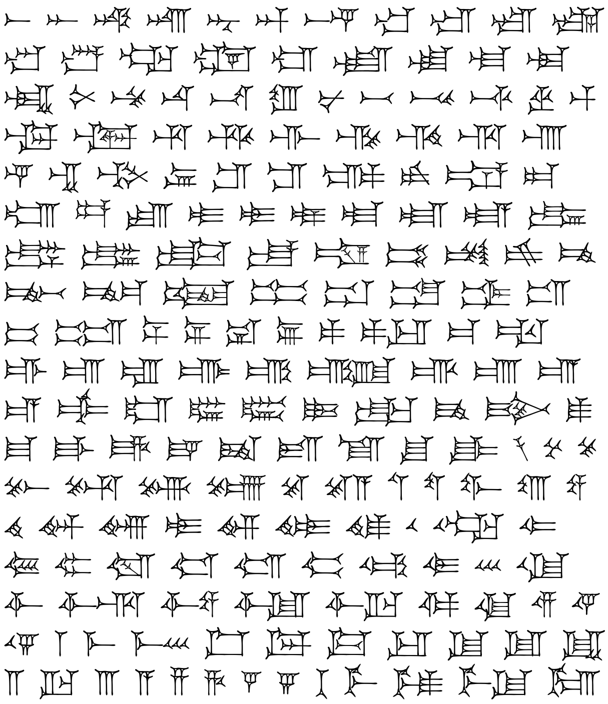


Figure 5.9: Unicode characters of sign code classes considered for detector training. We show all cuneiform signs which are supported by the Unicode font (181 out of 186).

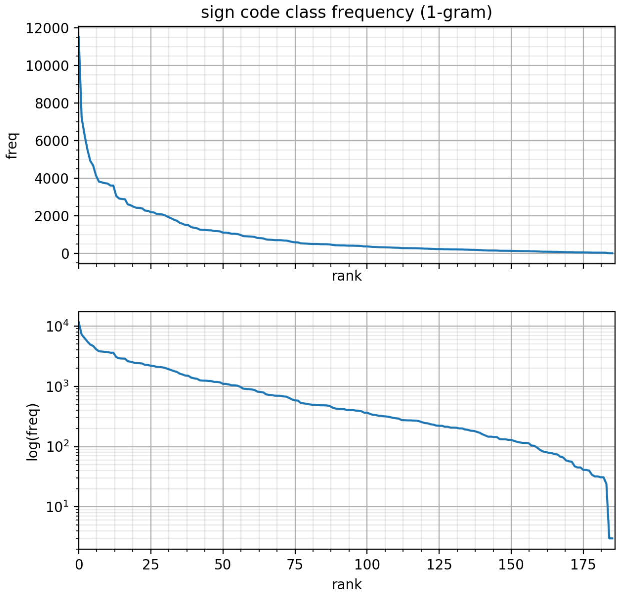


Figure 5.10: Plot of the (1-gram) frequency of the 186 sign code classes in the Train TL set with classes ranked according to their frequency. In the upper plot the exponential drop in class frequency is visible, which indicates a discrete Pareto distribution. This is confirmed by the lower plot with logarithmic scale that shows a linear trend for most of the classes.

CHAPTER 6

Weakly Supervised Cuneiform Sign Detection

The cuneiform script provides a glimpse into our ancient history, however, reading age-old clay tablets is time-consuming and requires years of training (cf. Chapter 4). To simplify this process, we present a deep-learning based sign detector that locates and classifies cuneiform signs in images of clay tablets. We propose a weakly supervised learning approach that leverages existing transliterations (sign-by-sign representation of tablet content in Latin script) to supervise the sign detector training - without requiring manual sign annotations. Since a transliteration does not provide the absolute position of cuneiform signs in a tablet image, our method learns how to align hundreds of tablet images and corresponding transliterations in parallel in order to generate localized annotations for detector training. This is significant because the proposed method shows how to perform weakly supervised learning in a difficult fine-grained detection setting.

To the best of our knowledge, this is the first approach to demonstrate robust sign detection for over a hundred sign code classes evaluated on a large and diverse collection of tablet images as described in Chapter 5. Since our method is applicable to cuneiform script of different periods, it enables large-scale digital analysis of cuneiform script in the field of Assyriology, thus helping to more efficiently open up the treasures of human civilization. To directly support Assyriologists in their analysis of cuneiform texts, we also present a web application of the cuneiform sign detector, thus making it easily applicable.

6.1 Introduction

Even for experts, the process of reading and analyzing clay tablets is difficult and time-consuming. To support Assyriologists in their analysis, we want to facilitate the decipherment of cuneiform script. In particular, our goal is to obtain a *cuneiform sign detector* that outputs for a tablet image where a sign is located (bounding box enclosing the sign) and what sign code class it belongs to (according to Borger's sign list [29]) as shown in Fig. 6.1. Our aim is not to replace Assyriologists by automatically generating a transliteration or translation, but rather to support them with sign detection as a fundamental part of reading cuneiform script. For this purpose, we also present a web application that

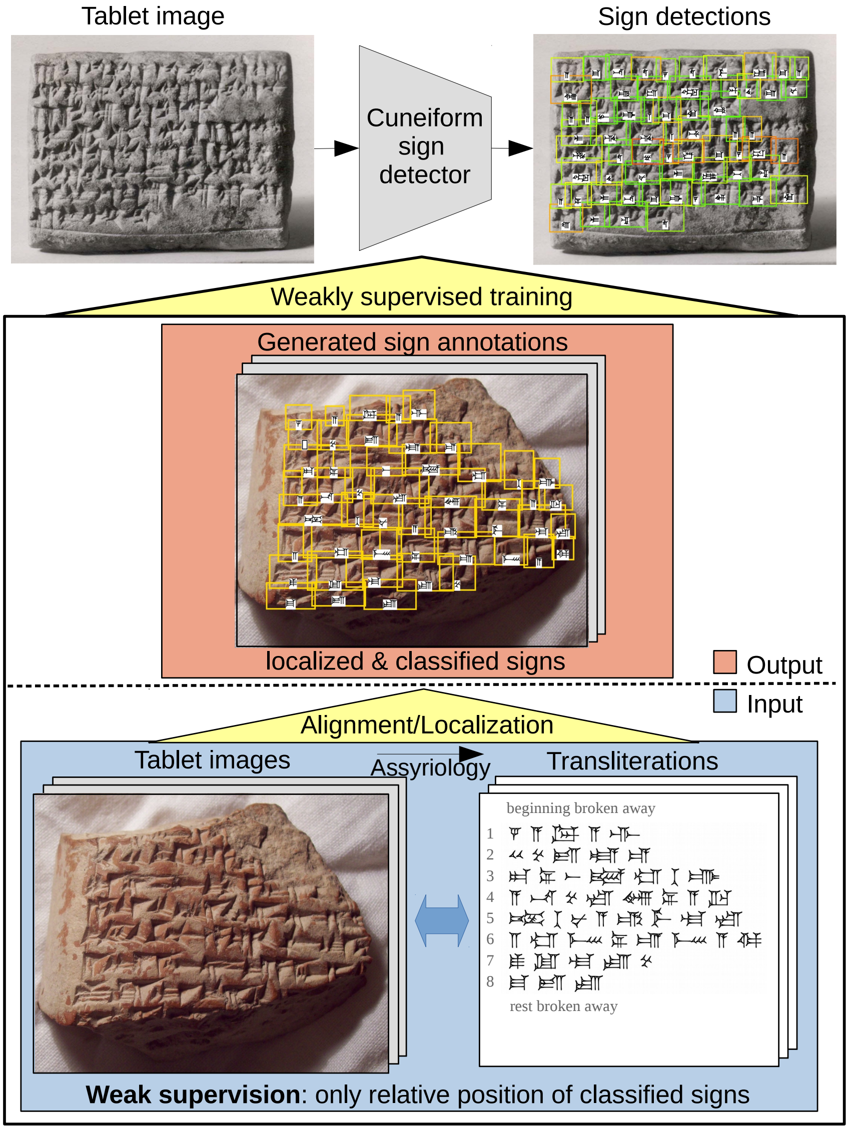


Figure 6.1: Overview of our approach. To support Assyriologists we train a cuneiform sign detector to localize and classify cuneiform signs in tablet images. The sign annotations necessary for training the sign detector are automatically generated by localizing signs of existing transliterations in their tablet images. This alignment turns weak supervision of the transliteration into full supervision in terms of bounding boxes. Image material at the top is shared by The Metropolitan Museum of Art under a CC0 license. Image material below by the authors.

makes the cuneiform sign detector readily available to Assyriologists, which we illustrate in Sect. 6.6.8.

Ideally, we would follow a standard machine-learning approach and train a sign detector from thousands of annotated examples of cuneiform signs (supervised learning) [108, 218, 219, 240]. However, a sufficient amount of training data in the form of bounding box annotations is not available and its collection is time-consuming and requires expert knowledge. Moreover, due to the changes in cuneiform script over time, we would need to collect multiple training sets. Instead of manually creating new annotations, it is more natural to use existing transliterations that are an essential part of Assyriologist's work and available for thousands of tablets via the ORACC websites [161].

While the transliteration[1] provides a sign-by-sign representation of the tablet content, it does not explicitly localize each sign in the 2D tablet image. If we want to learn how to detect cuneiform signs from tablet images and corresponding transliterations (weakly supervised learning), we need to localize the transliterated signs for each 2D image, i.e. solving the inverse problem with respect to sign detection as shown in Fig. 6.1. We can understand the transliteration as a Rosetta stone [2, 166] for learning how to detect cuneiform script: By aligning image and transliteration, we identify correspondences in different representations of the same underlying text. These correspondences provide examples for training a better sign detector that in turn improves the alignment of image and transliteration. By repeating this process in parallel over hundreds of tablet images, our approach gradually learns how to detect signs, starting with frequent and easy-to-locate signs, and step-by-step filling in the remainder. Instead of teaching the algorithm by annotating thousands of examples manually, it learns from existing 2D tablet images and corresponding transliterations.

From a machine-learning perspective, the transliteration provides *weak supervision*, in contrast to the fully supervised setting with available bounding box annotations. Weakly supervised learning is an important research area as it is one of the most common learning scenarios found in the wild as described in Sect. 2.4.2. Learning to detect cuneiform script challenges us to find new ways to bridge the gap between weak and full supervision, enabling state-of-the-art cuneiform sign detection for over a hundred sign code classes without requiring manual annotations. While our proposed solution deals with cuneiform sign detection, similar problems (fine-grained detection, few annotations) occur in other ancient scripts that might profit from our weakly supervised approach.

In the following we summarize the related work, describe our approach in detail, evaluate our approach and its components thoroughly in a number of experiments, and finish with a discussion of our approach and its implications for computer vision and Assyriology.

6.2 Related Work

Digitization and Analysis of Cuneiform Script Cuneiform tablets have been digitized in various forms that strongly differ in their availability and production cost. Cuneiform tablets are best studied in their original form as 3D objects: signs are often found not only on the front and back but also along the edges, and their visibility depends on lighting and viewpoint. Thus working with 3D scans of tablets is desirable [8, 48, 52, 71, 146]. From 3D scans, 2D representations are derived that are used for recognition [118, 181]. Similarly, autographs can be used as input for a detector [27, 28, 62, 231]. However, the creation of 3D scans or autographs is expensive and time-consuming. In contrast, we rely on 2D images of tablets (each a composite image of different tablet sides) which are readily available online [49].

Text and Handwriting Recognition There are three categories of computer vision approaches for text or handwriting recognition: Character-based [102, 108, 152, 207, 218,

[1] In the context of this thesis, we refer to the transliteration as the sign-by-sign representation in terms of sign code classes, rather than sign readings as explained in Sect. 4.2.2.

219], word-based [88, 138, 139, 240], or line-based approaches [35, 86, 221]. Unlike a character-based approach that localizes and classifies a sign in one step, the word-based and line-based approaches only localize words or lines of a text, which are then transcribed into a sequence of signs without explicit sign-level localization. In the context of cuneiform script, line-based approaches suffer from the noise introduced by the additional line detection step, since lines are often damaged and tricky to follow due to gaps and offsets. A word-level approach is ill-suited due to polyvalence of cuneiform signs that allows to assemble a word from various, often ambiguous combinations of signs. We adopt a character-based approach that results in a sign detector that directly outputs bounding boxes for individual cuneiform signs. Sign-level bounding boxes help to explain the decision making of the detector and thus provide a crucial asset for Assyriologists.

Learning with Limited Supervision Character-based methods [102, 207] have been proposed that use word-level annotations in a weakly-supervised learning framework. While they infer the positions of individual characters from the bounding box around a word, our approach localizes cuneiform signs by aligning the image of a clay tablet with its transliteration which usually involves multiple lines of cuneiform script. Additionally, our method works without an initial set of training annotations. Beyond weak supervision, there are unsupervised approaches. Using synthetic data is a common approach for (pre-)training a detector for typed texts unlike handwriting [88, 138]. Recently, a method for the generation of cuneiform script has been proposed with promising results [183], but the outputs still lack the diversity of real-world data needed to train a robust sign detector. [23] uses a probabilistic model to infer the most likely transcription of historic printed texts. [87] proposes a generative adversarial network to train a character classifier from unpaired data of line segmentations and unaligned text transcriptions, only requiring them to be remotely related. While these approaches show promising results for printed English text, they are not directly applicable to non-Latin handwritten scripts with far more characters that are written without regular spacing between words and that possess a high degree of self-similarity.

6.3 Approach

Training a cuneiform sign detector requires thousands of sign annotations. Instead of requiring manual annotations, our weakly supervised approach generates sign annotations by automatically aligning hundreds of tablet images with their transliterations in parallel. We propose an *iterative learning* procedure that alternates between generating sign annotations and training the sign detector. We model the sign detector as a convolutional neural network (CNN) whose representation of cuneiform signs is updated in each iteration of iterative training.

To start iterative training, an initial set of sign annotations is required to train a first sign detector. Since the transliteration only provides the relative positions of cuneiform signs, we propose a sign placement method that relies on line detections to turn the relative positions of the transliterated signs into actual bounding boxes in the tablet image (*placed detections*). To keep improving the sign detector, it is necessary to increase the quality and quantity of generated sign annotations from iteration to iteration. When applying the trained sign detector on tablet images (of the training set), this produces *raw detections*

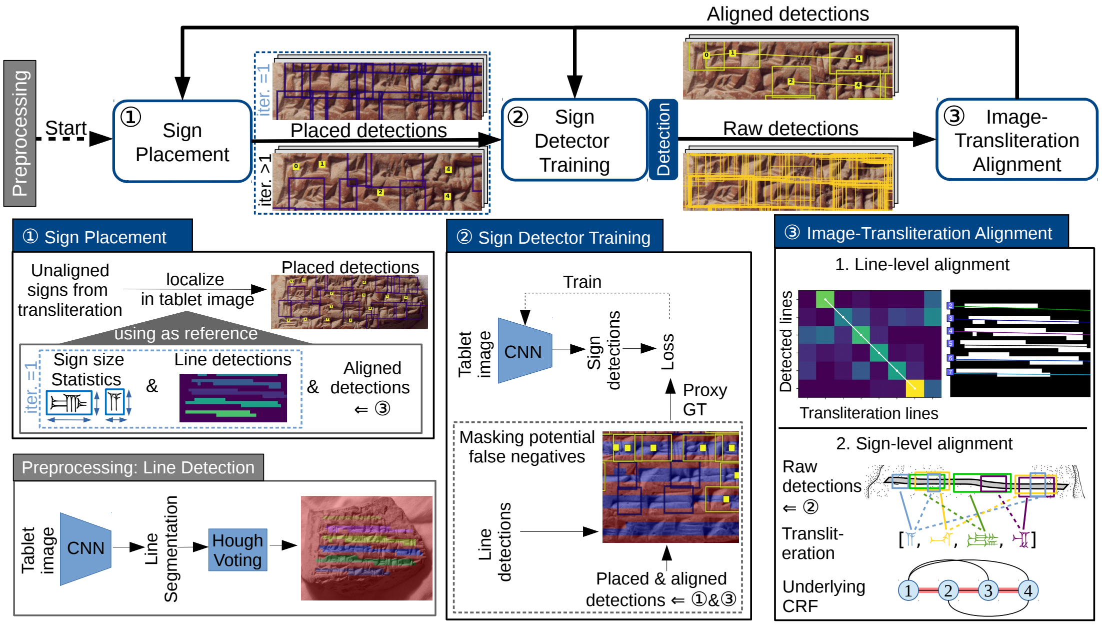


Figure 6.2: Weakly supervised training of sign detector comprises an iterative training loop with three steps. In each iteration the sign placement method localizes transliterated signs in the tablet image using detected lines and aligned detections as reference points. In the second step, the aligned & placed detections serve as sign annotations for training a sign detector. In the third step, the image-transliteration alignment filters the raw sign detections produced by the sign detector after the second step. Only the raw detections that are consistent with the transliteration and the line geometry are selected. The resulting aligned detections are very reliable sign annotations, which boost sign detector training in the next iteration. As preprocessing, we detect lines in all tablet images to simplify localization and alignment. Image material by the authors.

with few true positive (TP) and many false positive (FP) sign detections. By aligning transliterated signs with raw detections in a tablet image, our method identifies a reliable subset of raw detections (*aligned detections*) that are most likely TPs and can serve as sign annotations for the next training iteration. When aligned detections become available after the first iteration, the task of sign placement shifts to localizing the unaligned signs left in the gaps between aligned detections. As summarized in Fig. 6.2, our approach performs three key steps in each iteration of iterative learning:

1. Localize transliterated signs in the tablet image using detected lines and aligned detections as reference points (*sign placement*).

2. Train a sign detector using the generated aligned & placed detections as sign annotations (*sign detector training*).

3. After applying the sign detector, refine the raw detections by optimizing the alignment between transliteration and tablet image (*image-transliteration alignment*).

To better understand why the placement and alignment methods are essential for iterative learning, we can think about their interaction in terms of exploration vs. exploitation: The placement method provides an initial sign localization and later fills the gaps between aligned detections which include valuable training samples of sign code classes yet to be learned by the detector (exploration). In contrast, the alignment method converts weak supervision from transliterations into high-quality aligned detections (exploitation). The ability to explore new sign code classes is of particular importance due to the problem of sign code class imbalance: Many sign code classes are rare in cuneiform script because their occurrence follows a Zipfian distribution as shown in Fig. 5.6.2. During iterative learning, the sign detector is biased to perform best on sign code classes with the most training data, while ignoring rare sign code classes. To counter this effect, sign placement injects sign annotations that are difficult-to-learn or belong to rare sign code classes into the iterative learning process.

Line Detection Before iterative learning, line detection is performed as a preprocessing step on all tablet images. Cuneiform text is commonly inscribed along lines. Since these lines are easy to detect, we can simplify the alignment between tablet image and transliteration. Given the lines in a tablet image, the full alignment problem is decomposed into a line-level alignment (that matches detected with transliterated lines) and for each line a sign-level alignment (that matches detected with transliterated signs). Knowing the position of each line can effectively reduce the sign localization in a 2D tablet image to an 1D search along a line, significantly constraining the number of possible solutions.

We propose a line detection method for cuneiform script that benefits the three steps of our learning method described in the following paragraphs. As shown in Fig. 6.2, it is implemented as a CNN for line segmentation (pixel-wise labeling of line/no line) whose output is postprocessed using a straight line Hough transform [61] voting for line detections. We provide the implementation details of the line segmentation network, its training, and the postprocessing in Sect. 6.4.2.

Sign Placement Transliterations only provide a relative placement of some signs to another, but not the exact locations of individual cuneiform signs in the tablet images (bounding boxes), essential for training a sign detector. In the first iteration of iterative training, the sign placement method combines the relative sign positions from the transliteration with line detections in the tablet image in order to generate an initial set of sign annotations as shown in Fig. 6.2. The detected lines serve as reference lines along which to place sign bounding boxes. For the line-level alignment, we match the detected lines with transliterated lines from top to bottom. The heights of sign bounding boxes are derived from the distance between detected lines. The widths are computed by multiplying the sign heights with a class-specific sign width from a precomputed statistic. Even though only a small percentage of placed sign detections are correct, applying this method to hundreds of tablet images still yields a substantial number of valid sign annotations that our learning method can exploit iteratively.

In later iterations of iterative learning, the sign placement additionally leverages the aligned sign detections in order to localize the remaining unaligned signs from the transliteration. The aligned detections serve as additional reference points throughout the tablet image that complement the reference provided by the detected lines. In particular, the localization of signs close to aligned detections is improved. Therefore, the more aligned detections become available during iterative training, the better the remaining unaligned signs can be localized. Also the line-level alignment is now adopted from the previous image-transliteration alignment step. More details on the implementation of sign placement are provided in Sect. 6.4.3.

Besides providing an initialization for our learning method, in later iterations the sign placement step drives the learning of sign code classes not yet learned by the sign detector. Placed detections correspond to transliterated signs that have either not been detected or not been aligned successfully, and thus have the potential to be valuable training data for the next round of detector training. Through iterative learning with placed detections, the sign detector gradually learns to distinguish new sign code classes.

Sign Detector Architecture The CNN of the sign detector is based on the Single Shot Multi-Box Detector (SSD) architecture [137] for generic object detection, which we adapt for the detection of cuneiform signs. It defines a dense grid of anchor boxes over the input image and outputs for each box a class prediction as well as a regression of box coordinates for a better fit. It uses a class-generic bounding box regression, which is beneficial for classes with few examples [212]. We also extend the sign detector with a feature pyramid network [134] that enables SSD to cover additional image scales to detect signs of different sizes. This increases the sign detector's robustness against scale changes in the cuneiform script. More implementation details are provided in Sect. 6.4.4.

As backbone architecture of the sign detector, we use a MobileNet-v02 [187] with the width multiplier set to 0.625, since it shows good performance and has few trainable parameters which helps with regularization. We further reduce its capacity by removing the last inverted block and increasing the stride of the first layer to two for faster downsampling. A comparison of other backbone architectures is provided in Sect. 6.6.6.

Sign Detector Training When we train a cuneiform sign detector, we improve its internal representation of cuneiform signs. Different cuneiform signs share the same representation that captures the commonalities between signs. Since cuneiform signs are all constructed from very similar parts (wedges), success in the distinction of some sign code classes already lays the foundation for the distinction of yet unlearned sign code classes. Erroneous and ambiguous training samples are difficult to integrate into the learned representation and tend to be ignored. Thus the sign detector training provides in itself an critical regularization that filters out the noise in the generated sign annotations.

In each iteration of iterative training, the sign detector is trained on the aligned & placed detections as shown in Fig. 6.2. The training of a sign detector heavily relies on difficult negative examples of sign detections (hard negatives), e.g. bounding boxes centered between two lines. During sign detector training, hard negatives are automatically sampled in the close proximity of the generated sign annotations (positives). However, the generated aligned & placed detections only approximate ground truth annotations, and misplaced detections as well as undetected signs are part of the iterative learning process. Since aligned & placed detections only account for a subset of all visible signs in the tablet images, the automatic sampling of hard negatives can erroneously produce positive sign detections labeled as negatives (false negatives) which inhibits detector training. Our proposed method allows detector training despite incomplete and sometimes erroneous sign annotations. While we do not know the ground truth sign locations of all visible signs, we consider the line segmentation as a reliable indicator for their presence. By means of line segmentation, we mask and discard all potential FP training samples in regions of the tablet image that are not covered by the aligned & placed detections (blue regions outside of bounding boxes in Fig. 6.2). The resulting proxy ground truth (GT) incorporates all generated sign annotations and many valuable hard negatives, while reducing the negative impact of false negatives.

Our approach allows the integration of expert knowledge in the form of manual annotations, e.g. correcting some of the generated sign detections. While our learning approach works weakly supervised (without full supervision), we find that complementing the weak supervision (i.e. transliterations) with a small number of sign annotations is beneficial in such a fine-grained detection setting. For this purpose, we extend the sign detector training with an optional fine-tuning step that trains the sign detector with a reduced learning rate on training samples annotated by experts. This form of training is referred to as *semi-supervised* learning because sign annotations are only available for a small subset of the training data. The rest of the training data remains weakly supervised. During our experiments, we evaluate the sign detector training in both cases, purely weakly supervised and semi-supervised. In Sect. 6.4.4 and Sect. 6.5.2, we provide details on false-positive masking, fine-tuning, and hyperparameters for training.

Image-Transliteration Alignment There are many causes like damaged signs or wedge-shaped cracks that can fool a cuneiform sign detector to produce FP sign detections. We close the iterative training loop of our approach by including an image-transliteration alignment step that serves as a filter for the raw detections of the sign detector. By selecting only sign detections that are successfully aligned with transliterated signs, we obtain a set of reliable detections for the next iteration of iterative training.

By extracting reliable aligned detections from the raw output of the sign detector, the image-transliteration alignment addresses two problems caused by an inherent discrepancy between tablet image and its transliteration: First, the transliteration does not record every visual detail found in the tablet image like writing style, gaps in the lines, or damage to the clay tablet. Second, the transliteration often goes beyond what is actually visible in a tablet image. An Assyriologist may infer additional cuneiform signs due to context knowledge like multiple text sources, language understanding etc. While broken and damaged parts of the script are often documented, the level of detail varies between transliterations.

The optimization of the image-transliteration alignment yields a set of aligned detections that is consistent with the top-down information from transliteration and line geometry as well as the bottom-up information from line and sign detections. Due to line detection, we can decompose the problem of aligning transliterated signs of the transliteration with raw detections in the tablet image into two individual alignment problems as shown in Fig. 6.2: 1) Aligning transliteration lines with detected lines in the tablet image (line-level alignment). 2) Solving the line-wise alignment problem of localizing signs from the transliteration in the detected line (sign-level alignment). Compared to a full-image alignment, this line-wise decomposition results in local subproblems with constrained search spaces that increase the chance of finding a correct solution and speed up computation. Additionally, difficult lines (sub-problems) can be skipped without impacting easier lines that can already be aligned in early iterations of our learning method. In the following we formulate both line-level and sign-level alignment as individual optimization problems:

The line-level alignment is modeled as a path search problem through the matrix of all possible alignments where the longest path represents the best line alignment (see in Fig. 6.2 upper half of alignment box). Details on the implementation of the line-level can be found in Sect. 6.4.5.

The sign-level alignment is formulated as a pictorial structures model [66] where the best sign alignment is found by minimizing the energy of a Conditional Random Field (CRF) model . A sign alignment assigns each sign in the transliteration exactly one out of many available raw detections of the sign detector (see in Fig. 6.2 lower half of alignment box). The energy of a sign alignment is determined by the energy functions (potentials) of the CRF that are designed according to our prior knowledge of what constitutes a reliable sign alignment (e.g. correct order of signs, relative straight line).

To construct a CRF for a cuneiform line (as shown in Fig. 6.2 lower half of alignment box), we create a node for each sign in the transliteration vector and add edges to form a fully-connected graph. Nodes and edges of the CRF are associated with unary and pairwise potentials that rate the local appearance of signs as well as their joint spatial configuration. In the case of unary potentials, we rely on the confidence of the raw detection and the vertical offset of the detection bounding box from the detected line. In the case of pairwise potentials, we consider the horizontal distance between signs, the overlap of their bounding boxes, and the angle between the line drawn through the center points of two signs and the detected line.

By optimizing for an assignment associated with low energy, the raw detections are selected that best match our prior knowledge. In the case that no matching raw detections can be assigned to a transliterated sign, the outlier label is assigned that incurs a fixed energy penalty. Thus the sign-level alignment still produces useful results, even if a subset of a line cannot be assigned. Details on the implementation of the sign-level alignment, including the formal definition of the energy functions, can be found in Sect. 6.4.6.

6.4 Methods

In the following we provide additional details on all components of the iterative learning procedure that allows to train a cuneiform sign detector as summarized in Algorithm 1.

6.4.1 Iterative Training with Weak Supervision

Before the start of the iterative training, the line segmentation network is trained on a small set of labeled tablet images and applied to all tablet images of the train set in order to obtain line detections (Sect. 6.4.2). The iterative training generates in each iteration a new set of aligned and placed detections that is used to supervise the training of the cuneiform sign detector (Sect. 6.4.4). The iterative training, summarized in algorithm 1, proceeds as follows: 1) Sign placement produces placed detections using transliteration and line detections in the first iteration and later also aligned detections (Sect. 6.4.3). 2) We train a new sign detector on the union of aligned and placed detections, and fine-tune it with manual sign annotations if available (Sect. 6.4.4). 3) We apply the trained sign detector to the Train TL set (unlabeled tablet images) and use the line and sign detections to find an image-transliteration alignment on line-level (Sect. 6.4.5). Then for each line we find a sign-level alignment between sign detections and transliteration vector in order to produce a set of aligned detections (Sect. 6.4.6). The iterative training of steps 1-3 is repeated until the performance of the sign detector stops improving. The fine-tuning step is the only difference between the weakly and semi-supervised case of our learning approach.

Algorithm 1: Weakly Supervised Iterative Training of Cuneiform Sign Detector

```
Input: $L$: Line detections; $I$: Tablet images; $T$: Transliterations; [$B_{manual}$:
        Manual sign annotations]
Output: $D$: Cuneiform sign detector
$B_{placed} \leftarrow \emptyset$; $B_{raw} \leftarrow \emptyset$; $B_{aligned} \leftarrow \emptyset$   // Initialize placed, raw and aligned sign detections
$A_{line} \leftarrow \emptyset$; $A_{sign} \leftarrow \emptyset$  // Initialize line-level and sign-level alignment information

Iterative Learning:
repeat
    1. Sign Placement // Sect. 6.4.3
    if first iteration then
        $B_{placed} \leftarrow$ LocalizeAllSigns($T$, $L$)
    else
        $B_{placed} \leftarrow$ LocalizeUnalignedSigns($T$, $L$, $B_{aligned}$, $A_{line}$, $A_{sign}$)

    2. Sign Detector Training // Sect. 6.4.4
    $D \leftarrow$ TrainNewSignDetector($I$, $B_{aligned} \cup B_{placed}$, $L$)
    if $B_{manual}$ is available then
        $D \leftarrow$ FineTuneSignDetector($D$, $I$, $B_{manual}$) // semi-supervised case
    $B_{raw} \leftarrow$ RunSignDetection($D$, $I$)

    3. Image-Transliteration Alignment // Sect. 6.4.5 (line-level); Sect. 6.4.6 (sign-level)
    $A_{line} \leftarrow$ OptimizeLineLevelAlignment ($T$, $L$, $B_{raw}$)
    [$B_{aligned}$, $A_{sign}$] $\leftarrow$ OptimizeSignLevelAlignment($T$, $L$, $A_{line}$, $B_{raw}$)

until $D$ is converged
```

6.4.2 Line Detection

Our line detection consists of two steps: 1) Line segmentation with a convolutional neural network (CNN) and 2) Hough transform-based postprocessing to obtain line detections.

Line Segmentation

For line segmentation, we train a CNN to classify image patches as line center or background. As CNN architecture, we use a modified AlexNet [121] with BatchNorm [107] after convolutional layer 3-5 and only one linear layer with 512 neurons before the classification head. During training, image patches close to the boundary of line annotations are sampled more frequently, thus focusing the classifier training on hard-positive and hard-negative cases. At test time, we obtain a segmentation map of the full image by turning the classifier into a fully-convolutional network following the technique described in [140].

The most informative training samples (image patches) for segmentation training are located close to the annotated lines, since distinguishing these hard samples requires to

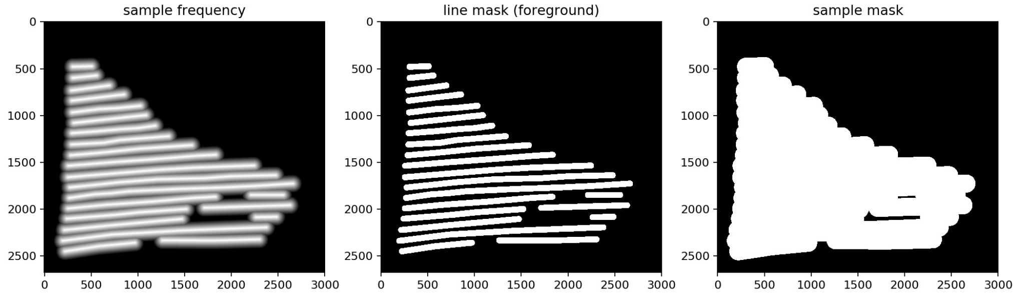


Figure 6.3: Sampling of training data for line segmentation. Most image patch centers are sampled according to the sample frequency (left) from the positive area of the sampling mask (right). Only image patches whose center is inside the line mask are considered positive (middle).

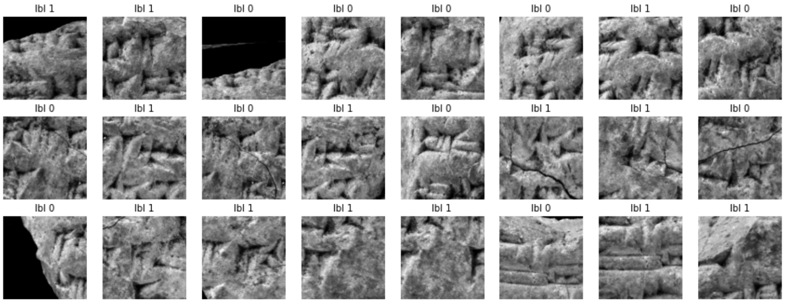


Figure 6.4: Training data for the line segmentation network consists of images patches sampled close to line annotations. Image patches centered on a line have label 1, otherwise 0. Labels are displayed above each image patch.

learn the fine difference between line and no line. Thus, instead of sampling image patches from the whole tablet image, we focus the sampling on the area close to the line annotations (sample mask) as visualized in Fig. 6.3. A sample is positive if its center is located on the line mask. Line mask and sample mask are obtained with the help of the distance transform by selecting the area around the annotated linear segments within a distance of 25 or 75 pixels respectively. To further encourage the sampling of the most difficult training samples, we compute a sample frequency map as shown in Fig. 6.3 which increases the likelihood of sampling with decreasing distance to a line. This sample frequency is again computed with the distance transform based on the annotated linear segments. Ten percent of training samples originate from the area around the sampling mask in order to learn about the general background class as well. In Fig. 6.4 we visualize training samples used for line segmentation training. Only a few samples are easy to distinguish, while many samples are more difficult as they are from the border between line and no line.

As training data for the line segmentation network (line center classifier), we use line annotations of 38 segmented views of clay tablets, where all visible lines have been manually annotated as described in Sect. 5.3. In total we rely on 410 annotated lines of cuneiform script for training.

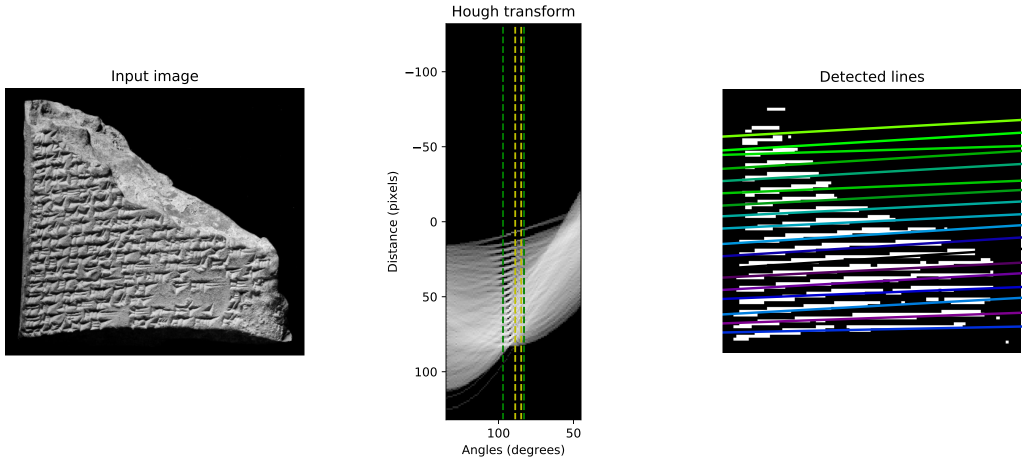


Figure 6.5: Postprocessing of line segmentation to obtain detected lines. By applying the line segmentation network to the input image (left) the line segmentation is obtained (right: white area). Using a straight line Hough transform (middle), first an estimate of the average line angle is obtained (green interval) and refined (yellow interval), then the detected lines are obtained (right: colored lines) by searching for the peaks in the yellow interval.

Line Segmentation Postprocessing

The goal of postprocessing is to produce a list of detected lines from the line segmentation of a full tablet image. For robust line detection, we postprocess the line segmentation of a tablet image using the straight line Hough transform [61]. By identifying the peaks in the Hough transform (Hough voting), we obtain a robust estimate of detected lines in the tablet image (which may consist of separate linear segments) as visualized in Fig. 6.5. We expect lines on the same tablet to be similar oriented and have a minimum distance to each other, therefore we require detected lines to be close in orientation (+/- 3 degree), but far enough apart vertically (minimum 50 pixels). Initially, we only search for lines that are roughly horizontal and thus focus the search range in the Hough transform (green dashed vertical lines) to 83 to 97 degrees. We use the median angle of all lines (peaks in Hough transform) that we find in a first run in order to refocus the line search on a tighter range of line orientations (yellow dashed vertical lines) in a second run and thus avoid outliers. Further, we merge lines if they intersect or are almost parallel. Finally, detected lines are associated with their segmentation mask which can provide additional information.

6.4.3 Sign Placement Method

In the first iteration of iterative learning, the sign placement hypothesis (placed detections) is purely created from line detections and sign size statistics, and thus, serves as initial training data for the cuneiform sign detector. In later iterations, the sign placement method additionally leverages aligned detections in order to localize unaligned transliterated signs with higher precision.

To generate the initial placed detections, we first align transliteration lines with detected lines by matching them in a greedy fashion from top to bottom, i.e. the first transliteration line is assigned to the first detected line etc. If aligned detections are available, we rely on the assignments of the line-level alignment method as described in Sect. 6.4.5.

Having solved the line-level alignment, the placed detections are generated line by line. For the sign placement method, a line is defined as the linear segment between a start and end point, which are localized in the tablet image with the help of line detections and its segmentation mask. If aligned detections are available, they provide additional reference points for sign placement besides the start and end of a detected line. Aligned detections effectively split a full line into smaller individual line segments of unaligned transliterated signs whose start and end points are the respective borders of the bounding box of the aligned detections. Knowing the start and endpoint of a line segment, we place the center points of corresponding transliterated signs on the line so that they span the full length of the line. The bounding box size of each placed detection is estimated in the following way: The sign height is estimated from the average line distance in the tablet image. Since sign widths strongly vary across sign code classes, we compute the sign width by multiplying a class-specific normalized sign width from a precomputed sign size statistic with the sign height. We collect the sign size statistic by measuring the relative length and width of sign characters in a realistic cuneiform Unicode font [215].

If aligned detections are available, we additionally filter the resulting placed detections twofold in order to increase their precision: 1) Placed detections that are more than three signs distant from the nearest line start or end point (e.g. aligned detection) are ignored. 2) Placed detections are only included, if at least two aligned signs are present in their line.

6.4.4 Sign Detector Training

We implement the SSD detector [137] with default boxes that cover four aspect ratios (3/5, 1/1, 2/1, 3/1) and three scales (1, 1.26, 1.59) that are adjusted for the various shapes of cuneiform signs. During training we use online hard-negative mining as described in [137] which maintains an one-to-three ratio of positive and negative boxes by keeping only the hardest negatives. Similarly, we follow in their choice of loss functions by using a cross-entropy loss for the classification head and a smoothed L1 loss for the bounding box regression head. In the implementation of the feature pyramid network, we deviate from [134] and only detect cuneiform signs across two feature scale levels that assume signs to fit in a window (anchor box) of 128×128 or 256×256 pixels, respectively. We do not search for cuneiform signs across all feature levels, since after preprocessing, tablet images are resized to match a sign height of 128 pixels as described in Sect. 6.5.2.

When training the sign detector, we always pre-train the backbone network on the simpler task of sign code classification and then use the pre-trained backbone as initialization for detector training. For this pre-training the backbone network is extended with an average pooling layer, followed by a liner layer with as many neurons as classes and a softmax function that provides class predictions. The training of the sign code classifier on the generated sign annotations from our weakly supervised approach is performed like training on supervised data, in contrast to the weakly supervised training of the sign detector.

Standard training of an object detector requires fully annotated images, however, the placed and aligned detections only cover a subset of all visible signs in tablet images. We use the line segmentation mask to prevent the incorrect labeling of foreground bounding boxes as background (false negatives) and to include many true hard negatives found at the border of the mask for training. In particular, bounding boxes are ignored, if both of the following conditions hold true: 1) The box center is located on a segmentation mask of a detected line, and 2) the maximum intersection-over-union (IoU) with any aligned detection box is in the range $[0, 0.35)$. For all other boxes the standard rules for detector training apply.

After training on weakly-supervised data, the sign detector can be fine-tuned with manual sign annotations which represents semi-supervised training. Annotated bounding boxes provide accurate sign localization also in difficult cases and thus mitigate the problem of localization drift as shown in Fig. 6.18. If all signs in an image are annotated, no masking based on the line segmentation is required which further increases the quality of hard negatives available for training.

6.4.5 Line-level Alignment

Given the detected lines in the tablet image and the lines in the transliteration, the goal is to find the correct line-level alignment. This optimization is necessary because of errors in line detection (e.g. false positives) and the discrepancy between tablet image and transliteration (e.g. transliteration contains signs that are not visible in image and vice versa).

We draw inspiration from the sentence alignment problem in natural language processing, and thus adapt the Bleualign algorithm [193] which formulates the alignment problem (between sentences of an original text and the sentences of its translation to another language) as a longest path search. For all combinations of detected lines and transliteration vectors (transliterated lines), an alignment score is computed and stored in a score matrix. The rows and columns of the score matrix correspond to detected lines and transliterated lines sorted by line number in descending order. We construct a directed grid graph G in the size of the score matrix, where diagonal edges resemble "matches" and horizontal or vertical edges "skips". The alignment is obtained by optimizing the path in G from the upper left to the lower right node. The alignment scores from the score matrix define the cost of matches and skips implied by the selected path. Since G is a Direct Acyclic Graph (DAG) with positive edge weights the optimization is performed in linear time using the topological sort algorithm with complexity $\mathrm{O}(|V|+|E|)$, where $|V|$ and $|E|$ are the number of vertices and edges in G.

The line-level alignment is the result of this longest path search through a score matrix. A solution to the path search problem can consist of multiple longest segments that do not need to be connect. Thus, problematic line-level alignments tend to be avoided and only the most reliable ones are retained for sign-level alignment.

Alignment Score Functions

The choice of the alignment score function is key parameter for the alignment algorithm. The original Bleualign algorithm makes use of a machine translation system to map a source sentence into the language of the target sentence, before computing their BLEU [165] score as alignment score. In the case of line-level alignment, we need to associate a detected line in image-space with a transliteration vector in sign-code-space. We rely on the raw detections of the sign detector (in image-space) to obtain a predicted transliterated line (in sign-code-space) which can be compared with the true transliteration vector by computing an alignment score. Besides the BLEU score, we consider alternative alignment score functions that are not only dependent on the output of our sign detector, but also incorporate geometric constraints. In Fig. 6.6 three different score matrices and the

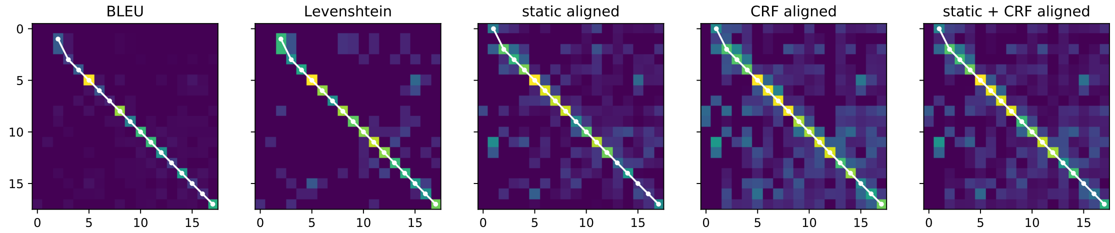


Figure 6.6: Score matrices and longest paths of the line-level alignment with different score functions. The rows of the score matrices correspond to detected lines (sorted by y-coordinate) and the columns correspond to lines in the transliteration. A higher score is indicated by a lighter color. A point in the path (white dot) assigns a detected line to a transliterated line.

optimized longest paths are visualized. Each matrix is computed with a different score function that quantifies the quality of the assignment between detected and transliterated lines. We consider the following score functions:

- BLEU score: For a transliterated line and a corresponding detected line candidate we select a subset of the raw detections using the detected line as filter mask. The selected raw detections are sorted by their x-coordinate and converted into a detection vector of sign code classes that is an estimate of the transliteration vector. The BLEU score for the tuple of the transliteration and detection vector is computed by comparing their 1-gram to 4-gram statistics. The BLEU score is always in the range $[0, 1]$.

- Levenshtein (edit) distance score: Like in the case of the BLEU score, we first obtain a detection vector from the raw detections. The Levenshtein distance $d_{lev}(\boldsymbol{y}, \boldsymbol{t})$ between the detection vector $\boldsymbol{y} = (y_1, \cdots, y_m)$ (candidate) of length m and the transliteration vector $\boldsymbol{t} = (t_1, \cdots, t_n)$ (reference) of length n is computed by counting the number of insertion, delete and replace operations in order to transform the candidate vector into the reference vector (cf. Sect. 7.4.2). To ensure that the Levenshtein distance score is always in the range $[0, 1]$ (independent of the length of the two vectors), we normalize the Levenshtein distance in the following way:

$$d_{lev}(\boldsymbol{y}, \boldsymbol{t}) / \max(m, n) \quad . \tag{6.1}$$

- Static sign-level alignment score: We rely on the initial sign placement hypothesis (using only the detected line, not aligned detections) and search for each placed detection in the direct neighbourhood around its bounding box for a matching raw detection of the same sign code class. For measuring the offset of the raw detection from the center of the placed detection, we use the standardized Euclidean distance $\|\cdot\|_S$ (equivalent to the special case of Mahalanobis distance with only diagonal entries in covariance matrix) with the variance along the y-axis and x-axis defined by the height and width of the bounding box of the placed detection. We limit the search space to two times the average sign height h_s as measured in the standardized Euclidean space (which resembles a search ellipse in the image). For each line in a tablet image, there is a transliteration vector $\boldsymbol{t} = (t_1, \cdots, t_n)$ and a corresponding ordered set of placed detections $(\boldsymbol{b}_1^p, \cdots, \boldsymbol{b}_n^p)$ that provide for each line position i a sign code class t_i and an initial hypothesis $\boldsymbol{b}_i^p$ of the signs' location, where $\boldsymbol{b}_i^p$ defines the bounding box and $ctr(\boldsymbol{b}_i)$ its center. We search in the set of all raw detections $\mathcal{B}_i^r$ of class t_i, the raw detection $(\boldsymbol{b}_i^*, p_i^*) \in \mathcal{B}_i^r$ with bounding box $\boldsymbol{b}_i$ and confidence p_i that minimizes

$$\min_{(\boldsymbol{b}_i, p_i) \in \mathcal{B}_i^r} \|ctr(\boldsymbol{b}_i) - ctr(\boldsymbol{b}_i^p)\|_S \ . \tag{6.2}$$

For the transliteration vector $\boldsymbol{t}$ of a line with n signs, we obtain the static alignment score L^{static} by computing

$$L^{static} = \sum_i^n (1 - p_i^*) + \min\left(\|ctr(\boldsymbol{b}_i^*) - ctr(\boldsymbol{b}_i^p)\|_S,\, 2h_s\right)/h_s \ . \tag{6.3}$$

As the static alignment score varies according to the number of transliterated signs, we divide by the worst-case score (of this length) in order to obtain the normalized static alignment score.

- CRF sign-level alignment score: The optimization of the sign-level alignment minimizes the energy function of the CRF model that rates the quality of the achieved match between raw detections and transliterated signs of the line. Since the energy value E of the sign-level alignments depends on the number of transliterated signs in the line, we divide E by the energy of the worst-case assignment (of this length) in order to obtain a normalized energy value E_{norm} that is comparable across different lines.

While the first two score functions compare transliteration vectors with their estimates, the later two score functions also incorporate geometric information in terms of sign size statistics and search across a larger set of raw detections to score the line-level assignment. All proposed score functions rely on line and sign detections, and thus depend on the quality of these detectors. Since the sign detector produces many false positives in early iterations of iterative training, the quality of all line-level alignment scores suffers. To improve the precision of the line-level alignment, we also consider combining two score functions by taking the intersection of their respective paths as the final line-level alignment.

6.4.6 Sign-level Alignment

For a given the line-level alignment between a detected line and its corresponding transliteration, the goal of sign-level alignment is to find an assignment between the raw detections in the tablet image and the signs in the transliteration vector. We follow the idea of part-based graphical models [66] and build a CRF model for a line, whose underlying graph structure is determined by the transliteration vector.

Formally, we represent the transliteration vector T with n signs as a fully connected graph $G = (V, E)$ as shown in the example in Fig. 6.7. Each node $a \in V$ we associate with a sign $t_a \in T$ in the transliteration and a random variable X_a that takes values $x_a \in C_a$, where C_a is the set of detections of sign code class t_a. Additionally C_a includes a candidate ϵ that indicates a sign that could not be aligned. The alignment of a transliteration with sign detections is then given by the vector $\mathbf{x} = (x_a)_{a \in V} \in C^n$, where we use C^n to describe the set of all possible alignments (ie. assignments to all random variables). The energy

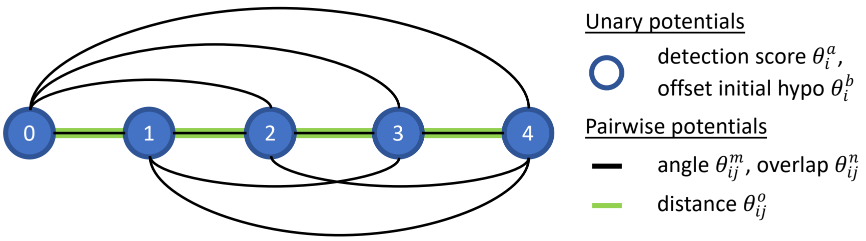

0
1
2
3
4
Unary potentials
detection score θ_i^a,
offset initial hypo θ_i^b
Pairwise potentials
angle θ_{ij}^m, overlap θ_{ij}^n
distance θ_{ij}^o

Figure 6.7: The CRF has a fully-connected graph which models appearance and geometric constraints of a line. The graphical model for a transliteration of five is shown. Each random variable (circled number) represents a sign in the transliteration. There are unary potentials for each variable depending on detection score and offset from a initial sign hypothesis. Connections between variables describe pairwise potentials that enforce geometric constraints found in a line, such as angle, overlap and distance between signs. The distance potential (green connections) only affects connections of neighbouring signs.

function $\mathbb{E} : C^n \rightarrow \mathbb{R}$ of the CRF maps any alignment to a real number. The function $\mathbb{E}(\cdot)$ is defined as the sum of all unary $\theta^U(\cdot)$ and pairwise potential terms $\theta^P(\cdot,\cdot)$ as follows:

$$\mathbb{E}(\mathbf{x}) = \sum_{i \in V} \theta_i^U(x_i) + \sum_{(i,j) \in E} \theta_{ij}^P(x_i, x_j) \tag{6.4}$$

The unary terms θ^a, θ^b take into account the detection confidence and the offset from an initial sign hypothesis respectively, and are combined using weights λ_a and λ_b as follows:

$$\theta_i^U = \lambda_a \theta_i^a + \lambda_b \theta_i^b \tag{6.5}$$

The pairwise terms $\theta^m, \theta^n, \theta^o$ constrain the overlap between sign detections, distance between bounding boxes as well as angle between signs and detected line respectively, and are combined using weights λ_m, λ_n and λ_o as follows:

$$\theta_{ij}^P = \lambda_m \theta_{ij}^m + \lambda_n \theta_{ij}^n + \lambda_o \theta_{ij}^o \tag{6.6}$$

Unary terms For detector confidence we define the unary term $\theta_i^a(x_i) = \exp((1 - score(x_i))/\sigma_a) - 1$, where $score(\cdot)$ returns confidence of the assigned detection. For the offset from the initial sign placement hypothesis, we define the unary term $\theta_i^b(x_i) = hypodist(x_i))$, where $hypodist(\cdot)$ returns the Euclidean distance between the estimated position of the sign from the initial hypothesis and the location of the assigned detection. We initial sign hypothesis is generated by the sign placement method described above.

Pairwise terms For the overlap between sign detections we define the pairwise term $\theta_{ij}^m(x_i, x_j) = \exp(iou(x_i, x_j)/\sigma_m) - 1$, where $iou(\cdot,\cdot)$ computes the intersection-over-union between the bounding boxes of the assigned detections. For the distance between bounding boxes, we define the pairwise term $\theta_{ij}^n(x_i, x_j) = \exp(boxdist(x_i, x_j)/\sigma_n) - 1$, where $boxdist(\cdot,\cdot)$ computes the distance between the bounding boxes of the assigned detections. The potential is only nonzero for connections between neighbouring signs (green connections in Fig. 6.7). For the angle between signs and the detected line, we define the pairwise term $\theta_{ij}^o(x_i, x_j) = \exp(angle(x_i, x_j)/\sigma_o) - 1$, where $angle(\cdot,\cdot)$ computes the angle of the vector that connects the bounding boxes of the assigned detections and the vector of the detected line.

Outlier treatment If a random variable X_a takes on the value $x_a = \epsilon$, it incurs a fixed outlier penalty λ_p independent of any neighbouring nodes. This outlier class deals with difficult alignments (e.g. no matching detection).

Inference After computing the unary and pairwise potentials for a given line, we use the sequential Tree-Re-Weighted message passing (TRW-S) algorithm [116] to minimize the energy function in (6.4). The solution aligns a detections with each sign in the transliteration vector except for signs that have been assigned to the outlier class. Our implementation makes use of the OpenGM framework [9].

6.5 Results: Learning Approach

Our experiments have the goal to study the individual components of our learning approach as well as the full iterative training of a cuneiform sign detector on a large scale. To conduct our experiments, we rely on the cuneiform sign detection dataset introduced in Sect. 5.5.1. We start with a description of the training and testing protocol used in our experiments. Then we present six experiments to demonstrate the effectiveness of our learning approach: The first three experiments investigate the effect of the components and the effect of iterative training. The fourth experiment illustrates the effect of available annotations on the learning performance. The last two experiments explore the learned feature representation and if it improves the sign detection of a different cuneiform script.

6.5.1 Evaluation Metrics

For the evaluation of sign detections, we follow the standard evaluation protocol for object detection[98]. A sign detection is considered a TP, if 1) its bounding box and a ground truth box overlap more than 50% measured as Intersection-over-Union (IoU), and 2) their labels match, otherwise it is considered an FP. Before evaluation, we use class-wise Non-Maximum Suppression (NMS) with 0.3 as IoU threshold in order to remove improbable and noisy detections.

We use slightly different metrics for raw sign detections of the sign detector and generated sign detections (aligned & placed detections). The performance on individual sign code classes is measured in terms of Average Precision (AP) which is the standard metric for object detection [98]. The overall sign detector performance is measured in terms of mean AP (mAP), the average of the AP values of all 186 sign code class. The quality of the aligned & placed sign detections are evaluated in terms of precision, recall and F2-score, each of which is averaged across the 186 sign code classes. We emphasize the recall of the generated sign annotations (F2-score weights it twice as important), since it is essential for iterative learning: The detector should not just exploit a few sign code classes that are easy to detect, but become proficient across all of them. Exploring examples that are difficult-to-learn or belong to rare sign code classes is particularly important due to the natural sign code class imbalance in the dataset.

Table 6.1: The configuration of λ and σ parameters for the sign-level alignment in Sect. 6.4.6 used for all experiments.

outlier	score		hypodist	iou		boxdist		angle	
λ_p	λ_a	σ_a	λ_b	λ_m	σ_m	λ_n	σ_n	λ_o	σ_o
25	12	0.88	1	0.4 (1.5)	0.4 (0.05)	5	3	2 (0.2)	0.6 (0.1)

6.5.2 Training and Testing

Preprocessing of Tablet Images First, we convert all tablet images to gray-scale and compute line detections using our method. Then, we resize the tablet images so that the height of a cuneiform sign is about 128 pixels. We obtain the necessary scaling factor for each tablet image by estimating the average sign height. We approximate the average sign height with the average distance between detected lines. If there is a transliteration available, we also estimate upper and lower bounds for the sign height in the table image by dividing the tablet image height by the number of transliterated lines and by dividing the tablet image width by the length of the longest transliterated line respectively. If the estimated sign height is out of bounds, we use the nearest bound instead. To refine the line detections and sign height estimates, we rerun the steps of preprocessing on the scaled version of the tablet images once.

Training parameters All neural networks of our approach are trained using a standard stochastic gradient descent (SGD) optimizer with momentum 0.9 and weight decay of 1e-04. For each training of a task, the configuration including learning rates (LR) and other parameters is described in the following.

Sign Code Classification The sign code classifier is trained with LR 0.01 until the train error plateaus, then LR is decreased by factor 0.1. This is repeated two times. As input data 224x224 patches are randomly cropped from a sign bounding box that has been context-padded to 256x256 box without altering the aspect ratio.

Line Segmentation The line segmentation network is trained with LR 0.01 which is decreased like in the case of sign code classification training. As input data 227x227 patches are randomly cropped from 256x256 patch that is sampled following the strategy described in Sect. 6.4.2.

Sign Detection The sign detector is always trained for 50 epochs with LR 0.001 with online hard-negative mining as described in [137]. As input data 512x512 patches are randomly cropped from a tablet image which has been split in 600x600 patches that overlap by 200px. As data augmentation, we use randomly resized crops with scale range $[0.65, 1]$.

Detector Fine-tuning In the case of semi-supervised training, the regular training on weakly supervised data is followed up by a fine-tuning step on annotated samples. The sign detector is fine-tuned on the annotated samples for 20 epochs with a reduced LR 1e-04.

Alignment Method The configuration of λ and σ parameters defined in Sect. 6.4.6 is shown in Table 6.1. The values in the brackets correspond to the parameter configuration of pairwise potentials between two non-neighbouring signs. Having aligned detections we compute for each tablet its alignment ratio (AR), the ratio between aligned detections and signs in transliteration. We only keep aligned detections from tablets with $AR > 0.3$ in order to focus on the most reliable alignments.

Table 6.2: Performance of different approaches to line-level alignment on test set.

#	Score name	#TPs	#FPs	F2-score	mAP
0	optimized	1870	279	0.808	55.0
1	gt lines	1485	262	0.736	43.6
2	greedy	1280	266	0.677	35.8

Table 6.3: Performance of different score functions for line-level alignment on test set.

#	Score name	#TPs	#FPs	F2-score	mAP
1	CRF aligned	1870	279	0.808	55.0
2	static aligned	1842	279	0.806	54.7
3	CRF + static aligned	1834	272	0.804	54.3
4	Levenshtein	1812	248	0.805	53.5
5	BLEU	1774	246	0.802	52.7

6.5.3 Effect of Line-level Alignment

In our first experiment, we study the impact of our proposed line-level alignment method. To quantify the gain, we compare the optimized approach discussed above with two baseline approaches as shown in Table 6.2. The first baseline (greedy) uses the detected lines and assigns the lines from the transliteration from top to bottom. The second baseline (greedy w/ GT lines) works as the greedy approach, but relies on ground truth lines (manual annotation of all visible lines in the tablet image) instead of detected lines. The line-level optimized approach significantly outperforms the two baseline methods. Despite using ground truth lines the second baseline performs worse than the optimized approach. The problem is the discrepancy between tablet image and transliteration: If there are more/less visible lines (gt lines) than lines in the transliteration, the ground truth lines are misaligned.

To compare the performance of the different score functions, we evaluate them in the context of the image-transliteration alignment on the test set. While keeping the initial raw detections and the sign-level alignment parameters fixed, we vary the score function and report the achieved quality of the alignment in terms of mAP and F2score in Table 6.3. We find that the CRF sign-level alignment score performs best, followed by the static sign-level alignment, then the Levenshtein, and at last the BLEU score. The combination of the static and the CRF sign-level alignment scores performs slightly worse than the CRF sign-level alignment scores on its own. The CRF aligned score probably works at least as well or better than the static aligned score, since it is based on the geometric prior used for static sign-level alignment (both use initial placed detections).

In Fig. 6.8 we show the assignment produced by the line-level alignment using the path of the CRF aligned score matrix shown in Fig. 6.6. If a detected line has been assigned a line of the transliteration (a point of the path in the score matrix), the transliteration line number is displayed next to the line. A '-1' indicates that the detected line could not be assigned to a transliteration line.

6.5.4 Effect of Alignment and Placement Method

In our second experiment, we analyze the impact of the alignment and placement methods on the quality of sign detections. For this purpose, we track the change in precision, recall and F2-score from raw detections to aligned detections and eventually to the union of

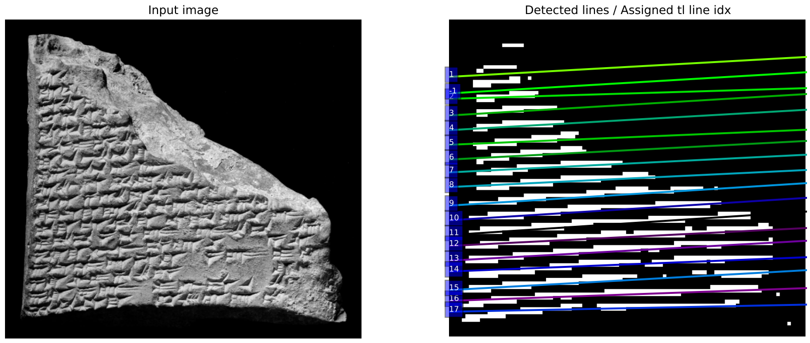


Figure 6.8: The line-level alignment produces an assignment of transliteration lines to detected lines. The assigned transliteration line index is displayed at the tip of each detected line.

aligned & placed detections. The experiment is conducted on the test set, starting from the raw detections of a sign detector obtained after three rounds of iterative training. Then the alignment and placement methods are applied consecutively.

In Fig. 6.9 we compare the quality of raw detections to aligned detections and to the union of aligned & placed detections. Using the weak supervision from transliterations helps to increase precision by aligning TP detections while discarding FPs. The alignment method produces aligned detections with a significantly increased precision (reduction in FPs) and a recall very close to the one of raw detections (while maintaining TPs). Since the alignment method effectively serves as a filter for the raw detections, the recall of aligned detections cannot surpass the recall of raw detections. In contrast, the placement method increases the recall of aligned & placed detections by filling the gaps between aligned detections with unaligned signs from the transliteration. While the placement method increases the recall of aligned & placed detections, it also leads to a decrease in precision. We find that our deep learning method is robust to the small loss in precision and the increase of recall helps with learning difficult and unexplored classes during iterative training.

Example of Sign-level Alignment

Fig. 6.10 visualizes the sign-level alignment of a tablet image that is performed for each pair of transliterated line and detected line (provided by the line-level alignment). The visualization shows the aligned detections (green squares), the initial placed detections (cyan diamonds), and the individual sign detection bounding boxes colored according to their detection confidence. It is apparent that the initial placed detections do not account for gaps in the text lines, in contrast to the aligned detections that adapt to the presence of gaps properly. While the majority of aligned detections have a high confidence (bright yellow), some lower confidence detections (dark purple boxes) are successfully aligned as well. The ability to promote these low confidence detections highlights the importance of the alignment method in our iterative learning approach: It not only exploits sign detections with high confidence to boost the precision of aligned detections, but also incorporates low confidence detections that help the sign detector learn rare and difficult sign code classes and thus boost its recall in future iterations.

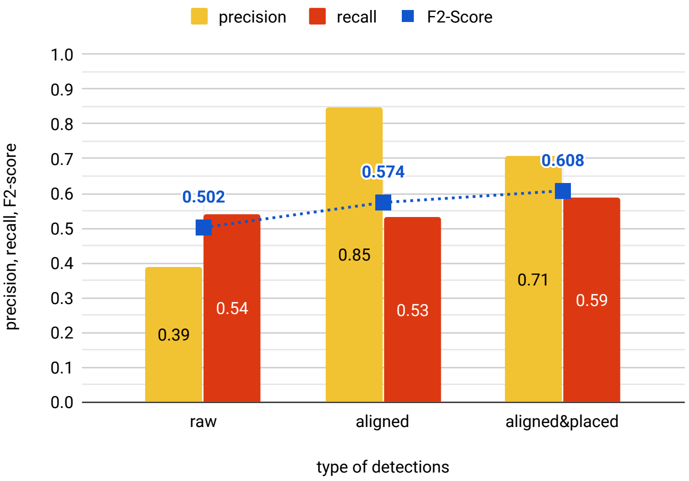


Figure 6.9: Effect of alignment and placement method on quality of detections. Our alignment method leverages weak supervision from transliterations to boost precision of aligned detections (aligned) compared to raw detections (raw). The placement method further adds detections (placed) that are combined with aligned detections. We report the change in precision, recall and F2-score of the detections produced by our method on the test set.

6.5.5 Effect of Iterative Training

Our third experiment investigates how iterative training gradually improves the sign detector in order to eventually detect all signs in a tablet image. To understand the effect of iterative training, we report for each iteration the composition of the aligned & placed detections in Fig. 6.11a, their F2-score as well as the performance of the sign detector after training in terms of mAP in Fig. 6.11b. Iterative training naturally suffers from a phenomenon called *drift* [198], that refers to performance degradation of the sign detector due to small errors in the aligned & placed detections that build up over the course of training. If the drift effect outweighs the performance gain between iterations, we stop iterative training.

The first and second training iterations of this experiment are weakly supervised without any manual sign annotations. In the first iteration, only placed detections are available for detector training of which less than twelve percent are TPs when evaluated on the test set. Despite the low precision of the placed detections, the sign detector training produces a first detector with 20.7 mAP. By aligning more than 80k detections, our weakly supervised approach increases the sign detector performance significantly (45.3 mAP) in the second iteration. An additional third iteration of purely weakly supervised training fails to improve performance. In particular, we find that sign localization of aligned & placed detections worsens due to poorly placed bounding boxes and this drift cancels out the positive effects of iterative training (see also the experiment in Sect. 6.6.2). On the third iteration, we add 745 manual sign annotations from Train-BB to the more than 100k already generated annotations (aligned & placed detections) as shown in Fig. 6.11a. This corresponds to the semi-supervised case of iterative training that entails fine-tuning the sign detector on a small number of manual annotations during each iteration. By semi-supervised learning, the localization drift is reduced and the performance increases up to 63.2 mAP. On the

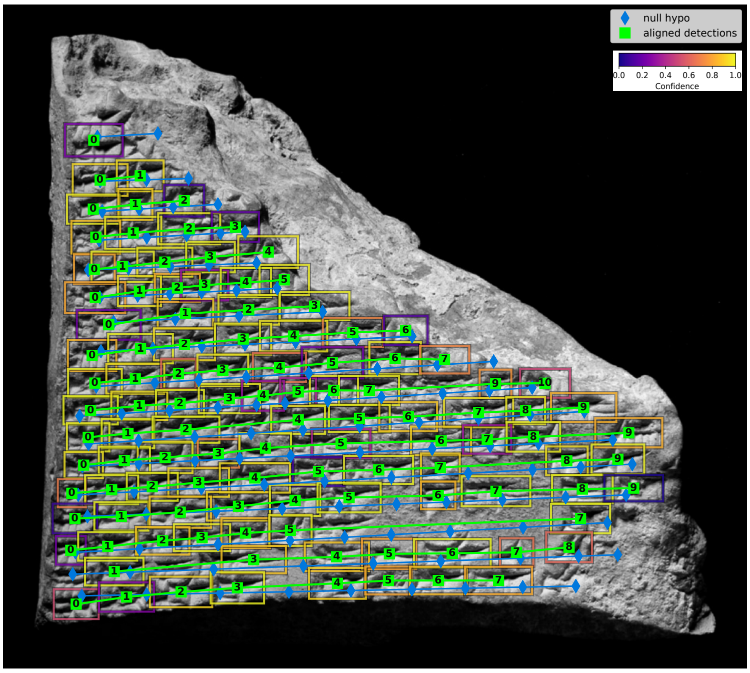


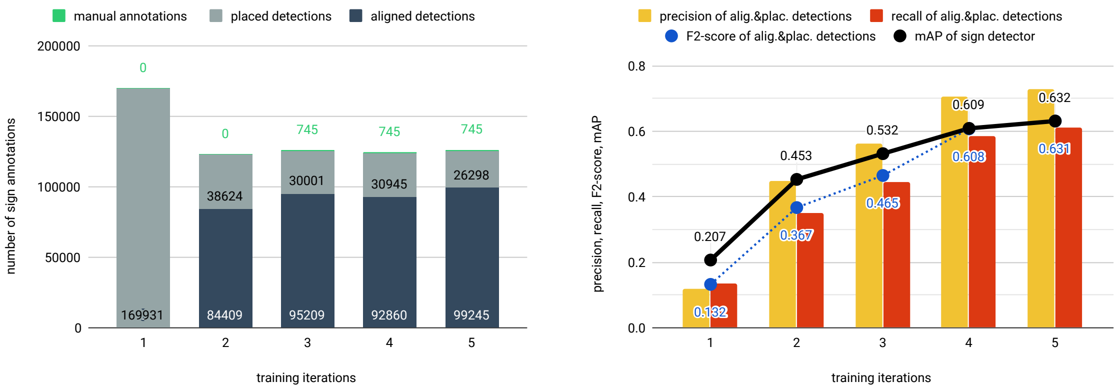
Figure 6.10: Sign-level alignment for specific tablet. The chain of green squares represents aligned signs of the transliteration vector and the numbers indicate the position of the sign in the vector. The chain of cyan diamonds represents the initial placed detections from the sign placement step which provide an initial hypothesis for the localization of signs. Additionally, the detection bounding boxes indicate the detection confidence, where a darker purple color indicates low and a bright yellow color indicates high confidence. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

fourth iteration of semi-supervised training, we find no performance improvement and thus stop iterative training. We again observe that further improvement is inhibited by drift that causes small errors to accumulate, and can neither be corrected by the alignment nor by the fine-tuning step.

In weakly supervised as well as in semi-supervised training, the performance gain is strongly linked to the gradual improvement of aligned & placed detections as measured by the F2-score. At the end of semi-supervised training, over sixty percent of the transliterated signs in train set B are correctly grounded in the tablet images, providing over 100k training samples with an F2-score of 0.63. While there are no aligned sign detections in the first iteration, the ratio of aligned detections increases fast and then levels off. Over the course of training, exploration is gradually replaced by exploitation: The more the sign detector improves, the more raw sign detections are available that are suitable for alignment.

Example of Aligned and Placed Detections

Fig. 6.12 visualizes how the composition of aligned and placed detections changes at the example of a single tablet image during iterative training. Initially, all generated annotations are placed detections, however, over the course of iterative training placed detections are quickly replaced by aligned detections. The alignment produces a large



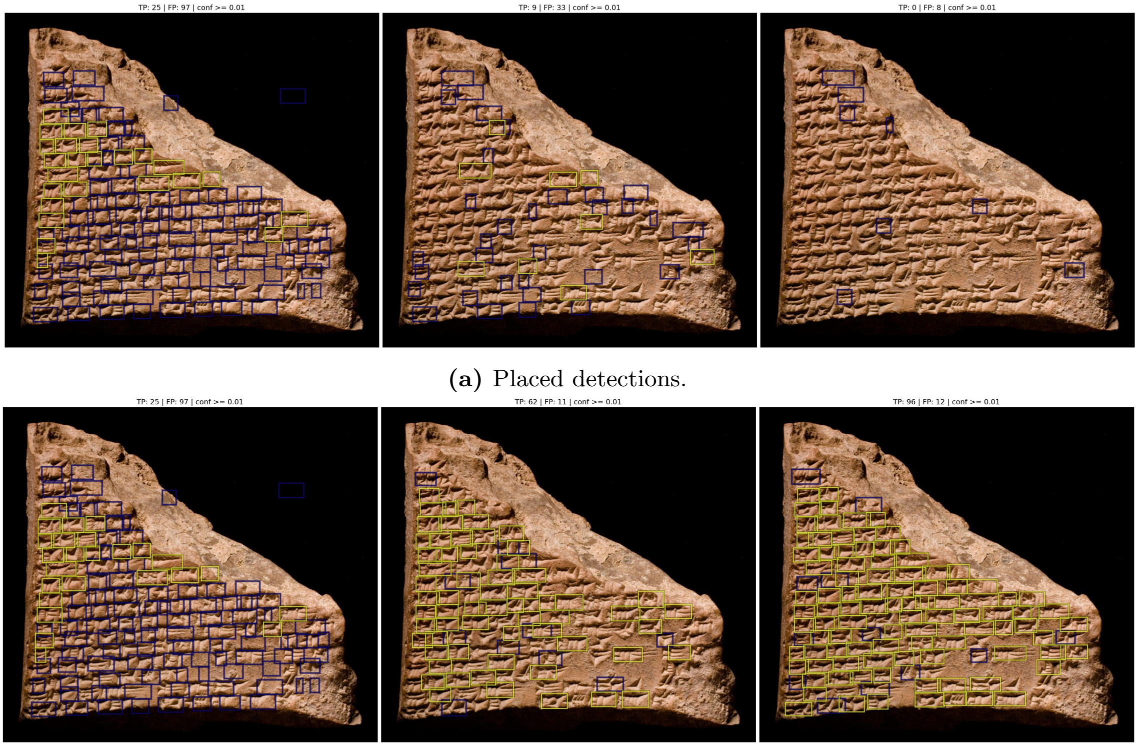
(a) Effect on ratio of generated detection types. **(b)** Effect on detection performance.

Figure 6.11: Effect of iterative training on ratio of generated detection types and detection performance. The first two iterations (1–2) are weakly supervised (w/o manual annotations) followed by three semi-supervised iterations (3–5) which include fine-tuning on 745 manual sign annotations. (a) The ratio of detection types varies between iterations. The training starts with only placed detections which are replaced with an increasing number of aligned detections over the course of training. (b) The precision and recall of the generated sign annotations and the performance (mAP) of the sign detector on the test set. To improve the sign detector, the quality and quantity of generated sign annotations needs to increase over the course of training.

number of TP detections (yellow boxes) compared to the initial placed detections, which is apparent in the coloring of the bounding boxes.

6.5.6 Effect of Available Annotations

In our fourth experiment, we evaluate the influence of manual sign annotations on the performance of the sign detector. The performance of a sign detector, after iterative training has converged, is compared with the performance of a sign detector after purely supervised training on manual sign annotations. We consider six different training configurations with zero (A), 463 (B), 745 (C), 1472 (D), 2910 (E) or 4663 (F) manual sign annotations respectively. For our iterative learning approach, configuration A corresponds to weakly supervised training and configuration B–F correspond to semi-supervised training. Similar to the experiment in Sect. 6.5.5, we train the weakly supervised case for two iterations and the semi-supervised cases for five iterations. The results are visualized in Fig. 6.13. Overall the detection performance improves with additional manual annotations as expected. However, our iterative training significantly outperforms purely supervised training across all settings. Even if 4663 manual annotations are available (F), our iterative training surpasses supervised training by a large margin (65.6 mAP compared to 33.9 mAP). Using our iterative learning approach, the dependency on supervised training data is strongly reduced. Already the semi-supervised setting with 745 annotations (C) produces a sign detector performance that would require ten thousands of manual annotations in the purely supervised case.


TP: 25 | FP: 97 | conf >= 0.01
TP: 9 | FP: 33 | conf >= 0.01
TP: 0 | FP: 8 | conf >= 0.01
TP: 25 | FP: 97 | conf >= 0.01
TP: 62 | FP: 11 | conf >= 0.01
TP: 96 | FP: 12 | conf >= 0.01

(a) Placed detections.

(b) Aligned detections.

Figure 6.12: (a) Placed and (b) aligned detections in the same tablet image at three consecutive iterations of iterative training. The leftmost images of both rows depict the initial placed detections for easier comparison. TP and FP detections are visualized as yellow and blue boxes respectively. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

6.5.7 Qualitative Detection Results with Visual Feature Activation

In our fifth experiment, we analyze individual sign detections and gain insight into the detector's decision making. Fig. 6.14 depicts detection results for a single sign code class (MZL no. 490) using a sign detector at two different iterations of iterative training. We apply the trained sign detector of the second experiment after the 2nd and 5th iteration to the test set. The resulting sign detections are classified as TP or FP based on ground truth annotations. The eight most confident (according to the sign detector) TP and FP detections are shown in the columns of Fig. 6.14 sorted by their confidence value. Further, we follow the analysis as introduced in [98] and classify FP detections into three categories: Localization error (Loc), background confusion (BG) and similar class confusion (Cls). As expected, the detection quality and confidence increases from 2nd to 5th iteration of iterative training which is visible in the TP columns. From studying the FPs, we find that the later detector makes more reasonable mistakes. In particular, sign localization is improved in the detector of the 5th iteration and all remaining FPs are due to class confusion, where the predicted class and actual ground truth class look very similar.

Finally, we analyze which visual features are captured by the learned representation of the sign detector when it decides on the sign code class of cuneiform signs. We employ a technique called grad-CAM [192] that maps feature activations of the detector network

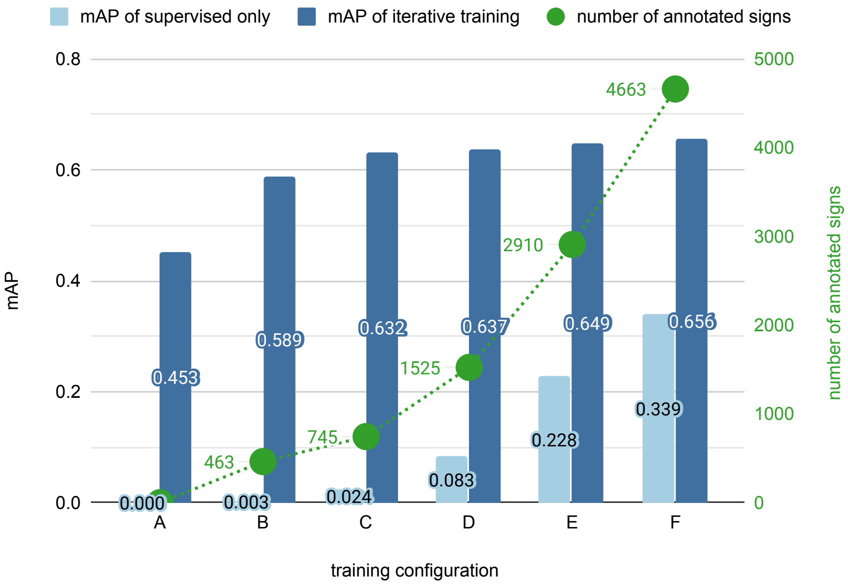


Figure 6.13: Effect of manual sign annotations on detection performance comparing purely supervised and our iterative training. There are six training configurations which differ in the number of available annotations. Configuration A corresponds to the weakly supervised case of iterative training. Configurations B to F correspond to the semi-supervised case. The detection performance (mAP) is reported on the test set.

back to the image domain. This back-projection can be conditioned on a specific sign code class in order to highlight the most relevant image regions (support) for the sign code class in a heatmap. Since in the case of FPs the false predicted class and the actual ground truth class are known, we can go further in our analysis and focus on the difference between the support of the predicted class (Pred) and the support of the ground truth class (GT). The heatmap of this difference highlights the regions that exclusively contribute to the classification error according to the sign detector.

We visualize the feature activation in the penultimate layer of our adapted MobileNet-v2 backbone (see Sect. 6.4.4). The resulting heatmap is shown as absolute values that are normalized in the range of $[0, 1]$. First, we compute the support with respect to the predicted class $Pred$ and ground truth class GT separately. Then, we compute the difference of support between $Pred$ and GT with the following function: $min(max(Pred - GT, 0), 1)$.

In Fig. 6.14 the third column of each iteration shows heatmaps with respect to the FPs from the 2nd column. In particular, the heatmaps of the 5th iteration of iterative training indicate where the detector hallucinates additional wedges to produce its FP prediction. Many mistakes appear plausible and are related to the difficulty of the fine-grained detection problem. There is no heatmap in the case of a localization error because the predicted and ground truth class are identical. See Fig. 6.22a, Fig. 6.22b, and Fig. 6.23 for additional qualitative detection results including full tablet images.

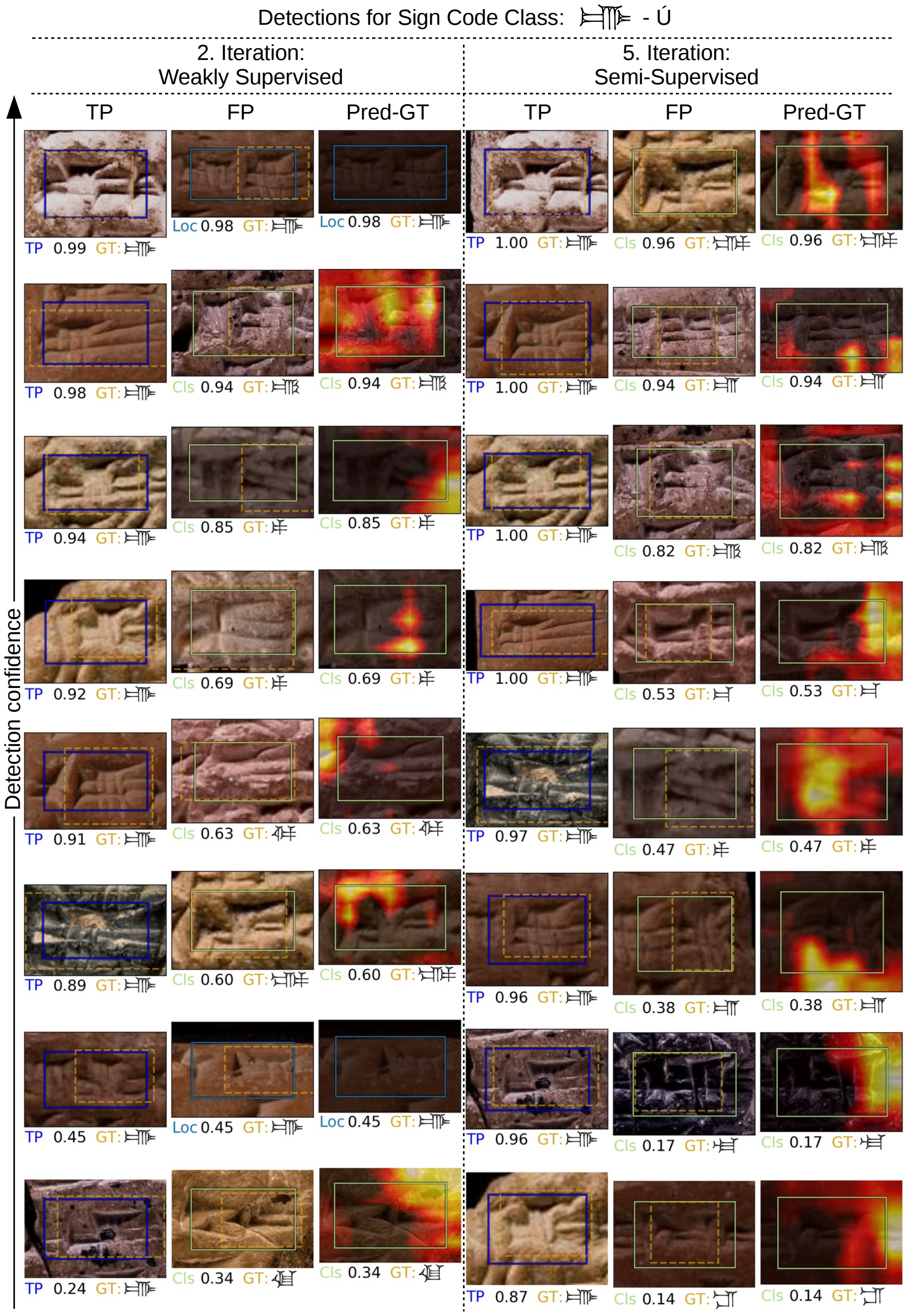


Figure 6.14: Individual sign detections for single sign code class at two different iterations of iterative training. For each iteration we show in the first two columns the eight most confident TP and FP sign detections on our test set. Each sign detection is represented as a solid bounding box, while the golden dashed box indicates the ground truth annotation (with the largest overlap with the predicted box). Below each detection we show the detection category, the detection confidence, and the actual ground truth (GT) sign code class. The detection category is either TP or one of three FP categories: Loc, BG, Cls. The third column of each iteration visualizes for each FP detection of the second column which image area contributed most to the detection error. Despite high intra-class variance and inter-class similarity of cuneiform signs the sign detector produces good results, makes plausible mistakes, and improves further in the semi-supervised case.

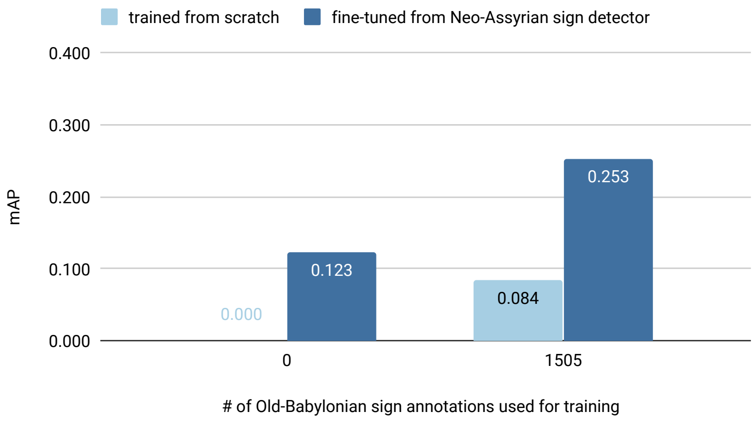


Figure 6.15: Comparison of two ways to train a Old-Babylonian sign detector on 1505 bounding box annotations of Old-Babylonian cuneiform. A sign detector trained from scratch is compared to one fine-tuned from a Neo-Assyrian sign detector that has been pre-trained with our iterative learning procedure. The performance is measured on a test set made up of clay tablets from the CUSAS36 collection [79].

6.5.8 Apply Sign Detector to Different Cuneiform Script

In our final experiment, we investigate if our approach is transferable to Old Babylonian cuneiform. Old-Babylonian shares most of the sign code classes of Neo-Assyrian cuneiform, however, the majority of sign code classes is written differently. For our experiment we annotated 18 Old-Babylonian clay tablets of the CUSAS36 collection [79], of which twelve were used for training (1505 signs) and six for testing (591 signs). Unfortunately, the transliterations of the CUSAS36 collection are not publicly available online at the time of writing, preventing us from employing our weakly-supervised learning strategy. Nevertheless, we show that a sign detector for Old-Babylonian script can be trained successfully and that fine-tuning a Neo-Assyrian sign detector on Old-Babylonian significantly improves the performance. This demonstrates that our sign detector take advantage of the close relationship between different cuneiform scripts.

We use the same training protocol as established for Neo-Assyrian cuneiform. First we train the Neo-Assyrian detector from scratch using only the sign annotations available in the train set. Then we compare this performance with a sign detector that was first trained for Neo-Assyrian and then fine-tuned for Old-Babylonian cuneiform. To provide a lower bound, we also apply a randomly initialized network and the Neo-Assyrian sign detector without any fine-tuning.

The results of the experiment is visualized in Fig. 6.15. A Neo-Assyrian sign detector without any fine-tuning (12.9 mAP) already outperforms a purely supervised sign detector (8.4 mAP). The detection performance of 12.9 mAP can be interpreted as an estimate of the similarity between Neo-Assyrian and Old-Babylonian cuneiform, indicating some overlap between the two. When fine-tuning the Neo-Assyrian sign detector on Old-Babylonian cuneiform, we note a significant increase in performance (25.4 mAP). To close the gap to a Neo-Assyiran detector, more training data is required using manual sign annotations or levering the weak supervision in transliterations by means of our proposed approach.

6.6 Results: Sign Detector Performance

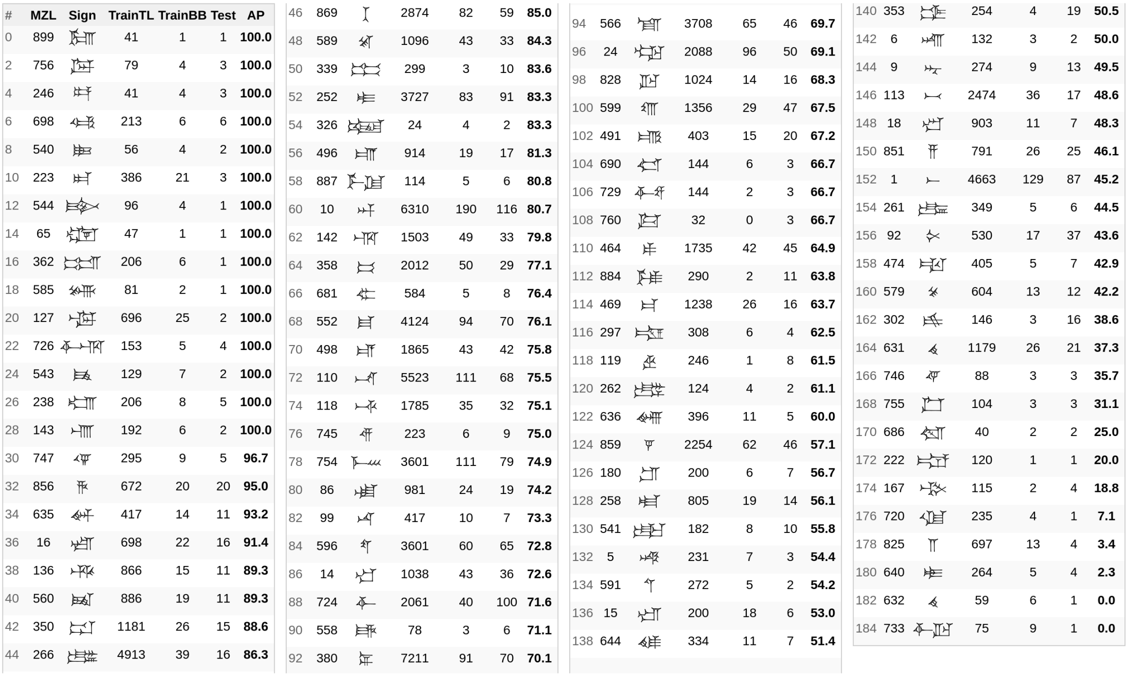
To complement our experimental analysis of the iterative learning approach, we provide details on the performance of the final sign detector produced by our method. In the following we present results on the class-wise detection performance, analyze the false positive errors of the sign detector, in particular the similar class confusions, and show qualitative results on full tablet images as well as for individual sign detections. Moreover, we compare the impact of different backbone architectures as well as ImageNet pre-training, and finally showcase a web application of the cuneiform sign detector.

6.6.1 Class-wise Detection Results

Fig. 6.16 reports the sign detector performance on individual sign code classes evaluated on the Test set in terms of class-wise Average Precision (AP). In contrast, the mean class AP (mAP) provides a metric for the average detector performance across classes. The results are based on the sign detector that we obtain after iterative training in a semi-supervised fashion using all manual sign annotations of Train BB for fine-tuning. The sign code classes are sorted according to their AP (in descending order). We show every second sign code class in the Test set to provide a broad overview of the class-wise sign detector performance, while not overfilling the page.

Since the different cuneiform signs have a strongly imbalanced class distribution (Pareto-distributed), we provide the frequency of signs in the Train TL, the Train BB and the Test set. One might expect that well performing classes usually belong to the group of signs that occur often in the dataset (high support). In Fig. 6.17 we analyze the correlation across the 186 sign classes between detector performance and the support in Train BB and the support in Train TL. We find a light positive correlation for the two properties, where sign code class frequency in Train TL seems to be more important than frequency in Train BB. While the sign detector seems to slightly favor more frequent signs, it still performs well for many less frequent signs.

Further, we make the observation that larger signs that consist of many wedges are detected well, e.g. first column of signs in Fig. 6.16. To investigate this hypothesis, we also analyse the correlation between detector performance and the normalized sign width (which we obtained from our pre-computed sign size statistics) in Fig. 6.17. We find a low positive correlation that indicates that larger composite signs tend to be detected better than smaller ones. This correlation might be caused by the fact that object detectors often perform better on larger objects than smaller ones. Additionally, this finding might be caused by special properties of cuneiform script: Small cuneiform signs are often used as building blocks of larger signs. In the extreme case, a small sign code class may consist of a single wedge (e.g. 152nd sign in Fig. 6.16) which is part of most other sign code classes. Naturally this results in many class confusion errors. In comparison, large signs are less ambiguous due to their complex structure. However, if there are two large signs that are very similar as in the case of 44th and 154th sign, detection performance can suffer despite their size. Since the 44th sign code class occurs very frequent, the detector rather predicts the 44th and thus 154th is often confused.



#	MZL	Sign	TrainTL	TrainBB	Test	AP
0	899	[illegible]	41	1	1	**100.0**
2	756	[illegible]	79	4	3	**100.0**
4	246	[illegible]	41	4	3	**100.0**
6	698	[illegible]	213	6	6	**100.0**
8	540	[illegible]	56	4	2	**100.0**
10	223	[illegible]	386	21	3	**100.0**
12	544	[illegible]	96	4	1	**100.0**
14	65	[illegible]	47	1	1	**100.0**
16	362	[illegible]	206	6	1	**100.0**
18	585	[illegible]	81	2	1	**100.0**
20	127	[illegible]	696	25	2	**100.0**
22	726	[illegible]	153	5	4	**100.0**
24	543	[illegible]	129	7	2	**100.0**
26	238	[illegible]	206	8	5	**100.0**
28	143	[illegible]	192	6	2	**100.0**
30	747	[illegible]	295	9	5	**96.7**
32	856	[illegible]	672	20	20	**95.0**
34	635	[illegible]	417	14	11	**93.2**
36	16	[illegible]	698	22	16	**91.4**
38	136	[illegible]	866	15	11	**89.3**
40	560	[illegible]	886	19	11	**89.3**
42	350	[illegible]	1181	26	15	**88.6**
44	266	[illegible]	4913	39	16	**86.3**
46	869	[illegible]	2874	82	59	**85.0**
48	589	[illegible]	1096	43	33	**84.3**
50	339	[illegible]	299	3	10	**83.6**
52	252	[illegible]	3727	83	91	**83.3**
54	326	[illegible]	24	4	2	**83.3**
56	496	[illegible]	914	19	17	**81.3**
58	887	[illegible]	114	5	6	**80.8**
60	10	[illegible]	6310	190	116	**80.7**
62	142	[illegible]	1503	49	33	**79.8**
64	358	[illegible]	2012	50	29	**77.1**
66	681	[illegible]	584	5	8	**76.4**
68	552	[illegible]	4124	94	70	**76.1**
70	498	[illegible]	1865	43	42	**75.8**
72	110	[illegible]	5523	111	68	**75.5**
74	118	[illegible]	1785	35	32	**75.1**
76	745	[illegible]	223	6	9	**75.0**
78	754	[illegible]	3601	111	79	**74.9**
80	86	[illegible]	981	24	19	**74.2**
82	99	[illegible]	417	10	7	**73.3**
84	596	[illegible]	3601	60	65	**72.8**
86	14	[illegible]	1038	43	36	**72.6**
88	724	[illegible]	2061	40	100	**71.6**
90	558	[illegible]	78	3	6	**71.1**
92	380	[illegible]	7211	91	70	**70.1**
94	566	[illegible]	3708	65	46	**69.7**
96	24	[illegible]	2088	96	50	**69.1**
98	828	[illegible]	1024	14	16	**68.3**
100	599	[illegible]	1356	29	47	**67.5**
102	491	[illegible]	403	15	20	**67.2**
104	690	[illegible]	144	6	3	**66.7**
106	729	[illegible]	144	2	3	**66.7**
108	760	[illegible]	32	0	3	**66.7**
110	464	[illegible]	1735	42	45	**64.9**
112	884	[illegible]	290	2	11	**63.8**
114	469	[illegible]	1238	26	16	**63.7**
116	297	[illegible]	308	6	4	**62.5**
118	119	[illegible]	246	1	8	**61.5**
120	262	[illegible]	124	4	2	**61.1**
122	636	[illegible]	396	11	5	**60.0**
124	859	[illegible]	2254	62	46	**57.1**
126	180	[illegible]	200	6	7	**56.7**
128	258	[illegible]	805	19	14	**56.1**
130	541	[illegible]	182	8	10	**55.8**
132	5	[illegible]	231	7	3	**54.4**
134	591	[illegible]	272	5	2	**54.2**
136	15	[illegible]	200	18	6	**53.0**
138	644	[illegible]	334	11	7	**51.4**
140	353	[illegible]	254	4	19	**50.5**
142	6	[illegible]	132	3	2	**50.0**
144	9	[illegible]	274	9	13	**49.5**
146	113	[illegible]	2474	36	17	**48.6**
148	18	[illegible]	903	11	7	**48.3**
150	851	[illegible]	791	26	25	**46.1**
152	1	[illegible]	4663	129	87	**45.2**
154	261	[illegible]	349	5	6	**44.5**
156	92	[illegible]	530	17	37	**43.6**
158	474	[illegible]	405	5	7	**42.9**
160	579	[illegible]	604	13	12	**42.2**
162	302	[illegible]	146	3	16	**38.6**
164	631	[illegible]	1179	26	21	**37.3**
166	746	[illegible]	88	3	3	**35.7**
168	755	[illegible]	104	3	3	**31.1**
170	686	[illegible]	40	2	2	**25.0**
172	222	[illegible]	120	1	1	**20.0**
174	167	[illegible]	115	2	4	**18.8**
176	720	[illegible]	235	4	1	**7.1**
178	825	[illegible]	697	13	4	**3.4**
180	640	[illegible]	264	5	4	**2.3**
182	632	[illegible]	59	6	1	**0.0**
184	733	[illegible]	75	9	1	**0.0**

Figure 6.16: Class-wise detection results showing every second sign code class sorted according to sign detector average precision (AP) in descending order. For each sign code class we report in the columns the corresponding sign code (MZL) as used in Borger's sign lists [29], Unicode character (Sign), number of occurrences in Train TL, Train BB (with bounding boxes), and Test set, and finally the AP of the sign detector on the Test set.

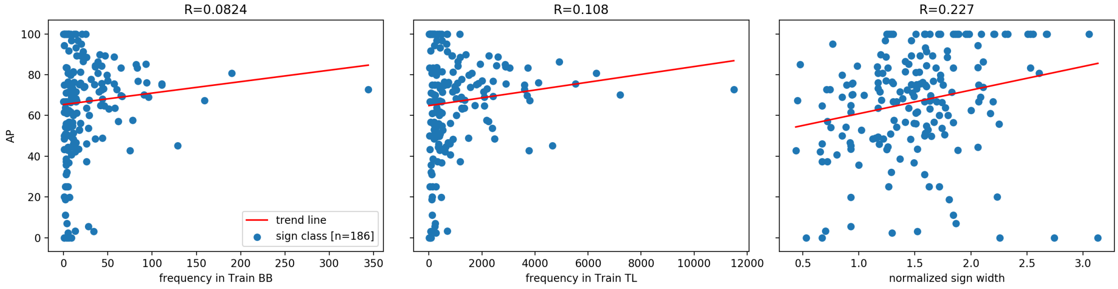


Figure 6.17: Three scatter plots studying potential correlation between performance of sign detector on individual sign code classes and their frequency in the Train TL, their frequency in Train BB, and their normalized sign width (from left to right respectively). The red curve is based on a linear regression indicating the data trend. The number on top of each plot is the Pearson correlation coefficient.

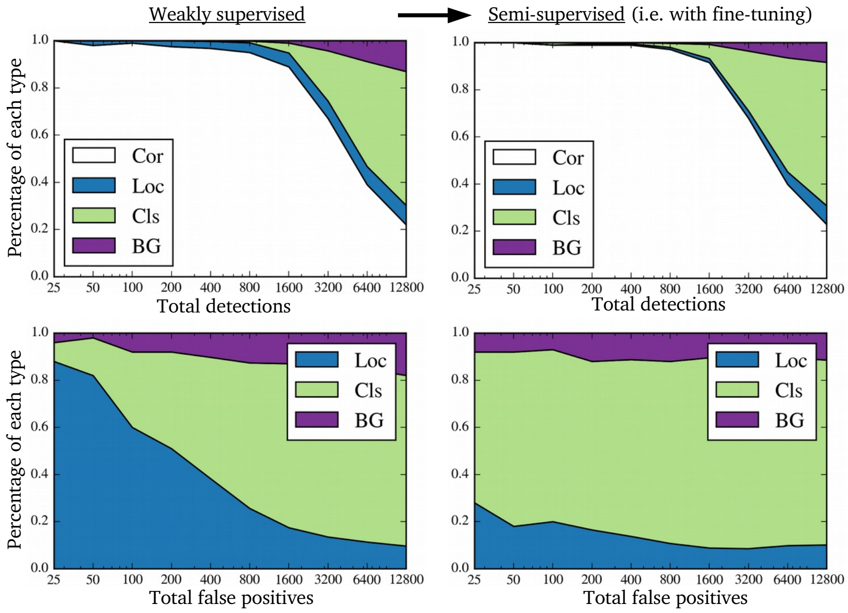


Figure 6.18: False positive error analysis of sign detector on test set. In each plot the detections are ranked according to their confidence along the x-axis (starting with highest confidence from the left). Along the y-axis the distribution of detection types (Cor: true positive, Loc: localization error, Cls: class confusion, BG: background confusion) is visualized. Top row: All detections including true positives (Cor). Bottom row: Only FP detections. Left column: Weakly supervised training causes a high percentage of localization errors w.r.t. all FPs. Right column: Fine-tuning mitigates the impact of badly localized detections significantly.

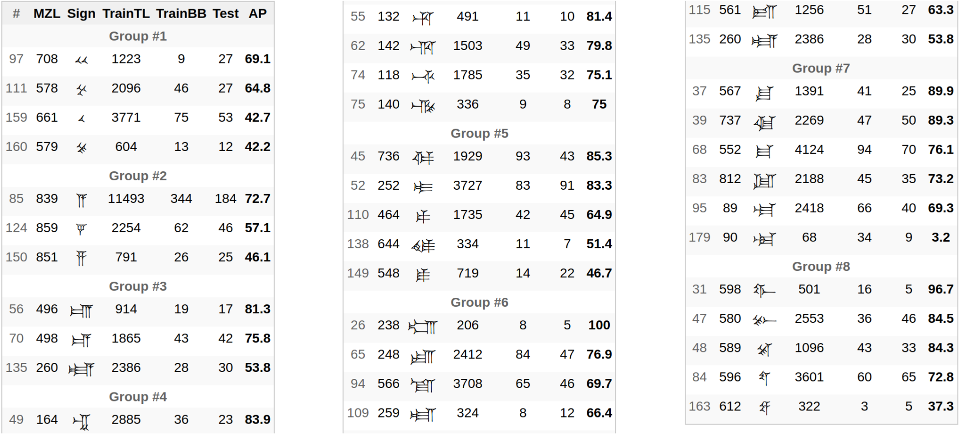
6.6.2 False Positive Analysis

False Positive (FP) categorization provides helpful insights into the performance of an object detector. We follow the analysis as introduced in [98] in order to classify false positives into three categories: Localization error (Loc), Background confusion (BG) and similar Class confusion (Cls). Fig. 6.18 shows the false positive analysis for a sign detector on the test set evaluated at two different stages of sign detector training.

When training the sign detector with weakly supervised data, badly localized bounding boxes can cause an increase of localization error as shown in the false positive category distribution in the left column of Fig. 6.18. The right column shows the false positive category distribution of the same sign detector after fine-tuning on manual annotations. Fine-tuning results in a significant decrease of localization error which is visible in the difference between the left to right column. The fine-tuned detector produces a large number of True Positives (TP) detections and reduces localization and background errors. The similar class confusions make up the majority of detection errors due to the fine-grained differences between cuneiform signs.

6.6.3 Similar Class Confusion Analysis

Similar class confusion errors are the most frequent error (FP category) produced by the cuneiform sign detector at the end of iterative training as shown in Sect. 6.6.2. When detecting cuneiform signs, there are different sets of cuneiform sign code classes (”false



#	MZL	Sign	TrainTL	TrainBB	Test	AP
Group #1						
97	708	[illegible]	1223	9	27	**69.1**
111	578	[illegible]	2096	46	27	**64.8**
159	661	[illegible]	3771	75	53	**42.7**
160	579	[illegible]	604	13	12	**42.2**
Group #2						
85	839	[illegible]	11493	344	184	**72.7**
124	859	[illegible]	2254	62	46	**57.1**
150	851	[illegible]	791	26	25	**46.1**
Group #3						
56	496	[illegible]	914	19	17	**81.3**
70	498	[illegible]	1865	43	42	**75.8**
135	260	[illegible]	2386	28	30	**53.8**
Group #4						
49	164	[illegible]	2885	36	23	**83.9**
55	132	[illegible]	491	11	10	**81.4**
62	142	[illegible]	1503	49	33	**79.8**
74	118	[illegible]	1785	35	32	**75.1**
75	140	[illegible]	336	9	8	**75**
Group #5						
45	736	[illegible]	1929	93	43	**85.3**
52	252	[illegible]	3727	83	91	**83.3**
110	464	[illegible]	1735	42	45	**64.9**
138	644	[illegible]	334	11	7	**51.4**
149	548	[illegible]	719	14	22	**46.7**
Group #6						
26	238	[illegible]	206	8	5	**100**
65	248	[illegible]	2412	84	47	**76.9**
94	566	[illegible]	3708	65	46	**69.7**
109	259	[illegible]	324	8	12	**66.4**
115	561	[illegible]	1256	51	27	**63.3**
135	260	[illegible]	2386	28	30	**53.8**
Group #7						
37	567	[illegible]	1391	41	25	**89.9**
39	737	[illegible]	2269	47	50	**89.3**
68	552	[illegible]	4124	94	70	**76.1**
83	812	[illegible]	2188	45	35	**73.2**
95	89	[illegible]	2418	66	40	**69.3**
179	90	[illegible]	68	34	9	**3.2**
Group #8						
31	598	[illegible]	501	16	5	**96.7**
47	580	[illegible]	2553	36	46	**84.5**
48	589	[illegible]	1096	43	33	**84.3**
84	596	[illegible]	3601	60	65	**72.8**
163	612	[illegible]	322	3	5	**37.3**

Figure 6.19: Class-wise detection results showing sign code classes organized in eight groups of very similar classes sorted according to sign detector average precision (AP) in descending order. For each sign code class we report in the columns the corresponding sign code (MZL) as used in Borger's sign lists [29], Unicode character (Sign), number of occurrences in Train TL, Train BB (with bounding boxes), and Test set, and finally the AP of the sign detector on the Test set.

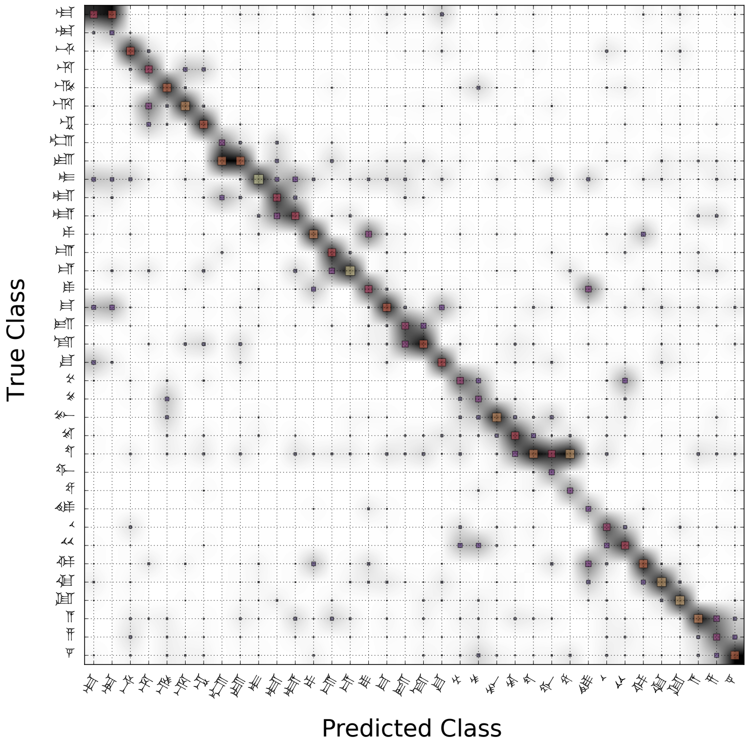


Figure 6.20: Confusion matrix of 36 sign code classes from eight groups of often confused sign code classes. Along the x-axis we visualize the predicted classes and along the y-axis the ground truth classes. The size of the squares and their shadow indicate the number of detections that have a specific predicted class and ground truth class. The detections on the diagonal are true positives, while all detections off the diagonal are false positives.

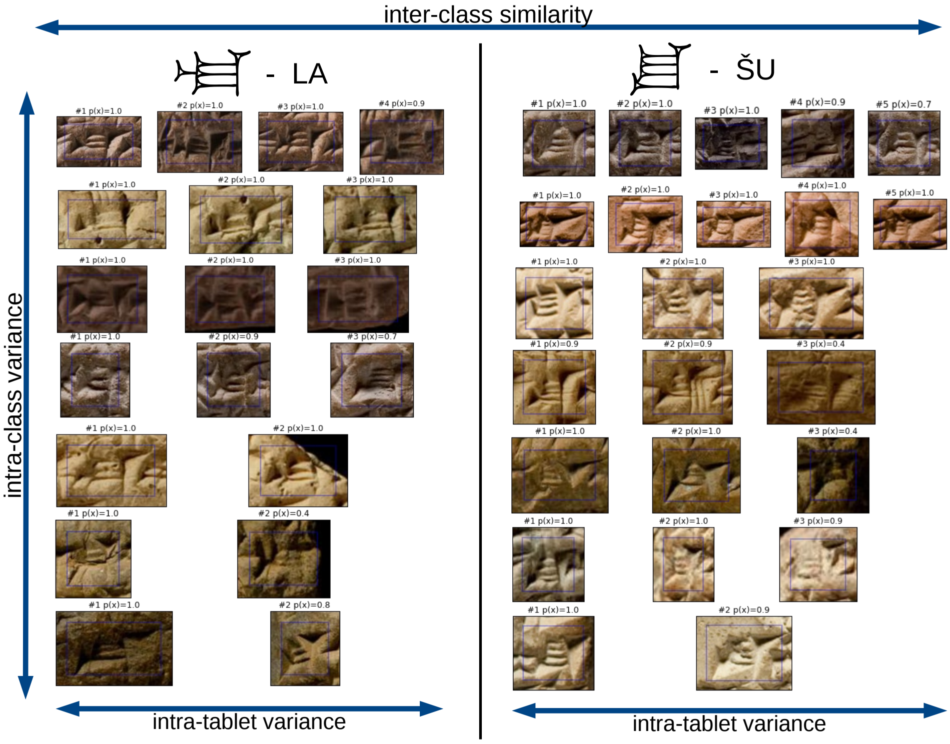


Figure 6.21: True positive detections of two different cuneiform sign code classes arranged in two columns. For each column each row of detections is obtained from a single tablet image. The blue box depicts the detected bounding box. Above each detection its confidence is reported.

friends") that are easily confused by the sign detector. Using the t-SNE visualization technique [143], we manually identify eight groups of false friends by identifying clusters of signs in the feature space defined by the trained sign detector. According to our Assyriology experts, the identified groups indeed match signs often confused by Assyriology students. Fig. 6.19 reports the sign detector performance on individual sign code classes of these eight groups evaluated on the Test set in terms of class-wise Average Precision (AP). We find that despite the high inter-class similarity inside the groups, the sign detector is robust to many possible class confusions and its performance only suffers in a few cases.

To better understand which classes are confused exactly, we additionally plot a confusion matrix in Fig. 6.20 for the sign code classes of these eight groups of very similar sign code classes. There are a few very strong class confusions. Fortunately, the diagonal is very prominent, indicating many true positive detections.

6.6.4 Individual Sign Detections

In Fig. 6.21 we plot individual true positive sign detections for two very similar-looking cuneiform sign code classes. Besides the high inter-class similarity between the different columns in Fig. 6.21, there is also a high intra-class variance visible across the different sign detections of each column. Even signs of the same class and from the same tablet (same scribe) can vary in appearance considerably. Multiple factors contribute to intra-class variance and are visible across the sign detections of each column: Changing writing styles, overlapping signs, differences in the the clay material, illumination, orientation of the tablet to the camera, and the state of conservation of each sign. The sign detector trained by our weakly supervised approach is able to deal with this challenging setting and demonstrates good results on a diverse set of tablet images.

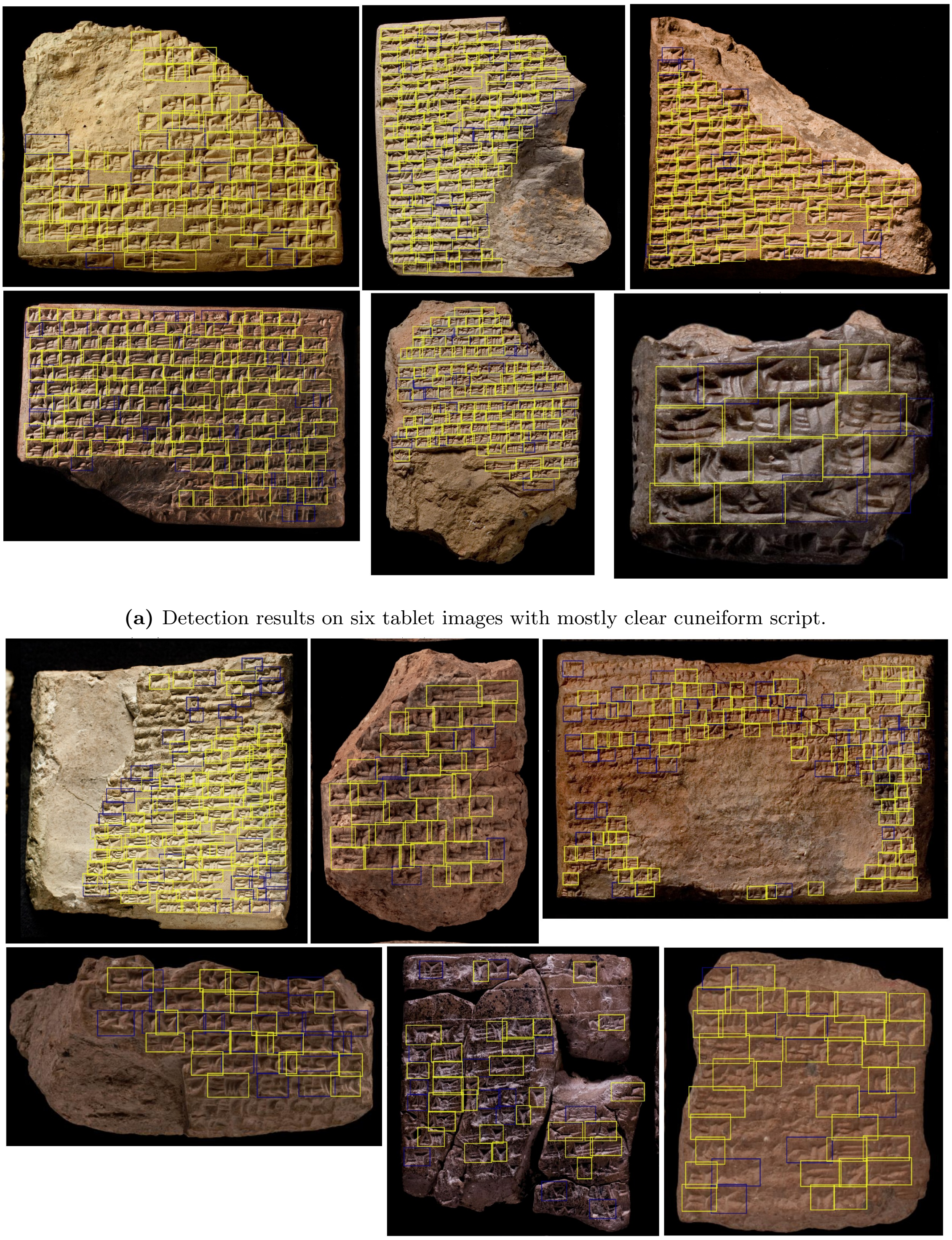
6.6.5 Detection Results on Tablet Images

Fig. 6.22a and Fig. 6.22b visualize qualitative detection results on full tablet images that are not part of the train set after iterative training (weakly supervised and semi-supervised with 745 manual sign annotations). We color-code TP and FP detections using the ground truth sign annotations (bounding boxes). In Fig. 6.22a relatively few errors are visible. Errors tend to occur more often on the border of the tablet which is mostly due to curvature of the tablet as well as damaged or broken signs. The examples in Fig. 6.22b illustrate levels of damage that are common in the SAAo dataset. Despite of this, the sign detector provides robust detections and many errors correlate with badly damaged signs.

Fig. 6.23 visualize qualitative detection results on full tablet images from the test set over three iterations of weakly supervised and two iterations of semi-supervised training (see Sect. 6.5.5 for quantitative results of this experiment). The performance improvement during iterative training is clearly visible. The most prominent changes occur in earlier iterations, while later changes are more subtle (e.g. improvements in localization and rare classes).

6.6.6 Backbone Architecture Comparison

A deep neural network-based sign detector is composed of a backbone and a detection head network. The backbone network is the core of the learned representation, which is usually pre-trained on a classification task. In Fig. 6.24 we compare four different backbone architectures on the task of sign code classification (with 186 classes) on our test set. The classification performance of a backbone architecture provides a decent indicator for the overall detection performance [103]. All configuration have been trained on the train set E using a SGD optimizer with momentum. Learning rate and training schedule have been manually tuned for best performance. The network performance is reported as percentage of correctly classified signs (classification accuracy). The number of trainable parameters of the backbone networks is also visualized, since this is an important factor for inference speed, hardware requirements and model regularization. We compare two MobileNet-v2 [187] based architectures and the well-known AlexNet [121] and ResNet-18 [91] architectures. The *cuneiform* version of MobileNet-v2 is our adapted version (see Sect. 6.4.4) used in all our experiments whereas the *original* version implements the exact architecture of [187] with its width multiplier set to 0.75. The *cuneiform* version of MobileNet-v2 is by far the smallest backbone architecture with about 700K parameters as plotted in Fig. 6.24. Nevertheless, it shows competitive performance when compared to the other models. Using the larger *original* version of MobileNet-v2 improves performance. The AlexNet architecture performs the worst, while the ResNet-18 model shows comparable performance to the *cuneiform* version, but requires over ten times the number of parameters.



(a) Detection results on six tablet images with mostly clear cuneiform script.

(b) Detection results on six tablet images with damaged cuneiform script.

Figure 6.22: Detection results on twelve tablet images from the test set with (a) clear and (b) damaged cuneiform script. We only show detections with a confidence score higher than 0.5. Yellow bounding boxes indicate true positive detections and blue bounding boxes false positive detections. Best viewed in electronic form. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

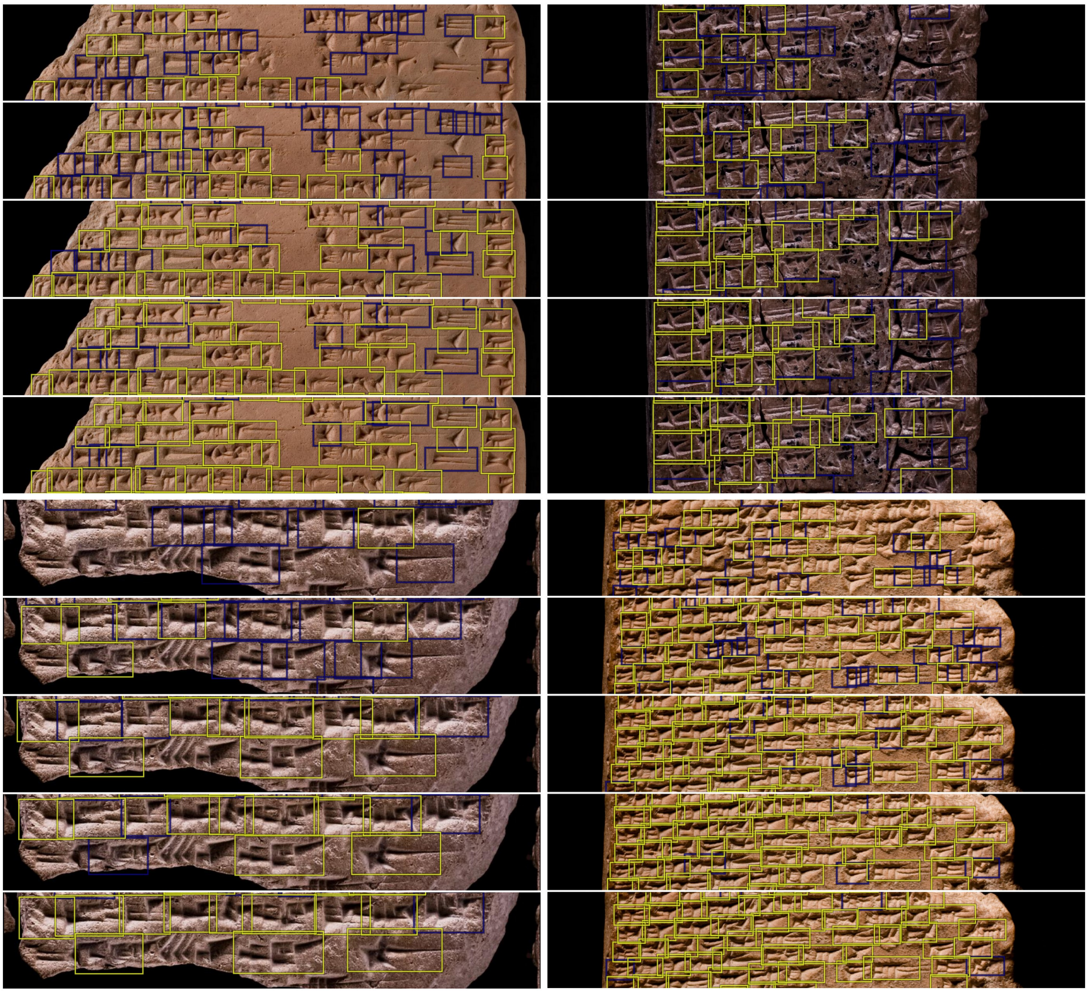


Figure 6.23: Image crops from four clay tablets illustrating the change of detection results during training. For each crop we show five consecutive iterations of iterative training (starting with first iteration from top). We only show detections with a confidence score higher than 0.2. Yellow bounding boxes indicate true positive detections and blue bounding boxes false positive detections. Best viewed in electronic form. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

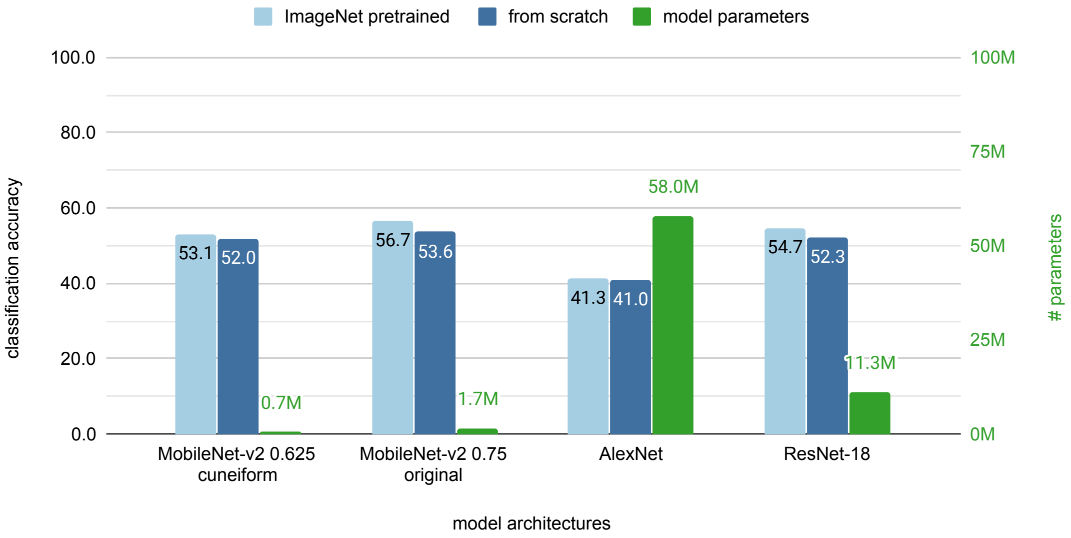


Figure 6.24: Evaluation of different backbone architectures and their initialization on the task of cuneiform sign code classification. Four backbone architectures trained from scratch are compared to ones trained from ImageNet pre-trained weights. The performance is measured on the test set as classification accuracy (%).

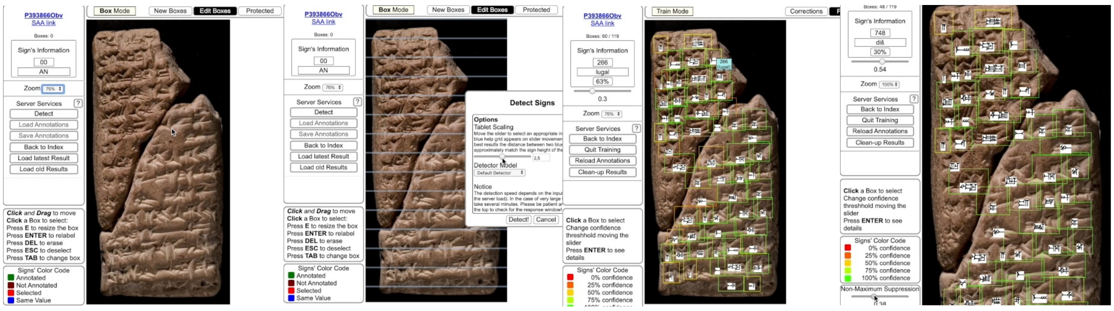
6.6.7 ImageNet Dataset Pre-training

Pre-training a backbone network on the large ImageNet dataset [54] is a common method to improve the performance on different tasks with little training data. In Fig. 6.24 we investigate how well a backbone network performs on the task of cuneiform sign code classification when its training is initialized with ImageNet pre-trained weights compared to when it is trained from scratch. We use the same backbone architectures and training configuration as described in previous experiment. Initializing the network training with ImageNet pre-trained weights results for each architecture in a 1%–5% performance improvement when compared to training from scratch. This is in line with current research on transfer learning [117] that finds only modest performance improvements in the case of fine-grained target domain. In addition the domain of ImageNet objects is very different from the domain of cuneiform sign code classes and distinguishing them requires to learn two different feature representations with little overlap. Overall ImageNet pre-training offers a small performance improvement, however, it is clear that it cannot replace the sign annotations required for detector training.

6.6.8 Web Application of Cuneiform Sign Detection

In Chapter 5 a custom web application is introduced that is designed to support the annotation and detection of cuneiform script. Sect. 5.3 describe how it can be used to facilitate the manual annotation process. This section explains how the web application integrates the cuneiform sign detector to support Assyriologist in their analysis of clay tablets.

Fig. 6.25 demonstrates the web application of the cuneiform sign detector in action and



Boxes: 0

Sign's Information: 00, AN

Zoom: 75%

Click and *Drag* to move
Click a Box to select:
Press **E** to resize the box
Press **ENTER** to relabel
Press **DEL** to erase
Press **ESC** to deselect
Press **TAB** to change box

Signs' Color Code
🟩 Annotated
🟥 Not Annotated
🟥 Selected
🟦 Same Value

Tablet Scaling: Move the slider to select an appropriate in blue help grid appears on slider movemen best results the distance between two blue approximately match the sign height of the 2,5

Notice: The detection speed depends on the input the server load). In the case of very large take several minutes. Please be patient ar the top to check for the response window!

Boxes: 60 / 119

Sign's Information: 266, lugal, 63%

0.3

Click a Box to select
Change confidence threshhold moving the slider
Press **ENTER** to see details

Signs' Color Code
🟥 0% confidence
🟧 25% confidence
🟨 50% confidence
🟩 75% confidence
🟩 100% confidence

Boxes: 48 / 119

Sign's Information: 748, diš, 30%

0.54

Zoom: 100%

Non-Maximum Suppression: 0.38

Figure 6.25: Usage of web application for cuneiform sign detection. From left to right: 1) A tablet image is uploaded and shown in the web application. 2) The sign detection window is opened, where the user adjusts the scaling of the tablet image. The blue grid lines help the user determine a scale that roughly matches the average sign height in the image. 3) After running the detection, raw sign detections are visualized with their bounding boxes and labels. 4) It is possible to adjust two thresholds to filter raw detections according to the detector confidence and their overlap. This figure contains image material shared by the British Museum under a (CC BY-NC-SA 4.0) licence.

illustrates how a sign detector could be made available to Assyriologists. The following steps are visualized in Fig. 6.25: A tablet image is opened in the web interface, and the detection function is called. Since the image resolution of tablet images varies a lot, the user manually determines the average height of a line for rescaling the tablet image to a standard sign height. A coarse approximation is sufficient, as the sign detector is searching across multiple scales by default. This step can be automated using line detection as implemented for the iterative training procedure. After executing the detection, the results are immediately available for analysis in the web interface. Raw detections are visualized as bounding boxes, whose color indicates the detection confidence. The sign code class of the detected signs can be displayed in the form of readings or Unicode symbols (by pressing [.] or [ctrl] + [.] respectively) or by hovering with a cursor over the signs. It is possible to adjust thresholds to filter raw detections in two ways: 1) Raw detections can be filtered according to their confidence values, only displaying detections with a minimum confidence. 2) Raw detections can be filtered by means of non-maximum suppression (NMS): For each pair of raw detections, compute the overlap of their bounding boxes and if two boxes have an overlap larger than the selected NMS-threshold, remove the detection with the lower detection confidence.

6.7 Discussion

In this chapter, we have investigated how to train a cuneiform sign detector with minimal supervision. Instead of requiring manual annotations, our approach generates sign annotations for sign detector training by leveraging existing transliterations that are part of the Assyriologist's workflow. In each iteration our iterative learning method alternates between training a sign detector and generating sign annotations by solving the alignment problem between tablet image and transliteration across hundreds of clay tablets in parallel. To improve the sign detector, our approach focuses on increasing the quality and quantity of generated sign annotations. To deal with the fine-grained nature of cuneiform sign detection, the strong imbalance of sign code class frequency and the difficulty of the transliteration alignment, we split the sign annotation generation into two distinct steps: sign placement and image-transliteration alignment. In this way we balance between learning from placed sign detections with sign code classes that are hard-to-predict or even unknown to the detector (exploration) and learning from aligned detections that are reliable (exploitation). The exploration part of our approach injects sign annotations with new sign code classes into the training of the sign detector by systematically placing signs from the transliteration in the gaps left after the alignment step. The exploitation part of our approach produces very reliable sign annotations that are consistent with the available bottom-up information (line and sign detections in tablet image) and top-down information (transliteration and line-geometry prior).

The performance analysis of the trained sign detector uncovers some of its limitations. The sign detector struggles with some rare sign code classes (see Sect. 6.6.1). Although the sign placement step explores new sign code classes and refreshes the memory of previously learned classes during iterative training, it cannot fully replace the collection of additional training data for rare sign code classes. Fortunately, only a few manual annotations are necessary to boost the sign detection performance thanks to weakly supervised learning, as shown in our third experiment. Additionally, to further reduce the burden of manual annotation, the purely weakly supervised sign detector can be leveraged to provide its sign detections to the annotator in a human-in-the-loop manner, which we describe in the case of our web application in Sect. 5.3. Instead of annotating tablet images from scratch, the annotation process is simplified to correcting the erroneous sign detections, i.e. changing labels and fixing bounding boxes. Even though the detector handles significant variations in writing style, detection performance suffers from large variations in tablet (text) orientation, camera angle and image scale (image transformations). In this chapter we make the assumption that most available tablet images are properly orientated, which allows us to focus on sign detection performance independent of large image transformations. To increase the invariance to image transformations like rotation and scaling, we can use additional data augmentation during detector training and additional scale levels in the feature pyramid network of the sign detector. Of course, this additional invariance comes at a price of larger model size (more parameters) and increased training time.

We think that weakly supervised machine learning is an important research area that can profit from the study of such a challenging fine-grained detection problem. Research into object detection is often confronted with problems like object occlusion, object overlap, object viewpoint change, and object class confusion. Cuneiform script provides a unique combination of these problems that are usually only available in separate object detection datasets: tightly packed signs, fine-grained class differences, signs that are made up of 3D wedges that are hard to recognize due to viewpoint or damage, language-related dependencies between signs, and the geometric regularity of written script. By providing access to a new dataset with thousands of clay tablet images, transliterations and a bounding box annotated subset for evaluation, we hope to inspire researchers to explore the limits of fine-grained object detection and weakly supervised learning in the context of noise and ambiguity of cuneiform script.

While solving a challenging computer vision problem, our learning approach also provides a powerful tool for Assyriology. With a cuneiform sign detector, an important step of reading cuneiform script is significantly simplified. The resulting sign-by-sign representation together with bounding boxes provides a detailed analysis of cuneiform script in a clay tablet that is easy to interpret by a user. Our sign detector can be deployed in many situations, since it only requires 2D images as input that any smartphone camera is ready to produce.

However, in some cases detecting cuneiform signs correctly is impossible without additional context information. Assyriologists naturally take additional context knowledge about a clay tablet into account in order to arrive at their analysis. They look at the clay tablet from multiple viewpoints, remember similar texts, have a syntactic and semantic understanding, integrate knowledge from language, history and culture. The proposed cuneiform sign detector is only taking into account the direct neighbourhood of a cuneiform sign in the form of an image crop, when inferring the sign code class and its bounding box. While the context understanding of an Assyriologist is still beyond the-state-of-the-art of machine learning, we would like to take steps in this direction in future research. For example, a sign detector might fuse detections from 2D images from different angles of a clay tablet in order to deal with difficult cases. Similarly, the integration of a language model can also help with ambiguous detections, which we will investigate in Chapter 7.

Our weakly supervised approach to learn a sign detector is not limited to cuneiform script of the Neo-Assyrian era. It is applicable to other cuneiform scripts like Old-Babylonian cuneiform as shown in Sect. 6.5.8. Our approach has been designed to deal with large discrepancies between tablet images and transliterations which is a common feature of ancient scripts and thus might provide a template for weakly supervised sign detection in other transliterated scripts.

Digital library projects like CDLI [49] and ORACC [161] grant researches online access to thousands of clay tablets and their transliterations, and offer new opportunities for large-scale digital analysis of cuneiform script. For this purpose, a sign detector provides a fast preliminary analysis of unstudied clay tablets. Additionally, the learned feature representation of a sign detector can serve as a similarity metric for the shape of cuneiform signs. Eventually, this will facilitate the visual search across thousands of tablet images and thus the reconstruction of broken tablets by speeding up the search for matching pieces.

CHAPTER 7

Linguistic Refinement of Cuneiform Sign Detection

So far we have approached cuneiform script as a composition of mostly independent characters along a line. While we considered a geometric dependency between characters (such as overlap and orientation), we did not take into account the linguistic dependency between adjacent characters. In this chapter, we explicitly model the dependency between characters that encode text of a specific natural language and use this additional source of information to improve the detection results.

7.1 Introduction

When a spellchecker (e.g. on a smartphone) corrects mistakes and sometimes auto-completes half-finished sentences, its corrections are guided by a statistical model of the natural language conditioned on the already written text. Such a language model enables a spellchecker to provide valuable corrections (besides occasional exceptions), as the syntax and semantics of a language strictly dictate what constitutes a valid sentence. The language model effectively reconstructs the original or intended sentence from the noisy and incomplete version of the sentence.

Our goal is to obtain a form of spellchecker for cuneiform script to further improve the detection results of the cuneiform sign detector described in Chapter 6). This method for linguistic refinement needs to integrate a language model for cuneiform script in the sign detector pipeline. Since the languages used with cuneiform script, mostly Akkadian and Sumerian, are long out of use, we cannot rely on off-the-shelf language models that exist for many mainstream languages. Instead we have to learn a language model from scratch. The logo-syllabic nature of cuneiform script and its high level of polyvalence (the same combination of signs can have multiple meanings and the same meaning can be expressed in multiple combinations of signs) make learning a language model challenging, in particular as the text corpus for training is not very large. Besides learning the language model, the central question is how the language model can be turned into a spellchecker for the existing sign detector.

We present a method for linguistic refinement of cuneiform sign detections that effectively combines the top-down information from the language model and bottom-up information from the detector. The method is applied as postprocessing step to the sign detections and consists of a two-stage approach with a *line text prediction* step followed by a *line text correction* step.

First, the line text prediction step outputs a sequence of cuneiform sign code classes as prediction for each line of cuneiform text in the tablet image. It makes use of line detections to identify the sign detections in the image plane that correspond to a particular line. For a whole tablet image, the line text prediction step amounts to a sign-level transliteration based on the bottom-up information of line and sign detections.

Second, the text correction step takes care of potential errors in the line text prediction. A RNN-based text correction model is trained to correct errors in the sign detections. At the heart of the model is a *language model* that exploits language statistics to identify and fix errors (cf. Sect. 2.5). We frame the training of the text correction model as a sequence-to-sequence learning problem: The model needs to learn how to map each line of the text prediction (based on potentially erroneous sign detections) to a corrected sequence of cuneiform sign code classes. The text correction model uses a general encoder-decoder architecture for mapping one sequence to another. The encoder and decoder are modeled as two recurrent neural networks. The encoder creates an internal representation of the input sequence (ie. line text prediction). The decoder generates the output sequence (ie. corrected line text prediction) conditioned on the internal representation of the encoder and thus takes the role of the language model. This way the text correction model naturally integrates a language model that captures the language-related dependencies between individual characters.

To train the text correction model, we make use of the transliterations collected for the cuneiform sign detection dataset in Chapter 5 which provide us with a large number of text lines in Neo-Assyrian cuneiform. Inspired by the language modelling task in Natural Language Processing (NLP), the text correction model is trained in a self-supervised fashion by generating artificially corrupted lines of text as training data. The error level in the training data is adapted to facilitate learning while approximating the error level in real data. The robustness of the text correction model is increased by the integration of word embeddings that we specifically pre-train for cuneiform script.

Our approach demonstrates the first steps towards the automatic generation of a transliteration from the image of a cuneiform clay tablet. The proposed linguistic refinement method maps sign detections to individual line positions and thus produces a sign-by-sign representation of the cuneiform clay tablet in terms of sign code classes which we also refer to as *cuneified transliteration.* While the cuneified transliteration, unlike the regular transliteration, only records the sign code class of each sign and not its reading (cf. Chapter 4 for differentiation), it is an essential step in the creation of a full transliteration. Furthermore, it enables a weakly supervised evaluation of the sign detector across most of the dataset presented in Chapter 5. While the evaluation in Chapter 6 requires manual bounding box annotations, the weakly supervised evaluation relies on cuneified transliterations that can be directly derived from regular transliterations available for most of the dataset. In the following, transliteration again means cuneified transliteration in terms of sign code classes, if not otherwise noted.

Our analysis of the potential of learning a language model for cuneiform script provides a proof-of-concept, but also identifies additional issues that need to be addressed in the future. It is possible to learn language representations for cuneiform script, that seem to group signs according to syntactic and semantic functions. The text correction model improves performance for frequent text patterns (n-grams), but fails for the long tail of infrequent text patterns. Our method to integrate additional bottom-up information improves the performance, if the noise in the data is not too high. Our findings provide a better understanding of the challenges of linguistic refinement and transliteration generation for cuneiform script, and motivate new research at the intersection of computer vision and natural language processing.

7.2 Related Work

A special case of Text Recognition Cuneiform text detection or transliteration generation is a special case of text recognition. There are several approaches for text recognition that combine bottom-up visual information with top-down language information [152, 232]. The language information is employed to constrain the text recognition problem.

The literature distinguishes between text recognition with constrained and unconstrained vocabulary. In the case of constrained vocabulary, text recognition is constrained by a mostly very small lexicon containing the set of all possible words. In case of an unconstrained vocabulary, text recognition is regularized by means of a language model. Text recognition is improved by decoding with a language model, i.e. parsing the bottom-up information based on top-down information. Since cuneiform script has a large vocabulary, we focus on unconstrained text recognition methods that include language models.

Language models usually require a large text corpus for training. In the case of cuneiform script, the readily available text corpus is limited to a few online sources and the cuneiform writing system is often ambiguous as discussed in Sect. 4.2. Thus learning a language model for cuneiform script has to overcome the problem of data sparsity as reported in [162] for the machine translation of Sumerian languages.

Another important aspect of a language model is the underlying tokenization of the text, i.e. how to segment the text into parts across which to compute the language statistics. The two most common approaches are character-level and word-level tokenization as described in Sect. 2.5. In the cuneiform setting character-level tokens are individual cuneiform signs while word-level tokens consist of one or multiple cuneiform signs. As the sign detector in Chapter 6 operates on individual signs, we focus here on sign-level (i.e. character-level) language models. This also reduces the problem of poly-valence of cuneiform script: It is often unclear how to map from word-level to sign-level tokens, since a word in Neo-Assyrian cuneiform can be written using different combinations of cuneiform signs. The literature distinguishes two major types of language models: n-gram language models or neural network language models. While the n-gram language model is usually integrated as a static component of text recognition, the neural network language model is part of end-to-end trainable pipeline.

Integration of n-gram Language Models Early on, the field of Handwritten Text Recognition (HTR) leverages top-down language information for postprocessing [206], as bottom-up shape information is often insufficient for the disambiguation of handwritten text. Similarly, the field of Optical Character Recognition (OCR) employs post-OCR correction to integrate language information [210] by means of a probabilistic model. The n-gram probabilities are estimated from a natural language corpus. Most postprocessing approaches are based on Hidden Markov Models (HMM) that adjust word confidence weights to identify the most likely text sequence. [32].

In the field of HTR, Graves *et al.* combine convolutional networks, a LSTM based RNN, and a Connectionist Temporal Classification (CTC) loss to obtain an end-to-end trainable HTR network. They integrate the language model in the decoding process by factoring in the probability of each potential sequence according to a bigram language model. The work by Graves *et al.* [86] has been very influential, providing a template for many recent publications [35, 221, 230].

Similarly, n-gram language models have been integrated in the field of text recognition and scene text recognition (text spotting). [151, 152] propose a CNN pipeline for scene text recognition with beam search informed by a language model prior. [26] introduce a postprocessing with two-stage language models for large-scale scene text recognition. [108] learn a text recognition network that outputs character and n-gram detections. The detections provide the potentials of a CRF that forms a structured loss for end-to-end network training. The language information is integrated by means of the n-gram detections as well as a small language model integrated as weighting factor in the CRF.

Integration of Neural Network Language Models as part of End-to-end Trainable Pipeline Sequence-To-Sequence (seq2seq) learning deals with the problem of mapping an input to an output sequence where the length of both sequences may be different. [43, 201] introduce an encoder-decoder architecture to tackle this problem by decoupling the decoding (sequence generation) from the encoding (feature extraction). The encoder creates an internal representation of the input that the decoder uses for generating the output. To improve the access to the internal representation of the encoder, various attention mechanisms [16, 142] have been proposed. The decoder network is essentially a language model that is additionally conditioned on the input sequence. Therefore, the language model is naturally integrated in the seq2seq architecture and is trained jointly with the seq2seq model in an end-to-end fashion.

Seq2seq provides a flexible architecture for the integration of language models in many computer vision problems. In the case of text recognition, a CNN is included as a feature extractor to map the 2D image to a 1D sequence of features that serve as input to the encoder network. HTR with a seq2seq architecture and attention has shown strong performance recently [45, 148] that surpasses the performance of approaches with static n-gram language model integration. A similar development occurred in the field of scene text recognition, where the seq2seq architecture with attention has been widely adopted [128, 224]. Seq2seq learning expends the capabilities of OCR as shown for the application of image-to-markup generation [55]. In the case of post-OCR correction, [6, 58, 119, 190] investigate text correction for a standard OCR pipeline based on the seq2seq architecture that integrates powerful language models. These approaches differ from ours in that they consider only the high-contrast OCR domain, rely on pre-trained language models, and do not have to deal with the peculiarities of cuneiform script.

7.3 Approach

Our approach for linguistic refinement aims to improve the output of the cuneiform sign detector based on the top-down information provided by a language model. Fig. 7.1

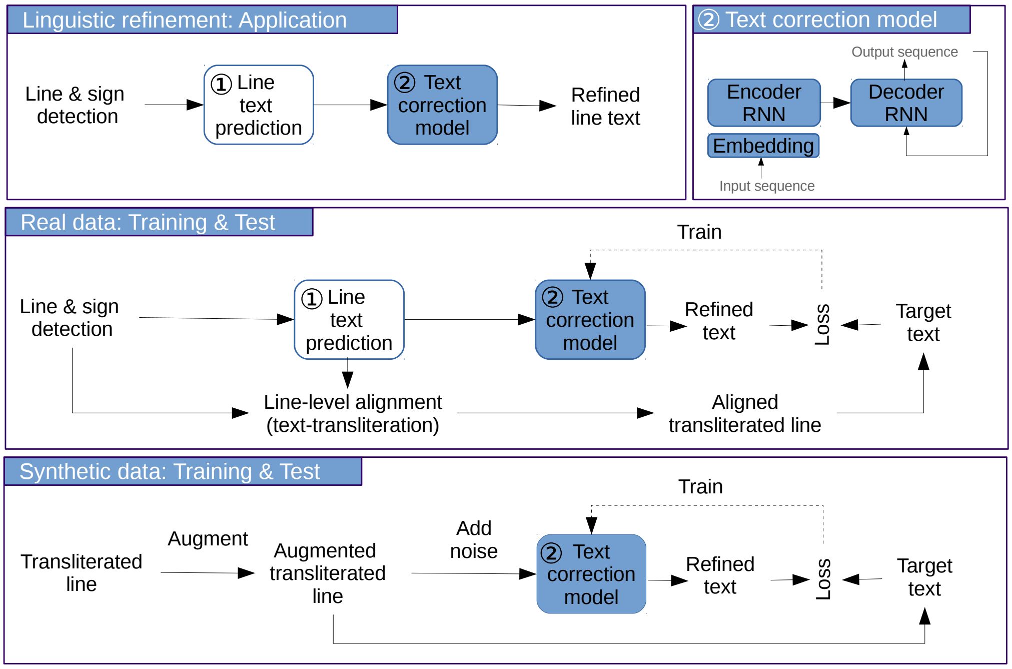


Figure 7.1: The approach for linguistic refinement of the sign detections consists of two stages: 1) line text prediction and 2) line text correction. While line text prediction maps the 2D results of sign and line detection to a 1D text sequence, the text correction model updates the line text prediction based on top-down language understanding that is acquired by training on a large text corpus. We consider two data sources for training and testing: a) real data based on actual line and sign detections or b) synthetic data generated in a self-supervised fashion. The text correction model is implemented as a seq2seq RNN with an encoder-decoder architecture and pre-trained word embeddings for cuneiform script. The text correction model is trained by back-propagating the loss between refined and target text during each iteration of training.

visualizes our approach which provides linguistic postprocessing of the sign detections in two stages:

1. Line text prediction: Combine line and sign detections in order to predict the text sequence of a line.
2. Line text correction: Given a line text prediction, correct the errors in the line by applying a correction model that has been pretrained on a large corpus of transliterations.

While the line text prediction already provides a basic transcription of the line using the sign detector from the previous chapter, the line text correction step additionally leverages the language-related dependencies that can be mined from transliterations only.

7.3.1 Line Text Prediction

The line text prediction combines sign and line detections (from the detectors trained in Chapter 6) in a sequential procedure to output the text sequence of a line. For each line in a tablet image, line text prediction essentially maps the 2D bounding box detections

Table 7.1: The three types of text prediction errors, their corresponding text correction steps which consists of localization and error fixing, and the associated sign detection errors.

text prediction error	text correction	associated sign detection error
inserted sign	localize sign and delete sign	FP: background confusion
substituted sign	localize sign and replace sign	FP: class confusion
deleted sign	localize gap and insert sign	false negative

to a 1D text sequence. As the assignment of sign detections to individual lines is initially unknown, we rely on the line detection to provide a line polygon that can be used to identify the sign detections of a particular line. Sign detections are associated with a detected line, if the distance of their center to the line is smaller than a specific threshold. As there are usually many false positive sign detections, we remove all sign detections with low detector confidence (confidence threshold) and apply Non-Maximum-Suppression (NMS) across all remaining sign detections. Finally, we sort the remaining sign detections according to their horizontal position to produce a list of sign code class labels that forms the predicted text sequence of the line.

7.3.2 Line Text Correction

The line text correction stage of our linguistic refinement approach aims to improve the text line prediction by incorporating language-related dependencies that allow to localize and fix potential text errors. For the design of the text correction model, we consider three general error types that occur in a predicted text sequence as shown in Table 7.1. The text error correction requires to 1) locate the error in the text sequence and then 2) fix the error by either inclusion, deletion, or substitution of signs. As the line text prediction is based on sign detection, text errors are often caused by specific sign detection errors which we also list in Table 7.1. Since class confusion errors were the most frequent error source in the proposed sign detection approach (cf. Sect. 6.6.2), we similarly expect the substitution errors to be the most frequent error for line text prediction.

Our proposed text correction model combines the bottom-up information from the line and sign detections with the top-down information from the language in order to localize text errors and fix them. The text correction model is implemented as a sequence-to-sequence (seq2seq) recurrent neural network that takes line text predictions as input and outputs the refined versions as visualized in Fig. 7.1. The text correction model is trained on training samples, each consisting of a tuple of noisy input text and ground truth target text. Only by training on a large text corpus, the text correction model learns a good language model that is able to localize and fix the type of errors present in the line text prediction.

Real and Synthetic Data

For the training and testing of the text correction model, we make use of two data sources: Real data based on actual line and sign detections (*real*) and synthetic data generated from transliterations only (*synthetic*). The training on real data resembles supervised training with a fixed number of training samples, whereas the training on synthetic data resembles self-supervised training with training samples generated in large numbers by augmenting the existing transliterations and injecting synthetic noise. In Fig. 7.1 we visualize the training and testing of the correction model with real and synthetic data. Given the input and target texts, the training procedures for real or synthetic data are the same: The text correction model is trained using stochastic gradient descent with backpropagation and the standard cross-entropy loss. The difference between training and testing on real and synthetic data is due to the preparation of the data samples (tuples of input and target text).

Real data samples consist of a line text prediction (input) and an aligned transliteration line (target) as shown in Fig. 7.1. To obtain the target text, we employ the line-level alignment from (reference previous chapter). The input and target texts differ due to errors introduced during line and sign detection and the consecutive line text prediction. The difference between the input and target texts of real data samples resembles the typical errors produced by cuneiform sign detection and line text prediction. Training with real noise informs the correction model how to localize and fix errors. While there are additional error sources like the line-level alignment, such errors are either relatively rare or the affected data samples can be easily identified and discarded during training. Synthetic data samples are generated from the same transliterated line as shown in Fig. 7.1. First, the transliterated line is augmented by random edit operations and the result used as target text. Then, to obtain the input text, the augmented line is combined with synthetic noise. We use random inserts, deletes or substitutes as synthetic noise. While the augmentation effectively increases the amount of training data and thus prevents overfitting, the injection of synthetic noise controls what type and amount of noise the correction model is trained to deal with.

Since real training data represents the actual use case, i.e. applying linguistic refinement to the sign detections, the correction model can learn to deal with the specific error patterns found in cuneiform sign detections. Disadvantages of real training data are the limited amount of available training data, the unknown error between input and target (with multiple error sources like line and sign detection and line-level alignment), and the large variance of the error between input and target text across training samples.

In contrast, synthetic training data offers a larger training dataset which can be expanded by data augmentation. Further, the synthetic noise between input and target can be controlled to adapt the learning process, e.g. curriculum learning or pre-training of the text correction model in self-supervised fashion before fine-tuning on real data. The drawbacks are that synthetic training data only approximates the real data distribution and lacks some of the sign-detection specific errors, thus inhibiting the learning of the correction model with regard to the application case.

Text Correction Model

The architecture of the text correction model follows the seq2seq architecture proposed in [43, 201] and consists of an encoder and decoder network as visualized in Fig. 7.1. By first parsing the input through a encoder the model summarizes the input in a state vector that is used in the language decoder to predict the refined output. The access to the encoder's memory of the input is further improved by the addition of an attention layer. It allows the decoder to focus its attention on individual signs of the input sequence (in the form of a weighted sum) when predicting the output signs. Encoder and decoder are each implemented as a two-layer GRU [43, 96] with 128 hidden units together with a dot-product attention layer [16, 142]. The decoder in the seq2seq architecture serves as

```
# 1) Line text prediction and detection confidence
pred: [110, 266, 661, 380, 490]
conf: [0.95, 0.99, 0.23, 0.81, 0.19]
# 2) Input sequence with LC tokens and corresponding GT target
> [110, 266, <slc>, 661, 380, <slc>, 490, <eos>]
= [110, 266, 113, 380, 260, <eos>]
```

Figure 7.2: Example of LC token usage. Signs of the line text prediction with confidence < 0.3 are tagged with a `<slc>`. The correction model learns to remove the LC tokens and fix the errors by providing the ground truth target during training.

a language model. Thus training the seq2seq text correction model includes the training of the language model. After learning the language-related dependencies during training, the decoder language model should be able to generate text that is consistent with the language of the training data, e.g. syntactically correct words or even semantically correct sentences.

Word Embeddings

We use word embeddings as input to the text correction model as visualized in Fig. 7.1. To be compatible with the sign detection, our word embeddings are based on characters (i.e. cuneiform signs) instead of words. Nevertheless, we simply refer to them as word embeddings in the following. The usage of pre-trained word embeddings improves the performance of the text correction model, as it offers a rich representation of the individual signs at a reduced dimensionality. The embeddings also speed up the training of the correction model, since the model does not have to learn the representation of individual signs from scratch. We follow the GloVe method [169] and pre-train 128-dimensional GloVe embeddings on a large corpus of cuneiform text. The text correction model not only uses embeddings in the encoder, but also in the decoder (not visualized). By sharing the weights of embeddings in the encoder and decoder (tied embeddings) [172], the text correction model is further regularized, which improves performance and mitigates overfitting during training.

Low Confidence Tokens

Given a line text prediction, the text correction models needs to first locate and then fix all errors in the text. For each position in the input text sequence, the correction model analyses the surrounding signs and checks if the sequence is consistent with its learned language model. Language-related inconsistencies help the model to locate errors. Due to the difficulty of cuneiform sign detection, the line text prediction often contains a high percentage of text errors. This raises the difficulty to localize errors, as it hurts the separation of what is error and what not. To reduce this ambiguity, we leverage the confidence information provided by the sign detector to indicate the presence of an error.

To supply confidence information to the correction model, we insert special low confidence (LC) tokens `<slc>` in front of signs with low detection confidence as shown in Fig. 7.2. We define a minimum confidence threshold below which labels are tagged. As the LC tokens are only used in the input, the seq2seq model will not reproduce them in the output. Thus LC tokens only provide information that a low confidence sign follows.

Table 7.2: Statistics of the TL dataset compared to the Train-TL dataset. The TL dataset comprises over three times as many signs as the Train-TL dataset used for sign detector training.

	TL Dataset	**Train-TL**
number of collections	19 (SAA:1–19)	6 (SAA:1,5,8,10,13,16)
number of clay tablets	4770	1745
number of signs	695636	185399
number of sign code classes	379	214
number of words	304601	-
number of lines	88188	28905
average line length in signs	7.88	6.41
number of continuous sequences	37958	-

LC tokens enable the text correction model to operate in two modes: Normal and aggressive text correction. In the normal mode, the correction model acts more carefully, thus only addressing very obvious errors according to the language model. In the aggressive mode, the correction model performs bolder corrections conditioned on the LC tokens. The correction model will only learn these two correction modes, if LC tokens in the training data provide reliable information about error locations. A high false positive rate of the LC tokens can dilute the expected positive effect. The effect of the minimum confidence threshold and the false positive rate is studied in the following experiment section.

7.4 Experiments

In our experiments we want to analyse the performance of our linguistic refinement approach for improving the output of the cuneiform sign detector. For this purpose, we first introduce a transliteration (TL) dataset that comprises a large corpus of Neo-Assyrian texts and conduct an initial analysis of the dataset based on n-gram statistics. Then, we describe the evaluation metrics and the training of the correction model with synthetic or real data. Finally, we provide a comprehensive evaluation of the line text prediction and line text correction stages.

7.4.1 Transliteration Dataset

We rely on the transliterated cuneiform tablets from the SAAo project as the data source for training the text correction model (including its language model). Unlike the Train-TL dataset described in Sect. 5.5.1, we now use all available transliterations in the SAAo dataset to obtain a larger text corpus sufficient for training the model. This new transliteration dataset (TL dataset) contains more than three times as many cuneiform signs and lines than the Train-TL dataset. Further the TL dataset differs from the Train-TL dataset in three important ways: 1) It includes all transliterations, even ones for which no tablet image could be assigned during data preparation. 2) It includes signs of the transliterations despite them being marked as broken. 3) It covers all sign code classes (not only classes with bounding box annotations), as long as they are listed in the MZL [29]. In Table 7.2 we report the statistics of the TL dataset compared to the Train-TL dataset.

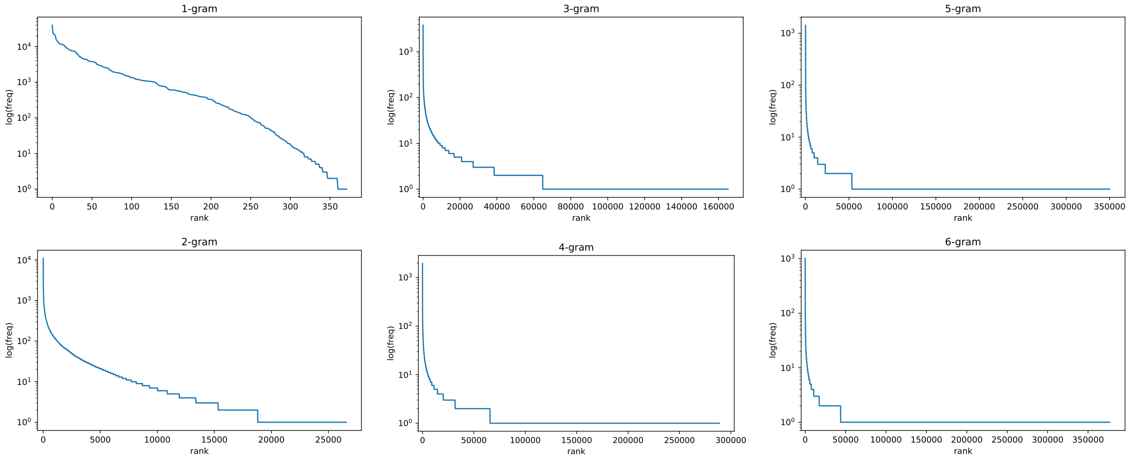


Figure 7.3: Frequency distribution of 1 to 6-grams found in the transliterations of the cuneiform dataset. Each plot ranks the ngrams according to their frequency along the x-axis and plots this frequency in log-space on the y-axis for each ngram.

n-gram Analysis

We conduct an n-gram analysis to study the language-related statistics of the TL dataset. In our experiments, we compute statistics of n-grams up to the sixth order across the whole dataset, which provide a glimpse at the typical language patterns and their frequency in the dataset. First we look at the frequency distribution of all available n-grams of a certain order. Then we take a look at the actual most frequent n-grams.

In Fig. 7.3 we look at the frequency distribution of all 1 to 6-grams extracted from the TL dataset. Similar to the Train-TL dataset, the frequency of cuneiform signs (1-gram) in the TL dataset follows a discrete Pareto-distribution (Zipfian law). There are about 380 different cuneiform signs with a Zipfian frequency distribution which is visible as a roughly linear trend in the log-scale plot. In case of 2 to 6-grams, we observe a rapid growth of n-gram realizations in the TL dataset from about over 25 thousand to over 350 thousand. While the majority of these sign combinations occur only once or twice in the whole dataset (long tail), there is a small subset that occurs very frequently.

In Fig. 7.4 we report the most frequent 1 to 6-grams found in the transliteration dataset. The most frequent (1-gram) cuneiform sign is the vowel *a*. This makes sense as the cuneiform script is a logo-syllabic writing system where *a* is very frequent in syllabic writing. Of course, *a* can also be interpreted as a logogram, e.g. meaning water. Another frequent logogram is the cuneiform sign with the MZL number 266. It is translated as king (*LUGAL*) which might explain its elaborate structure. The king also plays an important role in the most frequent 2 to 6-grams which are mostly related to the formal address of the king in a message: E.g. ”To the king, my lord” is the translation of the most frequent 6-gram (*a-na LUGAL be-lí-ia*). As cuneiform script was originally developed as a bookkeeping tool [69], it soon became an important tool for the administration of kingdoms which is evident here.

We conclude this analysis with a short discussion of the implications for the problem of learning a language model for cuneiform script. A language model should be able to learn statistics comparable to 1 to 6-grams in order to estimate the likelihood of a sequence of

#	Freq	MZLs	Sign0
0	39603	839	𒀀
1	23509	748	𒁹
2	22273	10	𒀭
3	21694	380	𒉌
4	19829	110	𒈾
5	15564	754	𒈨𒌍
6	14816	552	𒈠
7	13670	252	𒄿
8	12809	661	𒌋
9	12072	490	𒌑
10	11725	869	𒋙
11	11655	859	𒐉
12	11582	266	𒈗
13	11409	596	𒌓
14	11176	724	𒅆

#	Freq	MZLs	Sign0	Sign1	Sign2
0	3814	839 110 266	𒀀	𒈾	𒈗
1	3646	266 113 380	𒈗	𒁁	𒉌
2	1957	110 266 164	𒈾	𒈗	𒂗
3	1878	266 164 260	𒈗	𒂗	𒅀
4	1617	113 380 260	𒁁	𒉌	𒅀
5	1479	110 266 113	𒈾	𒈗	𒁁
6	1359	98 839 110	𒈬	𒀀	𒈾
7	1325	748 10 464	𒁹	𒀭	𒉺
8	1278	18 24 748	𒀴	𒅗	𒁹
9	1142	736 98 839	𒁲	𒈬	𒀀
10	1047	724 748 10	𒅆	𒁹	𒀭
11	985	260 18 24	𒅀	𒀴	𒅗
12	971	10 695 596	𒀭	𒀫	𒌓
13	793	380 552 839	𒉌	𒈠	𒀀
14	787	498 628 859	𒂄	𒄭	𒐉

#	Freq	MZLs	Sign0	Sign1	Sign2	Sign3	Sign4
0	1426	839 110 266 113 380	𒀀	𒈾	𒈗	𒁁	𒉌
1	1377	839 110 266 164 260	𒀀	𒈾	𒈗	𒂗	𒅀
2	1012	110 266 113 380 260	𒈾	𒈗	𒁁	𒉌	𒅀
3	791	736 98 839 110 266	𒁲	𒈬	𒀀	𒈾	𒈗
4	629	98 839 110 266 164	𒈬	𒀀	𒈾	𒈗	𒂗
5	513	812 736 98 839 110	𒇻	𒁲	𒈬	𒀀	𒈾
6	510	164 260 18 24 748	𒂗	𒅀	𒀴	𒅗	𒁹
7	476	266 164 260 18 24	𒈗	𒂗	𒅀	𒀴	𒅗
8	469	110 266 164 260 18	𒈾	𒈗	𒂗	𒅀	𒀴
9	463	10 695 596 839 110	𒀭	𒀫	𒌓	𒀀	𒈾
10	460	113 380 260 18 24	𒁁	𒉌	𒅀	𒀴	𒅗
11	456	380 260 18 24 748	𒉌	𒅀	𒀴	𒅗	𒁹
12	432	839 110 266 164 861	𒀀	𒈾	𒈗	𒂗	𒐊
13	422	10 596 164 553 490	𒀭	𒌓	𒂗	𒃲	𒌑
14	416	266 113 380 260 18	𒈗	𒁁	𒉌	𒅀	𒀴

#	Freq	MZLs	Sign0	Sign1
0	10969	839 110	𒀀	𒈾
1	4774	748 10	𒁹	𒀭
2	4664	724 748	𒅆	𒁹
3	4313	552 839	𒈠	𒀀
4	4230	113 380	𒁁	𒉌
5	4103	110 266	𒈾	𒈗
6	3857	266 113	𒈗	𒁁
7	3844	839 839	𒀀	𒀀
8	3233	266 164	𒈗	𒂗
9	2690	10 380	𒀭	𒉌
10	2642	661 380	𒌋	𒉌
11	2334	869 112	𒋙	𒉡
12	2100	754 869	𒈨𒌍	𒋙
13	2077	737 252	𒆠	𒄿
14	2056	164 260	𒂗	𒅀

#	Freq	MZLs	Sign0	Sign1	Sign2	Sign3
0	1955	839 110 266 164	𒀀	𒈾	𒈗	𒂗
1	1471	839 110 266 113	𒀀	𒈾	𒈗	𒁁
2	1433	110 266 113 380	𒈾	𒈗	𒁁	𒉌
3	1433	266 113 380 260	𒈗	𒁁	𒉌	𒅀
4	1377	110 266 164 260	𒈾	𒈗	𒂗	𒅀
5	1084	736 98 839 110	𒁲	𒈬	𒀀	𒈾
6	979	260 18 24 748	𒅀	𒀴	𒅗	𒁹
7	928	98 839 110 266	𒈬	𒀀	𒈾	𒈗
8	677	566 266 113 380	𒊭	𒈗	𒁁	𒉌
9	534	266 113 380 861	𒈗	𒁁	𒉌	𒐊
10	516	812 736 98 839	𒇻	𒁲	𒈬	𒀀
11	512	164 260 18 24	𒂗	𒅀	𒀴	𒅗
12	507	357 559 10 380	𒅖	𒁓	𒀭	𒉌
13	502	18 24 748 10	𒀴	𒅗	𒁹	𒀭
14	500	10 695 596 839	𒀭	𒀫	𒌓	𒀀

#	Freq	MZLs	Sign0	Sign1	Sign2	Sign3	Sign4	Sign5
0	1012	839 110 266 113 380 260	𒀀	𒈾	𒈗	𒁁	𒉌	𒅀
1	538	736 98 839 110 266 164	𒁲	𒈬	𒀀	𒈾	𒈗	𒂗
2	475	266 164 260 18 24 748	𒈗	𒂗	𒅀	𒀴	𒅗	𒁹
3	469	839 110 266 164 260 18	𒀀	𒈾	𒈗	𒂗	𒅀	𒀴
4	467	110 266 164 260 18 24	𒈾	𒈗	𒂗	𒅀	𒀴	𒅗
5	465	812 736 98 839 110 266	𒇻	𒁲	𒈬	𒀀	𒈾	𒈗
6	456	113 380 260 18 24 748	𒁁	𒉌	𒅀	𒀴	𒅗	𒁹
7	439	98 839 110 266 164 260	𒈬	𒀀	𒈾	𒈗	𒂗	𒅀
8	411	110 266 113 380 260 18	𒈾	𒈗	𒁁	𒉌	𒅀	𒀴
9	409	266 113 380 260 18 24	𒈗	𒁁	𒉌	𒅀	𒀴	𒅗
10	396	10 695 596 839 110 266	𒀭	𒀫	𒌓	𒀀	𒈾	𒈗
11	349	812 661 736 98 839 110	𒇻	𒌋	𒁲	𒈬	𒀀	𒈾
12	323	113 380 357 559 10 380	𒁁	𒉌	𒅖	𒁓	𒀭	𒉌
13	316	566 266 113 380 357 559	𒊭	𒈗	𒁁	𒉌	𒅖	𒁓
14	305	266 113 380 357 559 10	𒈗	𒁁	𒉌	𒅖	𒁓	𒀭

Figure 7.4: The top-fifteen 1 to 6-grams extracted from the transliterations of the cuneiform dataset are visualized in six tables. Each table shows the ngrams sorted according to their frequency (Freq) which is reported alongside the MZL numbers and the corresponding unicode symbols.

cuneiform signs. In the case of sign combinations from the long tail (low-frequency part) of the n-gram distributions, most combinations appear equally likely. Thus a language model based on 1 to 6-grams cannot provide additional validation for sign combinations of this long tail. A LSTM-based language model, like our proposed text correction model, can inform its decisions on larger context windows (¿ 20 signs) and is able to learn implicit dependencies between signs (unseen in the dataset). In the following experiments, we explore the limits of the text correction model trained on the TL dataset.

7.4.2 Evaluation Metric

To evaluate the performance of our approach for text prediction and correction, we require a metric for measuring the similarity between a candidate and a reference text sequence. We opt for a metric based on the well-known edit distance (ED), in particular the Levenshtein edit distance.

Levenshtein Edit Distance

The Levenshtein edit distance quantifies the minimal necessary number of operations to transform a text sequence t_1 into the reference sequence t_2. The Levenshtein edit distance counts three types of transform operations:

- i : number of characters inserted
- d : number of characters deleted

- s : number of characters substituted

As there are multiple sequences of operations to transform t_1 into t_2, the edit distance searches the transformation that minimizes the sum of $i + d + s$. The distance is computed as the solution of an optimization problem that can be solved efficiently using dynamic programming [37]. In the context of machine translation, several different metrics have been proposed like BLEU, ROGUE, or METEOR [17, 132, 165]. The machine translation setting assumes that there are multiple valid ways to translate a sentence from one language to another language. In contrast, the setting of text correction assumes a unique ground truth sequence against which to evaluate. Besides providing a good performance measure for text correction, the Levenshtein distance also counts the contribution of each of the three error types to the overall distance allowing for additional analysis (reference).

Character-Error-Rate

As text sequences have different lengths, we need a version of the Levenshtein distance that is comparable across variable lengths of text sequences. The Word-Error-Rate (WER) or Character-Error-Rate (CER) is such a performance measure well-known in the fields of speech recognition and optical character recognition. As the cuneiform sign detection and correction problem is character-based, we rely on the CER for performance evaluation. The CER is defined as follows

$$CER = \frac{i + d + s}{n}\,, \tag{7.1}$$

where i, d, s are the number of operations from the Levensthein distance and n is number of characters in reference sequence t_2. The CER effectively provides a form of a length normalized edit distance. However, the CER can be larger than one, since the length of t_1 and t_2 can be different.

7.4.3 Line Text Prediction

The evaluation of line text prediction not only provides a baseline for the the line text correction stage Sect. 7.4.4, but also offers a complementary perspective on the performance of the sign detector in Chapter 6. In particular, line text prediction allows the evaluation of the sign detection performance across the whole SAAo collection, since its evaluation does not require bounding box annotations unlike the evaluation of the sign detector in Chapter 6. In addition, line text prediction gives valuable insights on the overall problem of cuneiform sign detection, since it enables the quantification of the discrepancy between clay tablet images, sign annotations and transliterations.

Configuration

We use the following configuration for all line text prediction experiments. We employ the line and sign detector from Chapter 6 trained on the full Train split (see Sect. 5.5.1) to obtain the line text prediction for all SAAo collections. Sign detections with strongly overlapping bounding boxes are removed by applying NMS with an overlap threshold of 0.3, and only sign detections with a detection confidence larger than 0.1 are included.

To obtain ground truth targets for evaluation, we perform line-level alignment as described in Sect. 6.4.5. Only successfully aligned transliterated lines can be used during evaluation.

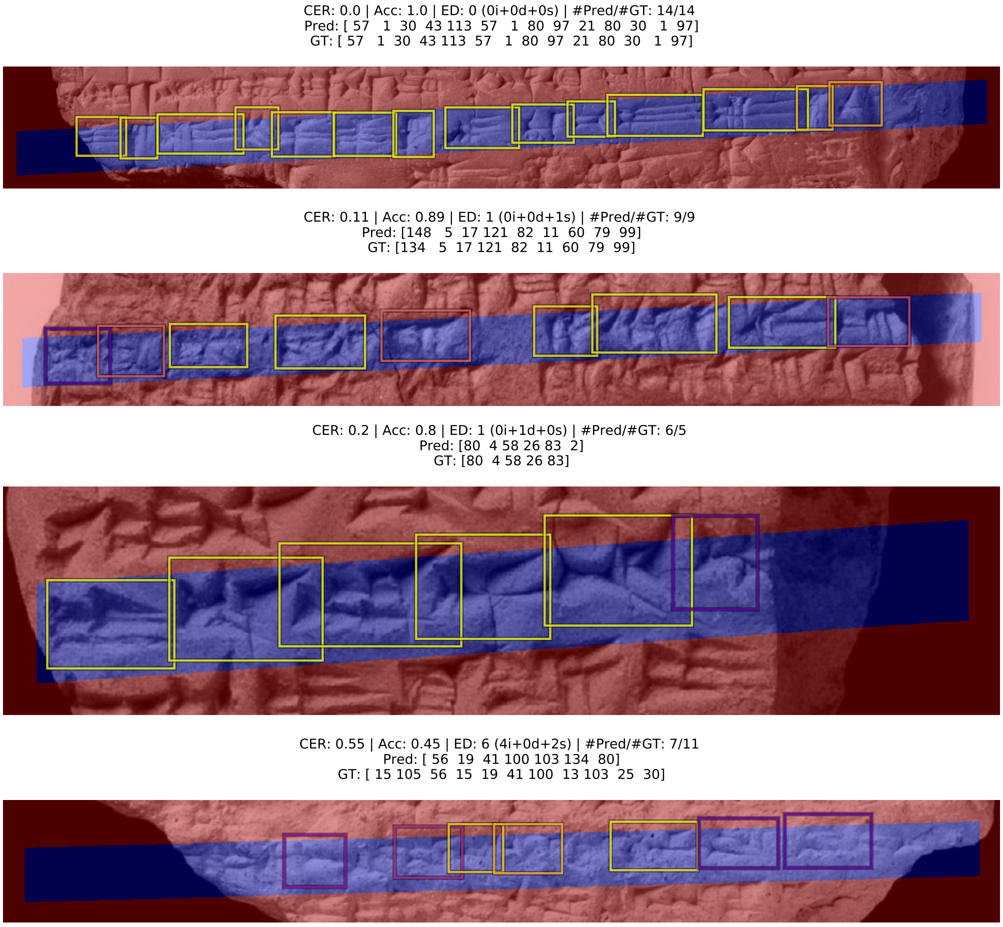

CER: 0.0 | Acc: 1.0 | ED: 0 (0i+0d+0s) | #Pred/#GT: 14/14
Pred: [57 1 30 43 113 57 1 80 97 21 80 30 1 97]
GT: [57 1 30 43 113 57 1 80 97 21 80 30 1 97]
CER: 0.11 | Acc: 0.89 | ED: 1 (0i+0d+1s) | #Pred/#GT: 9/9
Pred: [148 5 17 121 82 11 60 79 99]
GT: [134 5 17 121 82 11 60 79 99]
CER: 0.2 | Acc: 0.8 | ED: 1 (0i+1d+0s) | #Pred/#GT: 6/5
Pred: [80 4 58 26 83 2]
GT: [80 4 58 26 83]
CER: 0.55 | Acc: 0.45 | ED: 6 (4i+0d+2s) | #Pred/#GT: 7/11
Pred: [56 19 41 100 103 134 80]
GT: [15 105 56 15 19 41 100 13 103 25 30]

Figure 7.5: Line text prediction of four lines of cuneiform script found on different clay tablets. For each line we visualize the line and sign detections used as input (lighter color of bounding boxes indicates higher detector confidence). Above each line figure we report CER, Acc=1-CER, Levenshtein ED including the respective edit operations as well as predicted and ground truth transliteration of the line.

While the line-level alignment omits a large number of signs in the TL dataset, it still yields over 250 thousand signs for the evaluation of the text line prediction. Compared to the evaluation with manual annotations in the previous chapter, there are over one order of magnitude more signs available. We also exclude tablets with a CER above a fixed threshold (0.7) which most likely are affected by errors introduced during data preparation and should be avoided in the following experiments.

Qualitative Results

Fig. 7.5 visualizes the line and sign detections used to produce the line text predictions in the case of four lines from different clay tablets. The blue line detection box functions similar to a selection window for the bounding box detections. Starting from the topmost example in Fig. 7.5, we observe the following: The first two lines show text prediction results with only a small error. In the third line the text prediction contains an additional sign which is partly broken and thus omitted from the transliteration. In the fourth line the text prediction is missing multiple signs compared to the transliteration, as image and tablet quality inhibits detection.

Despite its simplicity, line text prediction yields decent results mostly due to the quality of the sign detector. However, errors in line and sign detection can amplify each other and produce bad line text prediction results as in the bottom example of Fig. 7.5. Therefore a control mechanism like a language model is needed to check the linguistic validity of the results, e.g. like a syntax or semantics check, as line text prediction constitutes a form of open-loop prediction.

Quantitative Results on all SAAo Collections

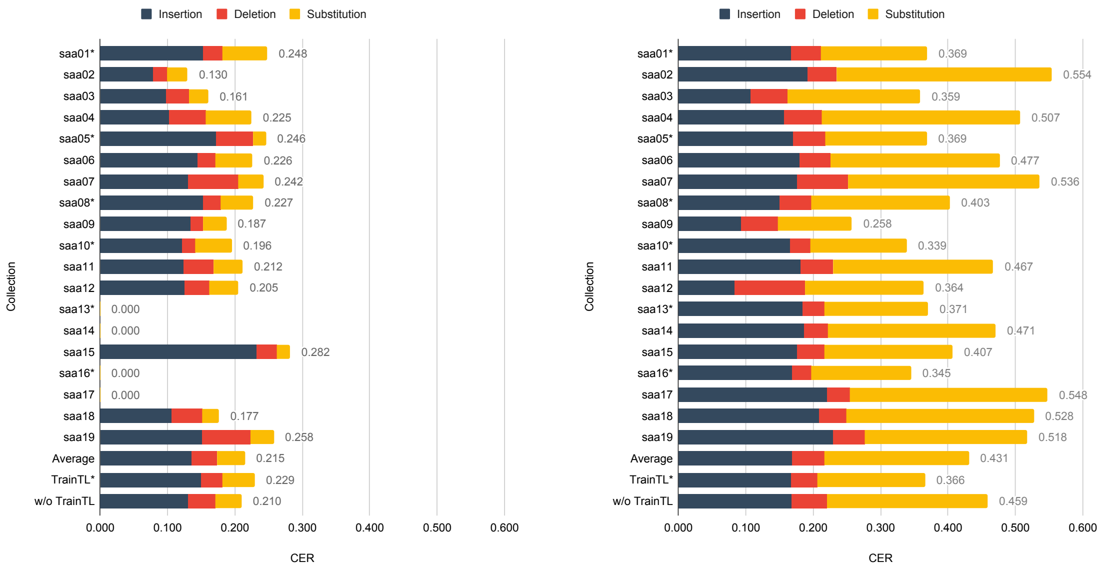
In this experiment, we want to evaluate the performance of the sign detector across all SAAo collections, even if no bounding box annotations are available. To this end, we compare the performance of line text prediction based on detections with the performance of line text prediction based on annotations. The line text prediction of a clay tablet is primarily based on the line and sign detections in the tablet image, while the evaluation of its CER also relies on the corresponding transliteration. To account for the discrepancy between tablet image and transliteration, we estimate the lowest achievable CER by additionally conducting the line text prediction based on the line and sign annotations.

It is important to note that line and sign annotations are only available for a small subset of all clay tablets in SAAo and that the number of available annotations varies between collections. In the case of four collections, no annotations are available, while the remainder of the collections has on average about 1500 sign annotations. We find that the estimate of the lowest achievable CER is robust across all collections despite the relative small and varying number of annotations.

Fig. 7.6 visualizes the results of the performance evaluation of line text prediction based on annotations as well as detections:

Annotations Case The line text prediction based on line and sign annotations achieves an average CER of 0.215 and the edit operations are distributed as follows: 63% of the operations are insertions (0.135 CER), 18% deletions (0.039 CER) and 19% substitutions (0.041 CER). While there is some variance of CER across collections (0.13 to 0.282 CER), the relative distribution of edit operations appears stable across collections. Moreover, there is no significant performance difference between Train-TL collections and the remainder.

Detections case When based on line and sign detections, the line text prediction achieves an average CER of 0.431 and the edit operations are distributed as follows: 39% of operations are insertions (0.168 CER), 11% deletions (0.048 CER) and 50% substitutions (0.215 CER). We observe a strong increase in substitutions compared to the annotation based evaluation. Substitutions in the text prediction domain can be associated with false positive sign detections due to sign code class confusion as discussed in the context of Table 7.1. This connection is supported by the fact, that both substitution and class confusion errors are each the most common error source for the task of line text prediction and sign detection respectively, as in Sect. 6.6.2. Interestingly, there is only a small absolute change in insertions and deletions compared to the annotations case. Thus we assume that line text detection (including sign detection as pre-processing) is very good in avoiding background confusions (false positive and false negative). Similar to the annotations case,



(a) Line and sign annotations.

(b) Line and sign detections.

Figure 7.6: Evaluation of line text prediction performance across all collections of the SAAo dataset. Performance is reported as character-error-rate (CER). Collections marked with an asterisk are part of the Train-TL dataset that is used for the weakly supervised training of the sign detector. As the CER is based on the edit distance the contribution of insertion, deletion and substitution to the overall CER is visualized as different colors.

we observe some variance of CER across the collections (0.258 to 0.554 CER), while the relative distribution of edit operations appears stable across collections. On the collections of Train-TL dataset, we find a performance improvement compared to the remainder of the collections (0.366 to 0.459 CER). This difference might be due to the fact that the sign detector overfits the Train-TL dataset, since new annotations are generated from this weakly supervised dataset during iterative training of the sign detector (cf. Sect. 6.5.5),

Quantifying the Discrepancy between Transliteration, Tablet Image and Annotations

Previously, we used the text line predictions based on annotations as an estimate of the lowest achievable CER. However, this estimate is biased, because the CER for line text prediction is ultimately based on the sign detections in the tablet image, whereas the annotation case rather measures the discrepancy between annotations and transliteration. This bias becomes clear if we look at the annotation process described in Sect. 5.3: Since the annotations for a side of a clay tablet are created based on a particular tablet image, their error are conditioned on the quality of the tablet image (e.g. if a bad tablet image only shows half of the obverse of a clay tablet as recorded in the transliteration, the image's annotations miss this half accordingly). A better estimate of the lowest achievable CER for line text prediction is the discrepancy between tablet image and transliteration. Since this discrepancy is not directly observable, we propose a simple approximation method. Besides yielding a better estimate of the lowest achievable CER, the method also allows to quantify the quality of the manual annotations.

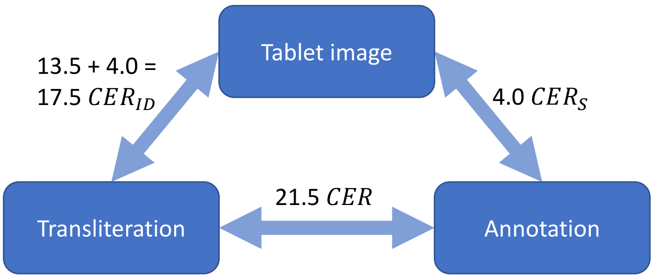


Figure 7.7: Triangle of discrepancies between annotations, tablet image and transliteration. We estimate the discrepancies for the entire SAAo dataset using the Levenshtein edit distance and two assumptions as described in the text below.

Fig. 7.7 visualizes the problem of estimating the discrepancies as a triangle with its three corners, labeled annotations, tablet image and transliteration, and its edges representing three discrepancies: tablet image vs. transliteration, transliteration vs. annotation, and annotation vs. tablet image. While we can estimate the error between transliteration and annotations (in terms of the CER for the text line prediction based on annotations), we cannot directly observe the error along the two other edges of the triangle. To obtain the two other discrepancies, we rely on the Levenshtein edit distance and two assumptions which we describe in the following: Given the text line prediction based on annotations, we approximate the discrepancy between tablet image and transliterations with the CER$_{ID}$ (based on insertions and deletions only) and the discrepancy between tablet image and annotations with the CER$_S$ (based on substitutions only). Our first assumption is that during the manual annotation process human experts only produce class confusion errors (but none of the other FP error types). This is based on the observation that human annotators spot localization and background confusion errors (i.e. the absence of any sign) far more easily than class confusion errors (i.e. the confusion of sign code classes). Since all substitutions (FP class confusions) are explained by the discrepancy between tablet image and annotations, our second assumption is that the insertions and deletions account for the discrepancy between tablet image and transliteration. This is based on the observation that transliterations can be considered free of substitution errors, as they have been produced by experts. Therefore the discrepancy between tablet image and transliteration is mostly caused by the problem that a transliteration contains more information than visible or vice versa (as discussed in the context of image-transliteration alignment in Sect. 6.3).

Fig. 7.7 visualizes the resulting estimates of all three discrepancies of the SAAo dataset along the edges of the triangle. Our best estimate for the lowest achievable CER for text prediction (evaluated using transliterations) is 0.175 averaged across all collections. This estimate is biased because it incorporates errors caused by data preparation (Sect. 5.2.3) and line-level alignment (Sect. 6.4.5). However, we assume this bias to be small, since these results are based only on tablets for which line and sign annotations exist: Firstly, errors caused by the data preparation are usually reported and fixed during the manual annotation process, and secondly, the line-level alignment is very reliable, if it is based on ground truth line and sign annotations. The discrepancy between the tablet image and the annotations provides us with a good estimate of the annotation error. The annotations have an average CER of about 0.04 which appears reasonable for such a fine-grained detection task. Nevertheless, the manual annotation error in datasets like ImageNet (0.003) is ten times smaller [184].

Quantifying the Error due to Data Preparation

We introduced a CER threshold to avoid errors introduced during data preparation (see Sect. 5.2.3), as we described in the configuration of this experiment. While the CER threshold limits the number of potential error sources during the presented experiment, it can be also leveraged to estimate the error of data preparation. For this purpose, we evaluate text prediction using line and sign detections with different CER thresholds of 0.7, 0.9 and 1.0 (unfiltered). We find that the unfiltered text predictions have an average CER that is 0.05 to 0.1 higher which we consider a rough estimate for the error of data preparation.

7.4.4 Line Text Correction

In the following experiments, we evaluate the performance of the line text correction stage of our linguistic refinement approach. First, we provide details on the training of word embeddings for cuneiform script and visualize the word embeddings for qualitative analysis. Then we describe the training and testing configuration of the text correction model. Finally, we conduct a through analysis of the line text correction model in three experiments with evaluation on synthetic, real and n-gram data.

Pre-trained Word Embeddings

The pre-trained word embeddings provide a rich representation of the cuneiform signs that are fed into the text correction model. To obtain word embeddings for cuneiform script, we train 128-dimensional GloVe embeddings on the TL dataset for all present 380 cuneiform signs. We use a context window of five signs for the skip-gram generation. The GloVe embeddings are trained with an Adam optimizer, with a learning rate of 0.01 and a batch size of 1024 for 30 iterations.

To visualize the GloVe embeddings, we employ the t-distributed Stochastic Neighbor Embedding (t-SNE) [143]. It projects the high-dimensional word vectors of the word embeddings onto the 2D plane. The method aims for nearby points in high-dimensional space to remain nearby in 2D space, thus optimizing the projection to maintain the neighbourhood relationships. In the case of word embeddings, a t-SNE visualization is particularly interesting, since proximity in the embedding space can indicate a semantic relationship between signs/words as demonstrated in [149, 150].

Fig. 7.8 visualize the trained GloVe embeddings for 120 cuneiform signs using t-SNE. As cuneiform signs are often used as syllables rather than individual words, clusters of signs do not necessarily indicate a semantic relationship. More often, clusters represent groups of consonants and vowels that appear in the same context (e.g. *BU, RU, UR* towards the lower left corner). Similarly, the reading *LUGAL* (king) towards the center of each plot is surrounded by signs (e.g. *IA, BAD, EN*) that all appear in Fig. 7.4 as part of the most frequent n-grams in cuneiform script. Nevertheless, the cuneiform sign embeddings also encode a semantic context as visible towards the lower right corner where a large group of number signs are clustered together (e.g. *U, AS, ES, LIMMU5, NINNU, IMIN*).

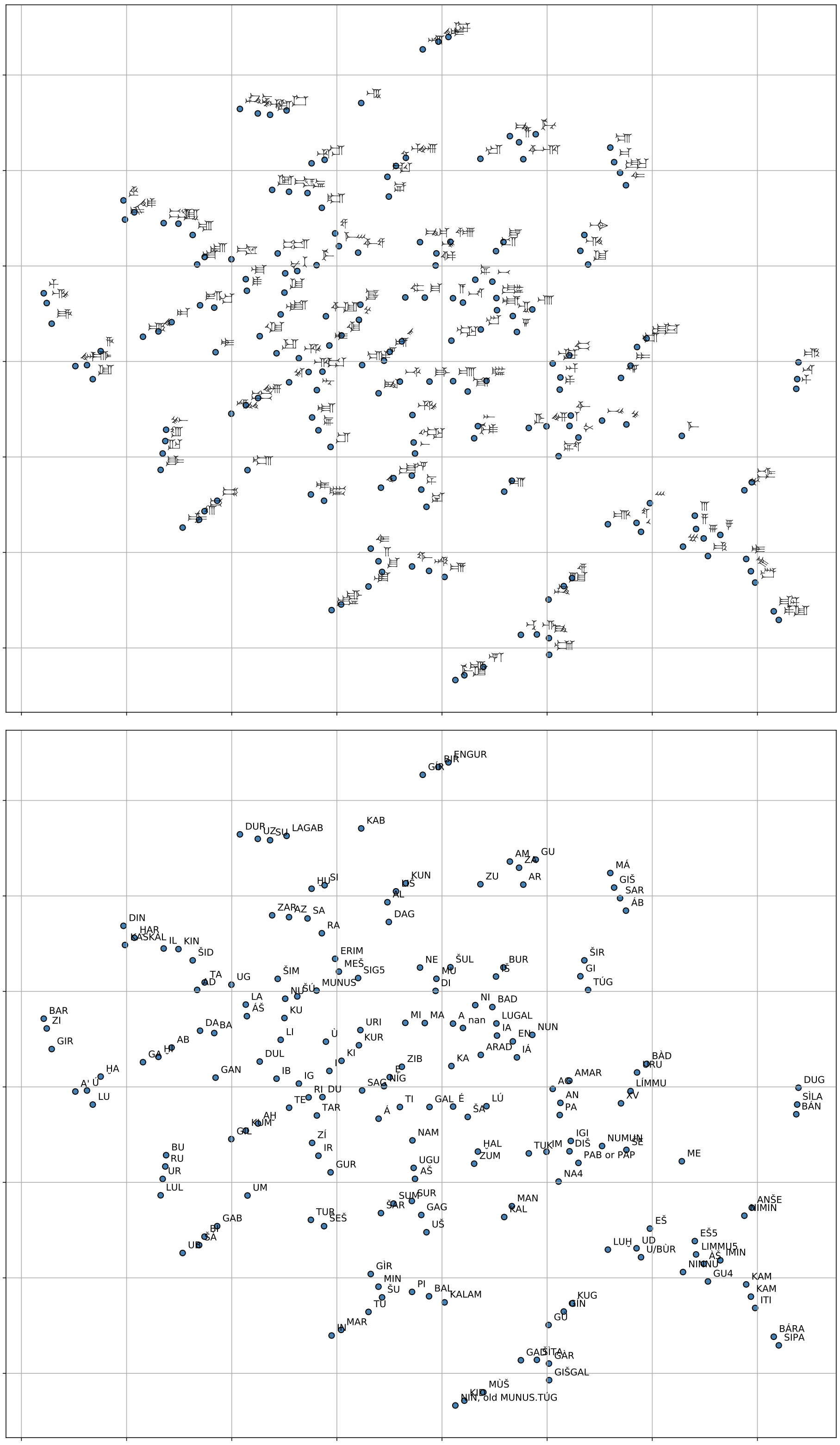


Figure 7.8: Visualization of GloVe word embeddings for cuneiform signs using the t-SNE method. Proximity between words indicates similarity. The cuneiform signs are either depicted in the form of Unicode symbols (top) or readings (bottom).

Table 7.3: Train and test split of SAAo collections for line text correction.

	train set	test set
synthetic data	1–3; 5–19	4
real data	1,5,8,10,13,16	4

Configuration

For training and testing of the text correction model, we use the TL datasets spanning all collections of SAAo. The train and test split of the SAAo collections are shown in Table 7.3. We use one collection for testing and dedicate the rest for training. Instead of using all available sign code class, we follow the convention from Sect. 6.5.1 and only use ones for which we have trained the sign detector.

The correction model is trained for 40 epochs using mini-batch stochastic gradient descent with batch size of 64 and an Adam optimizer with a learning rate of 0.0005 that is reduced by a factor 0.1 after 30 epochs. Dropout of 0.4 is applied at the penultimate layer of the decoder. Teacher-forcing [222] is employed with a rate of 0.8 or 0.5 for synthetic or real data respectively. During evaluation, the decoder employs greedy decoding to produce the refined text, i.e. at each time step predicting the most likely sign. Using beam search [72] does not yield an improvement.

Experiments evaluated on Synthetic Data

This experiment investigates the performance of the text correction model on synthetic data. Synthetic data allows a proof of concept of our text correction model under controlled conditions. By altering the quantity and quality of the noise in the input, we can explore the limits of our approach and obtain a baseline for the real data evaluation. In the following, we describe the preparation of the synthetic data and present our results of training and testing the text correction model on synthetic data.

Synthetic Data Preparation Synthetic data samples are generated from the TL dataset as visualized in Fig. 7.1: First, a transliterated line is augmented to obtain the target text, and then synthetic noise is added to the augmented line to obtain the noisy input text. In the following we describe the configuration of augmentation, synthetic noise injection and how LC tokens are included in the synthetic case:

- **Augmentation** The data is augmented by random substitution of five percent of signs in the original transliterated line. As usual, augmentation is only used during training.

- **Synthetic noise** As visualized in Fig. 7.9, the composition of synthetic noise for training and testing is the following: 2% random insertions, 2% random deletions and 20% random substitutions of signs. This composition is driven by the empirical findings in Sect. 7.4.3 that most errors in the line text prediction are substitution errors which affect about about 20% of the transliterated line as shown in Fig. 7.6b. During training, we do not use sign deletion noise to reduce the difficulty of the text correction task. We find the deletion noise to be the most challenging for training.

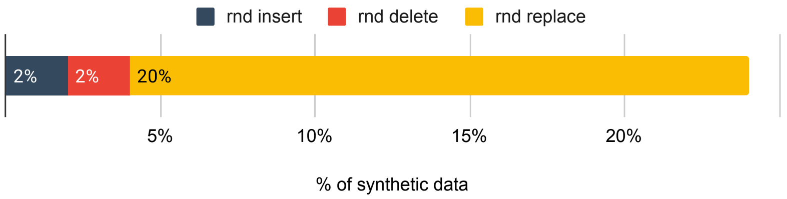


Figure 7.9: Error composition of all synthetic data configurations used for evaluation. The synthetic data set is focused on substitution errors, which account for most of the errors produced by the sign detector as observed in Fig. 7.6b.

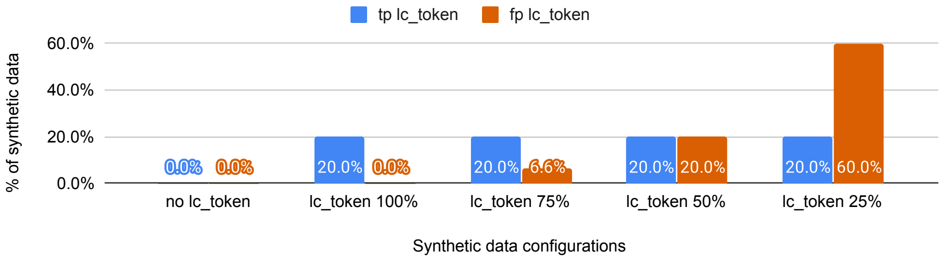


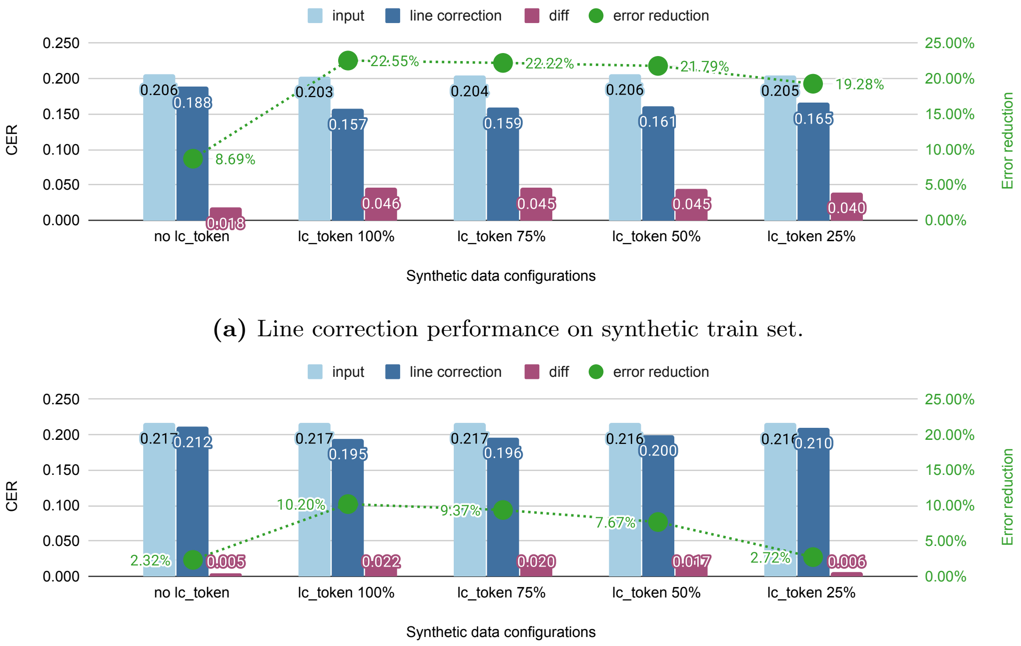
Figure 7.10: For evaluation, we use five synthetic data configurations with different amounts of LC tokens. Besides the first configuration without LC tokens, the four other configurations differ in the amount of false positive LC tokens (i.e. LC token falsely marking a valid sign for correction).

- **LC tokens** Since we know the location of each error in the synthetic case, we control the rate of true positives of the LC tokens, i.e. the percentage of errors correctly labeled with an LC token and the percentage of correct signs erroneously labeled with an LC token. During training, we mark a subset of substituted signs with LC tokens and ensure that exactly a third of LC tokens are true positives during training. During evaluation, we use one of five synthetic data configurations for LC tokens as shown in Fig. 7.10: One configuration without any LC tokens as a baseline and four configurations with 100%, 75%, 50% or 25% of LC tokens being true positive.

Results In Fig. 7.11 we evaluate the train and test performance of the line text correction on synthetic data for each of the five synthetic data configurations. We report the CER of input and output of the text correction as well as their difference, and measure the performance of text correction in terms of the relative error reduction of input CER.

On the training set, we find a significant error reduction of up to 22% for the configurations that make use of LC tokens. Without LC tokens, the error reduction is only a little over 8%. The TP rate of the LC tokens has only a weak impact (unlike in the test case), most likely due to overfitting to the training data (the detector learns the correct sequences by heart).

On the test set, we find an error reduction of up to 10% with LC tokens and only about 2 % without LC tokens. The information provided by the LC token, i.e. knowing about the existence and location of an error, has a significant effect (error reduction of 10.2% vs. 2.3%). However, LC tokens less than 50% correct results in substantial performance drop: 25% TP tokens with only 2.7% error reduction compared with 50% TP tokens with already 7.7% error reduction. Despite of this drop, the 25% TP tokens still produce a better result



(a) Line correction performance on synthetic train set.

(b) Line correction performance on synthetic test set.

Figure 7.11: Evaluation of line correction model on synthetic data.

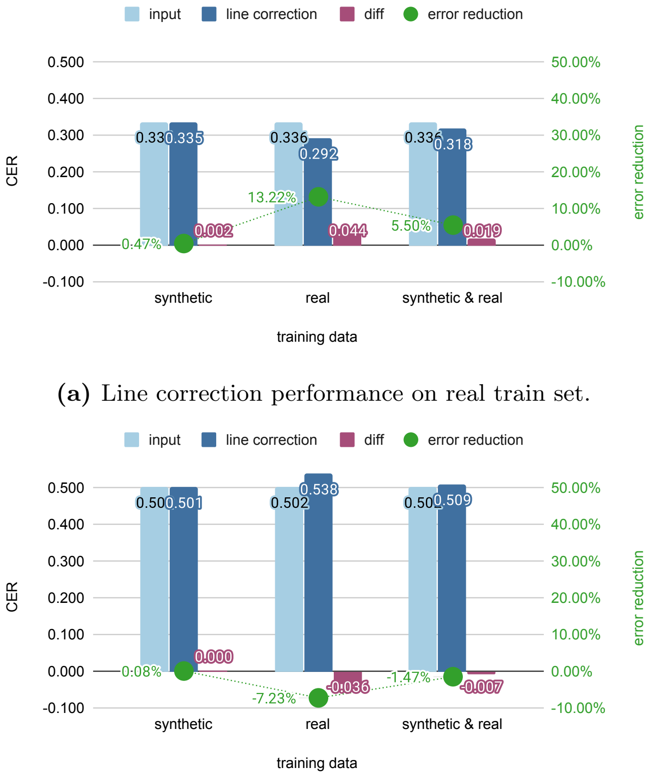
than no LC tokens. The correction model benefits from the information of the LC tokens despite the low TP rate.

Overall, we show that the text correction can be trained to localize and fix errors in a controlled setting. LC tokens can result in a significant performance boost by providing important guidance for the text correction model.

Experiments evaluated on Real Data

After having the text correction model successfully trained and evaluated on synthetic data (reference to previous section), we investigate its performance on real data in this experiment. Ideally, the text correction model can improve on the results of line text prediction presented earlier in Fig. 7.6b. First, we provide details on the real data generation, then, we describe the training of the text correction model on synthetic and real data, and finally, report the results of our experiment.

Real Data Preparation We obtain a pair of input and target text as visualized in Fig. 7.1. For text line prediction, we follow the configuration of the text line prediction experiment (reference). In contrast to synthetic data it is not necessary to inject additional noise, since real input texts already come with high levels of noise as we have shown in Fig. 7.6b. During training, too noisy samples are even discarded, if the CER of the input text is larger than 0.7. To deal with low confidence detections, LC tokens are inserted in front of signs with a detection confidence smaller than 0.3. If a line is longer than fifty signs it is cropped.



(a) Line correction performance on real train set.

(b) Line correction performance on real test set.

Figure 7.12: Evaluation of line correction model on real detection data.

Training on Synthetic and Real Data The text correction model trained on synthetic and real data is obtained by pre-training on synthetic data and then fine-tuning on real and synthetic data. For our experiment, we use as pre-trained model the text correction model from the experiment on synthetic data and fine-tune it on a mixture of 4/5 real and 1/5 synthetic data for 20 epochs using the same learning rate as during regular training.

Results In Fig. 7.12 we evaluate the train and test performance of the line text correction on real data. We compare the performance of three text correction models trained on either 1) synthetic data, 2) real data, or 3) synthetic & real data. As in the previous experiment, we report the CER of input and output of the text correction as well as their difference, and measure the performance of text correction in terms of the relative error reduction of input CER.

On the training set, we find that the text correction model trained on synthetic data only produces a very small error reduction of about half a percent. In contrast, the model trained on real data (13 %) and synthetic & real (5.5%) both improve the results. As the noise in real data is different from the noise in synthetic data, the model trained on synthetic data probably struggles to correct any but the very obvious errors (e.g. errors in the most frequent n-grams). The two other models benefit from training on real data.

On the test set, we observe that the text correction model trained on synthetic data now results in even less than 0.1 % error reduction. If the models are trained on real and real &

synthetic data, the error even increases. These two models seem to suffer from overfitting on the real training data. Combining synthetic and real data reduces the overfitting from over -7% to about -1.5%, but we still find no positive error reduction.

For a model trained purely on synthetic data, we hypothesize that the difference between the noise of synthetic and real data is making the error correction difficult. This model performs worse on the test set than on the train set, since only sees samples from the train set during training (even if only with synthetic noise).

Unlike in the synthetic case, the usage of LC tokens does not yield a significant performance gain. This is most likely due to the low true positive rate of LC tokens. While the detection confidence is not an arbitrary measure, it seems too unreliable that the text correction model can exploit it properly.

Experiments evaluated on n-gram Data

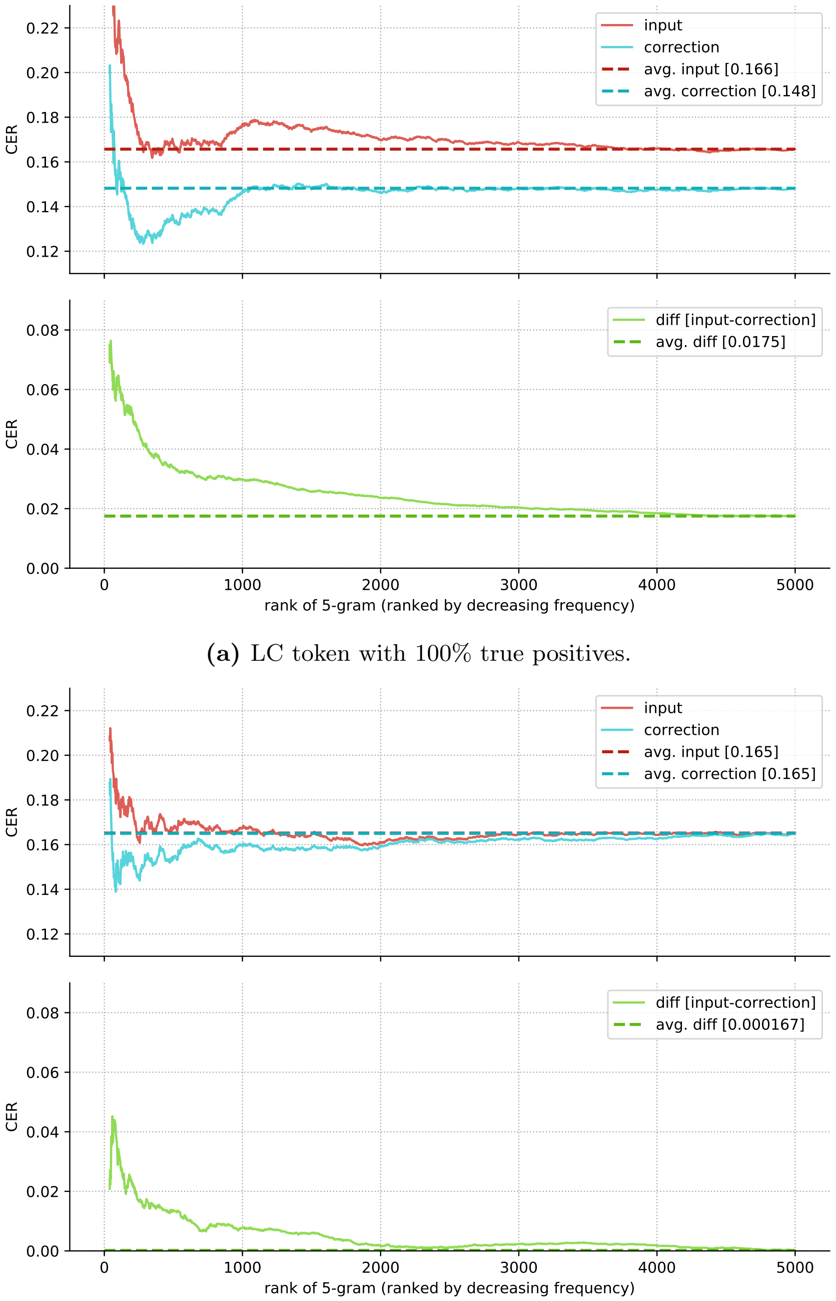
As the text correction model yields mixed results in the previous experiments on real and synthetic data, we take a closer look at what the model has learned. To better understand which errors the text correction model is able to fix and which not, we evaluate the performance of the text correction model on a synthetic n-gram data. The ability of n-gram correction provides valuable information about the ability to correct general errors in a whole line of cuneiform text, since a line is literally constructed from n-grams (and vice versa). In particular, we make use of the n-grams we identified in the TL dataset in Sect. 7.4.1 and combine them with synthetic noise to obtain synthetic n-gram data.

Synthetic n-gram Data Preparation For this experiment, we focus on the five thousand most frequent 5-grams found in the TL dataset. To add the synthetic noise to the 5-gram inputs, we make use of the synthetic data configurations 1 and 4 as described in (reference table). Thus on average one of the five signs of the 5-gram is replaced with a random sign and the used LC tokens are either 25% of the time or 100% correct.

Results In Fig. 7.13, we study the performance of the text correction model on our synthetic n-gram data. For this experiment, we use the text correction model trained on synthetic data in (reference). In the case of synthetic data configurations 1 and 4, we plot the performance of the text correction model for all five thousand 5-grams ranked according to their occurrence frequency along the x-axis. On the y-axis, the performance is measured in terms of CER of noisy input 5-gram, refined output and their difference.

We observe a correlation of text correction performance and 5-gram frequency: the more frequent a 5-gram occurs in the 5-gram data, the more frequent it occurs in the train set and the more likely it is learnt by the text correction model. It should be noted that this is not a simple overfitting on the training data, as the 5-grams usually occur as parts of longer lines of text. Thus the text correction model has learnt to correct noisy 5-grams, independent of the surrounding context, which resembles some form of generalization.

Another takeaway from the experiment is the effect of the LC tokens and their true positive rate. In the case of correct LC tokens, we find a clear improvement for the first five thousand 5-grams, visible in the gap between input and correction as well as in their difference. In the case of 25% TP LC tokens, we find a smaller improvement, which is only visible for the first 2000 most frequent 5-grams. While LC tokens have to be correct in order to be helpful, the frequency of n-grams always helps.



(a) LC token with 100% true positives.

(b) LC token with 25% true positives.

Figure 7.13: Evaluation on synthetic 5-gram data. The performance of the line correction model is visualized in the case of 100% true positive LC tokens (left column) and in the case of 25% true positives (right column). The CER of the noisy input and the CER of the output of the correction model are visualized in the upper row of plots, while the effective CER improvement is visualized in the lower row. The solid lines represent the cumulative moving averages of the errors which expand from left to right and gradually converge towards the corresponding dashed lines which represent the average across all five thousand 5-grams. The correction model has implicitly learned to correct a large number of 5-grams. The model performs best for the most frequent 5-grams. The model benefits from the TP LC tokens, as it clearly corrects more 5-grams including less frequent ones.

Overall, the text correction model learns meaningful aspects of the cuneiform language. However, only the most frequent 5-grams can be corrected properly in the presence of noisy inputs and without any reliable error localization by LC tokens.

7.5 Discussion

In this chapter, we propose a method for linguistic refinement of the cuneiform sign detections produced by the sign detector introduced in Chapter 6. Our two-step approach combines bottom-up visual information from the tablet image with top-down linguistic information about cuneiform script to improve the detection results. The first step consists of line text prediction that groups sign detections in a tablet image (i.e. labeled bounding boxes) according to their line membership and outputs a sequence of sign code classes for each line. The second step consists of a RNN-based text correction model that locates errors in the line text prediction and outputs a corrected version. To train the model in the task of error correction, training samples are generated in a self-supervised fashion by artificially injecting noise in lines of transliterated text. LC tokens are introduced to provide additional bottom-up information to the correction model.

We collect a large corpus of cuneiform script to conduct our experiments. Line text prediction offers a weakly supervised evaluation of the cuneiform sign detector across all SAAo collections with several thousand tablet images - without requiring extra manual annotations. Furthermore, this evaluation enables us to estimate the inherent error of the dataset, in particular the error between tablet image, transliteration and annotations. By estimating the errors, we better understand the current limitations of our approach and can draw more confident conclusions from our results.

We obtain a proof of concept of the proposed linguistic refinement method on synthetic data. The text correction model successfully learns an internal language model for cuneiform script that even generalizes to unseen data samples. Unfortunately, the text correction model does not perform well on real data. We assume that the following two reasons are central to this problem: 1) There is a obvious gap between synthetic and real data. While the gap can be reduced by mixing real and synthetic data during training, the text correction model usually fails to improve upon the line text prediction baseline on the real test set. 2) Most combination of cuneiform signs are very rare in the dataset which is visible in the long tail of sign combinations in Fig. 7.3. Thus, learning a good language model seems only feasible for the more frequent sign combinations, given the constraints of the dataset. This hypothesis is supported by the evaluation of the text correction model on n-gram data in Sect. 7.4.4, where the model outperforms the baseline only in case of frequent n-grams.

To improve how the text correction model performs on real data, the difference between synthetic and real data needs to be reduced during the learning process. Therefore, the self-supervised training employs synthetic noise that approximates the noise found in real data. However, some important differences remain: 1) While the synthetic noise is distributed uniformly across the generated samples, the real noise is often focused on rare sign code classes (which the sign detector struggles with). Thus real data is also more challenging to fix, since there are fewer examples to train the correction model. 2) Real noise is not uniformly distributed across the text corpus, i.e. errors accumulate in specific locations and the composition of errors varies between lines and tablets significantly.

The text correction model benefits from the incorporation of additional bottom-up visual information. In particular, our approach employs LC tokens to leverage the detection confidence to indicate potential errors. However, there is more visible information that can be included. For example, the horizontal distance between sign detections could be encoded with additional tokens or by relying on techniques like positional encoding [216, 225]. Besides the information content, it is important to consider its reliability as we have seen in the case of LC tokens in our experiments. An interesting extension of the proposed approach is to train cuneiform sign detection and text correction as a single neural network in an end-to-end fashion. In this way, the network can learn a representation that contains the most useful visual information for text correction.

At the core of the text correction model is the language model for cuneiform script. There are several options to boost the performance of the language model: The collection of additional training data in the form of transliterations of Neo-Assyrian cuneiform script can improve performance for rare sign combinations. Another option is transfer learning, which is widely used in NLP domain like in the case of machine translation for low-resource languages [144]. It can be beneficial to build on pre-trained language models, even if their language is only remotely related to Akkadian and Sumerian.

The proposed method for linguistic refinement allows the automatic generation of a cuneified transliteration, which is helpful for Assyriologists. The text correction model operates on character-level (cuneiform signs) and thus its internal language model only learns language implicitly. A full transliteration contains the linguistic interpretation of the written text in an explicit form (grammar, syntax, semantics) and thus resolves many ambiguities of the cuneified transliteration that arise from the special characteristics of cuneiform writing system (cf. Sect. 4.2.2). As the linguistic context compared to writing system context is more structured, the training of a language model that bridges the gap between sign code classes and full transliterations might be more data-efficient and thus a promising target for future work.

CHAPTER 8

Conclusion

In this thesis, computer vision algorithms for visual recognition have been developed that are able to leverage the power of deep learning for tasks with limited supervision. To compensate for the lack of supervision, we have employed the general idea of wrapping the supervised learning of a deep network inside an iterative learning method to solve a particular task. In each iteration, this method generates annotations from limited supervision and in turn trains the deep network. During the learning process, the method not only optimizes the sign detector, but simultaneously increases the quality and quantity of the generated sign annotations. To this end, we have developed learning procedures and models for two computer vision applications with limited supervision.

For the first application of human pose analysis, we have proposed a self-supervised approach to learn a representation that captures the characteristics of human posture. Therefore, we have designed two complementary self-supervised tasks, temporal ordering and spatial placement, which automatically generate the supervision necessary for learning a representation from unlabeled video data. To facilitate self-supervised training, we have embraced the idea of iterative learning and introduced a motion-based curriculum as well as a procedure to mine and utilize repetitive poses as valuable training data. We have demonstrated in experiments on several datasets that our method learns a representation of human pose that is competitive with the comparable methods in the literature.

For the second application of cuneiform sign detection, we have proposed a weakly supervised approach to train a sign detector that locates and classifies cuneiform signs in images of clay tablets. Our approach generates sign annotations for training the sign detector by using weak supervision from transliterations. In each iteration, our iterative learning method alternates between training a sign detector and generating sign annotations by solving the alignment problem between tablet image and transliteration. Besides optimizing the sign detector, iterative learning aims to improve the precision and recall of the generated sign annotations. For this purpose, we divide the generation of sign annotations into two distinct steps, sign placement and image transliteration alignment. The two steps provide difficult (exploration) and reliable (exploitation) training samples for a balanced sign detector training. While our method works weakly supervised, a small number of annotations further boost the performance of the cuneiform sign detector. For the training and evaluation of our approach, we have created the first large-scale dataset for cuneiform sign detection. Its several thousand sign annotations provide a rich benchmark for fine-grained sign detection that can facilitate further research into cuneiform script. Moreover, a linguistic refinement method has been proposed that trains a text correction model in a self-supervised fashion to further improve the detection results of the cuneiform sign detector. The self-supervised task consists of reconstructing synthetically corrupted text sequences from transliteration of cuneiform script. The correction model is trained to effectively combine bottom-up information from the sign detections and top-down language information from several thousand lines of cuneiform script. The proposed method demonstrates the first steps towards the automatic transliteration of cuneiform text. To enable experts to directly apply the sign detector in their study of cuneiform texts, we have additionally provided a web application for the analysis of clay tablets with a trained cuneiform sign detector. Since our method is applicable to cuneiform script of different periods, it enables large-scale digital analysis of cuneiform script in the field of Assyriology, thus helping to more efficiently open up the treasures of human civilization. We have explored the boundaries of weakly supervised learning for computer vision while facilitating the difficult analysis of complex script to directly support research based on ancient written texts for which existing text recognition methods are not applicable.

Interesting Findings for Learning with Limited Supervision

Although the considered applications are quite different in their task and limited supervision, we have found that different learning scenarios are often dealing with similar problems. The general idea is to guide the learning algorithm towards the desired outcome by integrating alternative forms of supervision for the task. Extending the supervised learning procedure with a supervision generation step provides a new level of control over the training data for the learning algorithm (accumulated experience). Designing self-supervised tasks as in Chapter 3 or leveraging weak supervision as in Chapter 6 not merely replaces supervision. Instead, it offers a great opportunity to boost the overall outcome by integrating additional task-specific prior knowledge and expanding the scope of data used for learning. Such a broader learning experience is especially helpful because deep networks tend to learn cheap shortcuts [76, 77], such as a preference for texture instead of an universal representation for visual objects. When applying iterative learning to tasks with limited supervision, it is important to consider the optimization of the model as well as the optimization of the generated supervised data. Thus, learning with limited supervision resembles an alternating optimization, where the dynamics of the learning process have to be taken into consideration. In the case of the two considered applications, we have identified the main problems associated with such an iterative learning procedure and devised several strategies to overcome them, which are summarized below:

Learning with Noisy Supervision The generated supervised data exhibits a high degree of noise, e.g. annotations are erroneous, incomplete, or ambiguous, beyond the inconsistencies found in manually supervised data. Fortunately, deep learning is very efficient at filtering the signal from the noise and is able to learn a useful representation that surpasses the consistency of the training data. The effect can be amplified by choosing an appropriate deep architecture that matches the complexity of the task and by applying regularization techniques to mitigate the risk of overfitting. However, deep learning alone is not sufficient to deal with the side effects of iterative learning, such as drift and imbalanced supervision.

Self-Correction for Supervision Generation The quality of the generated supervised data matters. We have found it essential to implement a self-correction method that exploits task-specific prior knowledge to refine the generated supervision, before it is used to train the model. For this purpose, the bottom-up information by the trained model must be verified by some top-down information such as task-specific assumptions or weak supervision at each iteration of supervision generation. In the case of learning pose embeddings, we have exploited the self-similarity of poses under different camera angles and the speed of motion in video data to control the quality of the generated supervision. In the case of learning a cuneiform sign detection, we have combined in an alignment method the geometric structure of lines in clay tablets together with the weak supervision from transliterations.

Balancing Exploration and Exploitation The diversity of the generated supervised data matters as well. Because data is often imbalanced in terms of certain concepts, it is important to deal with the long tail of the data distribution. By making supervised data more diverse, we enable the learning of new and difficult concepts from this long tail. Thus we have devised methods to identify and explore difficult examples for learning, such as a curriculum for learning pose embeddings in Chapter 3 and a sign placement step for training the sign detector in Chapter 6 However, too much emphasis on diverse data risks introducing many additional errors to the training data. As a general rule for generating supervised data, it is important to find a good balance between generating reliable data (exploitation) and generating diverse data (exploration) for learning. This presents an interesting analogy to the field of reinforcement learning [202], where the trade-off between exploration and exploitation is equally important. The generated supervised data is comparable to the experience an agent gains by interacting with its environment in order to accomplish a goal. While interested in solving a particular task, the agent has to discover a series of intermediate steps that must be learned to accomplish the general objective. The agent generates its own data by taking actions (simulate). In the case of iterative learning, we similarly have to pay attention to the learning curve (experience) which is shaped by the supervised data generated from self-supervised tasks or bridging the gap from weak to full supervision.

Dealing with Rare Classes The generation of supervised data cannot generate arbitrary data samples, but is fed from an existing dataset with limited supervision originally sampled from the data distribution. Learning a representation to capture fine-grained details of rare classes from the long tail of the data distribution can be very inefficient, since a large dataset may only yield a few suitable training samples. If the supervision generation process is very noisy, the training samples might even be just caused by noise. In the case of our weakly-supervised approach to learn cuneiform sign detection, this makes the learning of some rare sign code classes infeasible. However, we have demonstrated in experiments in Chapter 6 that iterative learning with limited supervision is very annotation-efficient and that a few manual sign annotations are sufficient to mitigate this problem.

List of Publications

Parts of this thesis have been published in the following scientific articles:

- Ö. Sümer, T. Dencker, and B. Ommer. "Self-supervised Learning of Pose Embeddings from Spatiotemporal Relations in Videos". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 4298–4307.
- T. Dencker, P. Klinkisch, S. M. Maul, and B. Ommer. "Deep learning of cuneiform sign detection with weak supervision using transliteration alignment". *PLOS ONE* 15:12, 2020, pp. 1–21. URL: https://doi.org/10.1371/journal.pone.0243039.

The majority of ideas, texts, figures, and experiments are by the first author, with the exception of the first paper on *self-supervised learning*, to which the first two authors contributed equally. All other authors had important advisory roles, helped to perform experiments, and/or contributed directly in revising a small number of individual sections of the articles listed above.

List of Figures

1.1 Examples of computer vision tasks 3

2.1 Components of a machine learning algorithm and the associated development phases 17
2.2 Design process of machine learning algorithm 20
2.3 Generating supervised data for iterative learning with limited supervision . 29
2.4 Learning approaches for limited supervision 30

3.1 Sampling procedure for training self-supervised pose embeddings 38
3.2 Mining repetitive poses 41
3.3 Network architecture for temporal ordering and spatial placement 42
3.4 Histogram of the curriculum used for training on Olympic Sports dataset . 43
3.5 Groups of repetitive poses mined from a single video 44
3.6 Groups of repetitive poses for two sport categories. 45
3.7 Pose retrieval results on MPII validation set 48
3.8 Pose retrievals for four query images on the MPII Human Pose dataset . . . 49
3.9 Qualitative pose estimation results in Leeds Sports Pose dataset 50

4.1 Drawing of the three types of wedges used in Neo-Assyrian cuneiform . . . 54
4.2 Challenges of reading cuneiform script 56
4.3 Excerpt of a cuneiform sign list based on the MZL by Borger 56
4.4 Standard process of Assyriology to study a cuneiform tablet 57
4.5 The Assyriological process of analyzing a cuneiform clay tablet 58
4.6 The problem of fine-grained sign similarity in cuneiform script 59

5.1 Data preparation pipeline for the cuneiform sign detection dataset 63
5.2 Composite image of clay tablet and corresponding transliteration 64
5.3 Composite image of clay tablet with segmented views 65
5.4 Heuristics for the assignment of sides of transliteration to segmented tablet views 67
5.5 Failure cases of automatic assignment 67
5.6 Manual annotation of lines and cuneiform signs in images of clay tablets . . 68
5.7 Screenshot of web application for the annotation of cuneiform signs 70

5.8 Frequency distribution of segmented views with respect to two properties . 74
5.9 Unicode characters of sign code classes considered for detector training . . . 76
5.10 Plot of the frequency of the 186 sign code classes in Train TL set 76

6.1 Overview of weakly supervised learning approach for cuneiform sign detection 78
6.2 Weakly supervised training of sign detector comprises an iterative training loop with three steps 81
6.3 Sampling of training data for line segmentation 87
6.4 Examples of training data for the line segmentation network 87
6.5 Postprocessing of line segmentation to obtain detected lines 88
6.6 Score matrices and longest paths of the line-level alignment with different score functions 91
6.7 The CRF has a fully-connected graph which models appearance and geometric constraints of a line 93
6.8 Line-level alignment produces an assignment of transliteration lines to detected lines 97
6.9 Effect of alignment and placement method on quality of detections 98
6.10 Sign-level alignment for specific tablet 99
6.11 Effect of iterative training on ratio of generated detection types and detection performance 100
6.12 Evolution of placed and aligned detections during iterative training 101
6.13 Effect of manual sign annotations on detection performance comparing purely supervised and our iterative training 102
6.14 Individual sign detections for single sign code class at two different iterations of iterative training 103
6.15 Comparison of two ways to train a Old-Babylonian sign detector with little supervised data 104
6.16 Class-wise detection results with average precision of every second sign code class in test set 106
6.17 Scatter plots for studying correlation between detection performance and data frequency 106
6.18 False positive error analysis of sign detector on test set 107
6.19 Class-wise detection results showing sign code classes organized in eight groups of very similar classes 108
6.20 Confusion matrix of 36 sign code classes from eight groups of often confused sign code classes 108
6.21 True positive detections of two different cuneiform sign code classes 109
6.22 Detection results on twelve tablet images with clear and damaged cuneiform script 111
6.23 Image crops from four clay tablets illustrating the change of detection results during training 112
6.24 Evaluation of different backbone architectures and their initialization on the task of cuneiform sign code classification 113
6.25 Usage of web application for cuneiform sign detection 114

7.1 Overview of the approach for linguistic refinement of the sign detections . . 121
7.2 Example of LC token usage 124

7.3 Frequency distribution of 1 to 6-grams found in the transliterations of the cuneiform dataset . . . 126
7.4 The top-fifteen 1 to 6-grams extracted from the transliterations of the cuneiform dataset . . . 127
7.5 Line text prediction of four lines of cuneiform script found on different clay tablets . . . 129
7.6 Evaluation of line text prediction performance across all collections of the SAAo dataset . . . 131
7.7 Triangle of discrepancies between annotations, tablet image and transliteration 132
7.8 Visualization of GloVe word embeddings for cuneiform signs using the t-SNE method . . . 134
7.9 Error composition of all synthetic data configurations used for evaluation . 136
7.10 Five synthetic data configurations for evaluation of line text correction . . . 136
7.11 Evaluation of line correction model on synthetic data . . . 137
7.12 Evaluation of line correction model on real detection data . . . 138
7.13 Evaluation of line text correction on synthetic 5-gram data . . . 140

List of Tables

3.1 Average AUC in Olympic Sports benchmark shows effect of curriculum training . . . 46
3.2 Positive and negative sampling ranges in temporal ordering task . . . 46
3.3 Comparative posture analysis performance of auxiliary tasks in Olympic Sports dataset . . . 47
3.4 Pose estimation results in Leeds Sports Pose dataset . . . 50
3.5 Comparing different initialization approaches on MPII Pose benchmark dataset 51

5.1 Aggregated statistics of SAAo after performing data collection, preparation and annotation . . . 72
5.2 Statistics of SAAo collections after performing data collection, preparation and annotation . . . 72
5.3 Collection sources of the two train sets and the test set . . . 73
5.4 Composition of the two train sets and the test set . . . 73
5.5 Statistics on data collection, preparation and annotation for legacy collections 74

6.1 The configuration of λ and σ parameters for the sign-level alignment . . . 95
6.2 Performance of different approaches to line-level alignment on test set . . . 96
6.3 Performance of different score functions for line-level alignment on test set . 96

7.1 The three types of text prediction errors . . . 122
7.2 Statistics of the TL dataset compared to the Train-TL dataset . . . 125
7.3 Train and test split of SAAo collections for line text correction . . . 135

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. “Tensorflow: A system for large-scale machine learning”. In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016, pp. 265–283.

[2] R. G. Adams. *A Translation of the Rosetta Stone*. Printed at the Harvard University Press, 1925.

[3] P. Agrawal, J. Carreira, and J. Malik. “Learning to See by Moving”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 37–45. DOI: 10.1109/ICCV.2015.13.

[4] B. Alexe, T. Deselaers, and V. Ferrari. “Measuring the objectness of image windows”. *IEEE transactions on pattern analysis and machine intelligence* 34:11, 2012, pp. 2189–2202.

[5] G. F. Altorientalistik. “Der Umfang des keilschriftlichen Textkorpus'”. *Mitteilungen der Deutschen Orient-Gesellschaft zu Berlin* 42, 2010, p. 201.

[6] C. Amrhein and S. Clematide. “Supervised ocr error detection and correction using statistical and neural machine translation methods”. *Journal for Language Technology and Computational Linguistics (JLCL)* 33:1, 2018, pp. 49–76.

[7] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang. “Bottom-up and top-down attention for image captioning and visual question answering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6077–6086.

[8] S. E. Anderson and M. Levoy. “Unwrapping and visualizing cuneiform tablets”. *IEEE Computer Graphics and Applications* 22:6, 2002, pp. 82–88.

[9] B. Andres, T. Beier, and J. H. Kappes. “OpenGM: A C++ library for discrete graphical models”. *arXiv preprint arXiv:1206.0111*, 2012.

[10] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele. “2D Human Pose Estimation: New Benchmark and State of the Art Analysis”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3686–3693. DOI: 10.1109/CVPR.2014.471.

[11] B. Antic and B. Ommer. “Per-Sample Kernel Adaptation for Visual Recognition and Grouping”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1251–1259. DOI: 10.1109/ICCV.2015.148.

[12] B. Antic and B. Ommer. “Learning latent constituents for recognition of group activities in video”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 33–47.

[13] Y. M. Asano, C. Rupprecht, and A. Vedaldi. “A critical analysis of self-supervision, or what we can learn from a single image”. *arXiv preprint arXiv:1904.13132*, 2019.

[14] Y. M. Asano, C. Rupprecht, and A. Vedaldi. “Self-labelling via simultaneous clustering and representation learning”. *arXiv preprint arXiv:1911.05371*, 2019.

[15] J. L. Ba, J. R. Kiros, and G. E. Hinton. “Layer normalization”. *arXiv preprint arXiv:1607.06450*, 2016.

[16] D. Bahdanau, K. Cho, and Y. Bengio. “Neural machine translation by jointly learning to align and translate”. *arXiv preprint arXiv:1409.0473*, 2014.

[17] S. Banerjee and A. Lavie. “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments”. In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 2005, pp. 65–72.

[18] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. “Relational inductive biases, deep learning, and graph networks”. *arXiv preprint arXiv:1806.01261*, 2018.

[19] M. A. Bautista, A. Sanakoyeu, E. Tikhoncheva, and B. Ommer. “CliqueCNN: Deep Unsupervised Exemplar Learning”. *Advances in Neural Information Processing Systems* 29, 2016, pp. 3846–3854.

[20] A. Bearman, O. Russakovsky, V. Ferrari, and L. Fei-Fei. “What’s the point: Semantic segmentation with point supervision”. In: *European conference on computer vision*. Springer. 2016, pp. 549–565.

[21] S. Becker and G. E. Hinton. “A self-organizing neural network that discovers surfaces in random-dot stereograms.” *Nature* 355, 1992, pp. 161–163.

[22] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. “Curriculum Learning”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. ACM, Montreal, Quebec, Canada, 2009, pp. 41–48. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553380. URL: http://doi.acm.org/10.1145/1553374.1553380.

[23] T. Berg-Kirkpatrick, G. Durrett, and D. Klein. “Unsupervised transcription of historical documents”. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2013, pp. 207–217.

[24] H. Bilen and A. Vedaldi. “Weakly supervised deep detection networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2846–2854.

[25] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[26] A. Bissacco, M. Cummins, Y. Netzer, and H. Neven. “Photoocr: Reading text in uncontrolled conditions”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 785–792.

[27] B. Bogacz, N. Howe, and H. Mara. “Segmentation free spotting of cuneiform using part structured models”. In: *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE. 2016, pp. 301–306.

[28] B. Bogasz, M. Gertz, and H. Mara. “Cuneiform Character Similarity Using Graph Representations”. In: *Proceedings of the 20th Computer Vision Winter Workshop*. 2015.

[29] R. Borger. *Mesopotamisches Zeichenlexikon*. Alter Orient und Altes Testament. Ugarit-Verl., Münster, 2004, VIII, 712 S. ISBN: 978-3-927120-82-2.

[30] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah. “Signature verification using a “siamese” time delay neural network”. *International Journal of Pattern Recognition and Artificial Intelligence* 7:04, 1993, pp. 669–688.

[31] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. “High accuracy optical flow estimation based on a theory for warping”. In: *ECCV*. Vol. 3024. Lecture Notes in Computer Science. 2004, pp. 25–36. URL: http://lmb.informatik.uni-freiburg.de//Publications/2004/Bro04a.

[32] H. Bunke. “Recognition of cursive Roman handwriting: past, present and future”. In: *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings*. IEEE. 2003, pp. 448–459.

[33] M. Cammarosano. “The cuneiform stylus”. *Mesopotamia* 49, 2014, pp. 53–90.

[34] M. Caron, P. Bojanowski, A. Joulin, and M. Douze. “Deep clustering for unsupervised learning of visual features”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 132–149.

[35] E. Chammas, C. Mokbel, and L. Likforman-Sulem. “Handwriting Recognition of Historical Documents with few labeled data”. In: *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE. 2018, pp. 43–48.

[36] L. Chan, M. S. Hosseini, and K. N. Plataniotis. “A comprehensive analysis of weakly-supervised semantic segmentation in different image domains”. *International Journal of Computer Vision*, 2020, pp. 1–24.

[37] K.-M. Chao, R. C. Hardison, and W. Miller. “Recent developments in linear-space alignment methods: a survey”. *Journal of Computational Biology* 1:4, 1994, pp. 271–291.

[38] D. Charpin. *Reading and writing in Babylon*. Harvard University Press, 2010.

[39] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. “Return of the devil in the details: Delving deep into convolutional nets”. *arXiv preprint arXiv:1405.3531*, 2014.

[40] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs”. *IEEE transactions on pattern analysis and machine intelligence* 40:4, 2017, pp. 834–848.

[41] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. “A simple framework for contrastive learning of visual representations”. *arXiv preprint arXiv:2002.05709*, 2020.

[42] X. Chen and A. L. Yuille. “Articulated pose estimation by a graphical model with image dependent pairwise relations”. *Advances in neural information processing systems* 27, 2014, pp. 1736–1744.

[43] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. *arXiv preprint arXiv:1406.1078*, 2014.

[44] F. Chollet. “On the measure of intelligence”. *arXiv preprint arXiv:1911.01547*, 2019.

[45] A. Chowdhury and L. Vig. “An efficient end-to-end neural model for handwritten text recognition”. *arXiv preprint arXiv:1807.07965*, 2018. eprint: 1807.07965. URL: http://arxiv.org/abs/1807.07965.

[46] X. Chu, W. Ouyang, H. Li, and X. Wang. “Structured Feature Learning for Pose Estimation”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4715–4723. DOI: 10.1109/CVPR.2016.510.

[47] D. Ciregan, U. Meier, and J. Schmidhuber. “Multi-column deep neural networks for image classification”. In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 3642–3649.

[48] J. D. Cohen, D. D. Duncan, D. Snyder, J. Cooper, S. Kumar, D. V. Hahn, Y. Chen, B. Purnomo, and J. Graettinger. “iClay: Digitizing Cuneiform.” In: *VAST*. 2004, pp. 135–143.

[49] *Cuneiform Digital Library Initiative*. http://cdli.ucla.edu. accessed Feb 8, 2018. URL: http://cdli.ucla.edu.

[50] S. Dalley. *Myths from Mesopotamia: creation, the flood, Gilgamesh, and others*. Oxford University Press, USA, 1998.

[51] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars. “Continual learning: A comparative study on how to defy forgetting in classification tasks”. *arXiv preprint arXiv:1909.08383* 2:6, 2019.

[52] N. Demoli, H. Gruber, U. Dahms, and G. Wernicke. “Characterization of the cuneiform signs by the use of a multifunctional optoelectronic device”. *Applied optics* 35:29, 1996, pp. 5811–5820.

[53] T. Dencker, P. Klinkisch, S. M. Maul, and B. Ommer. “Deep learning of cuneiform sign detection with weak supervision using transliteration alignment”. *PLOS ONE* 15:12, 2020, pp. 1–21. URL: https://doi.org/10.1371/journal.pone.0243039.

[54] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[55] Y. Deng, A. Kanervisto, J. Ling, and A. M. Rush. “Image-to-markup generation with coarse-to-fine attention”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 980–989.

[56] S. Dodge and L. Karam. “A study and comparison of human and deep learning recognition performance under visual distortions”. In: *2017 26th international conference on computer communication and networks (ICCCN)*. IEEE. 2017, pp. 1–7.

[57] C. Doersch, A. Gupta, and A. A. Efros. “Unsupervised Visual Representation Learning by Context Prediction”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1422–1430. DOI: 10.1109/ICCV.2015.167.

[58] R. Dong and D. A. Smith. “Multi-input attention for unsupervised OCR correction”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 2363–2372.

[59] A. Dosovitskiy and T. Brox. “Inverting visual representations with convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4829–4837.

[60] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian. “Centernet: Keypoint triplets for object detection”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 6569–6578.

[61] R. O. Duda and P. E. Hart. “Use of the Hough transformation to detect lines and curves in pictures”. *Communications of the ACM* 15:1, 1972, pp. 11–15.

[62] N. M. Edan. “Cuneiform symbols recognition based on k-means and neural network”. *AL-Rafidain Journal of Computer Sciences and Mathematics* 10:1, 2013, pp. 195–202.

[63] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. “The Pascal Visual Object Classes (VOC) Challenge”. *International Journal of Computer Vision* 88:2, 2010, pp. 303–338.

[64] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. “Object Detection with Discriminatively Trained Part-Based Models”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32:9, 2010, pp. 1627–1645. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2009.167.

[65] P. F. Felzenszwalb and D. P. Huttenlocher. “Efficient graph-based image segmentation”. *International journal of computer vision* 59:2, 2004, pp. 167–181.

[66] P. F. Felzenszwalb and D. P. Huttenlocher. “Pictorial structures for object recognition”. *International journal of computer vision* 61:1, 2005, pp. 55–79.

[67] J. Feng, Y. Wei, L. Tao, C. Zhang, and J. Sun. “Salient object detection by composition”. In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 1028–1035.

[68] V. Ferrari, M. Marin-Jimenez, and A. Zisserman. “Progressive search space reduction for human pose estimation”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587468.

[69] I. L. Finkel and J. Taylor. *Cuneiform*. British Museum, 2015.

[70] C. Finn, P. Abbeel, and S. Levine. “Model-agnostic meta-learning for fast adaptation of deep networks”. *arXiv preprint arXiv:1703.03400*, 2017.

[71] D. Fisseler, F. Weichert, G. Müller, and M. Cammarosano. “Towards an interactive and automated script feature analysis of 3D scanned cuneiform tablets”. *Scientific Computing and Cultural Heritage*, 2013, p. 16.

[72] M. Freitag and Y. Al-Onaizan. “Beam search strategies for neural machine translation”. *arXiv preprint arXiv:1702.01806*, 2017.

[73] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning.* Vol. 1. 10. Springer series in statistics New York, 2001.

[74] K. Fukushima and S. Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets.* Springer, 1982, pp. 267–285.

[75] R. Ge, F. Huang, C. Jin, and Y. Yuan. "Escaping from saddle points—online stochastic gradient for tensor decomposition". In: *Conference on Learning Theory.* 2015, pp. 797–842.

[76] R. Geirhos, J. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann. "Shortcut Learning in Deep Neural Networks". *arXiv preprint arXiv:2004.07780*, 2020.

[77] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness". *arXiv preprint arXiv:1811.12231*, 2018.

[78] I. J. Gelb and R. Whiting. "Methods of decipherment". *Journal of the Royal Asiatic Society* 107:2, 1975, pp. 95–104.

[79] A. George. *Old Babylonian Texts in the Schøyen Collection, Part One: Selected Letters. (Cornell University Studies in Assyriology and Sumerology 36).* CDL Press, 2017.

[80] R. Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision.* 2015, pp. 1440–1448.

[81] R. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2014, pp. 580–587.

[82] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics.* 2010, pp. 249–256.

[83] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning.* Vol. 1. 2. MIT press Cambridge, 2016.

[84] R. Goroshin, J. Bruna, J. Tompson, D. Eigen, and Y. LeCun. "Unsupervised Learning of Spatiotemporally Coherent Metrics". In: *2015 IEEE International Conference on Computer Vision (ICCV).* 2015, pp. 4086–4093. DOI: `10.1109/ICCV.2015.465`.

[85] A. Graves. "Generating sequences with recurrent neural networks". *arXiv preprint arXiv:1308.0850*, 2013.

[86] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. "A novel connectionist system for unconstrained handwriting recognition". *IEEE transactions on pattern analysis and machine intelligence* 31:5, 2008, pp. 855–868.

[87] A. Gupta, A. Vedaldi, and A. Zisserman. "Learning to read by spelling: Towards unsupervised text recognition". *arXiv preprint arXiv:1809.08675*, 2018.

[88] A. Gupta, A. Vedaldi, and A. Zisserman. "Synthetic data for text localisation in natural images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 2315–2324.

[89] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. “Momentum contrast for unsupervised visual representation learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9729–9738.

[90] K. He, R. Girshick, and P. Dollár. “Rethinking imagenet pre-training”. In: *Proceedings of the IEEE international conference on computer vision*. 2019, pp. 4918–4927.

[91] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[92] K. He, X. Zhang, S. Ren, and J. Sun. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[93] X. He, K. Zhao, and X. Chu. “AutoML: A Survey of the State-of-the-Art”. *arXiv preprint arXiv:1908.00709*, 2019.

[94] G. Hinton. *Taking inverse graphics seriously*. `http://www.csri.utoronto.ca/~hinton/csc2535/notes/lec6b.pdf`. Accessed: 2020–11-05. 2013.

[95] G. E. Hinton and R. S. Zemel. “Autoencoders, minimum description length and Helmholtz free energy”. In: *Advances in neural information processing systems*. 1994, pp. 3–10.

[96] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. *Neural computation* 9:8, 1997, pp. 1735–1780.

[97] E. Hoffer, I. Hubara, and D. Soudry. “Train longer, generalize better: closing the generalization gap in large batch training of neural networks”. In: *Advances in neural information processing systems*. 2017, pp. 1731–1741.

[98] D. Hoiem, Y. Chodpathumwan, and Q. Dai. “Diagnosing Error in Object Detectors”. In: *Proceedings of the 12th European Conference on Computer Vision - Volume Part III*. ECCV’12. Springer-Verlag, Florence, Italy, 2012, pp. 340–353. ISBN: 978-3-642-33711-6. DOI: `10.1007/978-3-642-33712-3_25`. URL: `http://dx.doi.org/10.1007/978-3-642-33712-3_25`.

[99] T. Homburg and C. Chiarcos. “Word segmentation for akkadian cuneiform”. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC’16)*. 2016, pp. 4067–4074.

[100] K. Hornik, M. Stinchcombe, and H. White. “Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks”. *Neural networks* 3:5, 1990, pp. 551–560.

[101] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. “Meta-learning in neural networks: A survey”. *arXiv preprint arXiv:2004.05439*, 2020.

[102] H. Hu, C. Zhang, Y. Luo, Y. Wang, J. Han, and E. Ding. “WordSup: Exploiting word annotations for character based text detection”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 4940–4949.

[103] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. “Speed/accuracy trade-offs for modern convolutional object detectors”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7310–7311.

[104] D. H. Hubel and T. N. Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. *The Journal of physiology* 160:1, 1962, p. 106.

[105] F. Hutmacher. “Why is there so much more research on vision than on any other sensory modality?” *Frontiers in psychology* 10, 2019, p. 2246.

[106] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size”. *arXiv preprint arXiv:1602.07360*, 2016.

[107] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. *arXiv preprint arXiv:1502.03167*, 2015.

[108] M. Jaderberg, A. Vedaldi, and A. Zisserman. “Deep features for text spotting”. In: *European conference on computer vision*. Springer. 2014, pp. 512–528.

[109] D. Jayaraman and K. Grauman. “Slow and Steady Feature Analysis: Higher Order Temporal Coherence in Video”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 3852–3861. DOI: 10.1109/CVPR.2016.418.

[110] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. “Caffe: Convolutional Architecture for Fast Feature Embedding”. *arXiv preprint arXiv:1408.5093*, 2014.

[111] L. Jing and Y. Tian. “Self-supervised visual feature learning with deep neural networks: A survey”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[112] S. Johnson and M. Everingham. “Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation”. In: *Proceedings of the British Machine Vision Conference*. doi:10.5244/C.24.12. 2010.

[113] E. R. Kandel, J. H. Schwartz, T. M. Jessell, D. of Biochemistry, M. B. T. Jessell, S. Siegelbaum, and A. Hudspeth. *Principles of neural science*. Vol. 4. McGraw-hill New York, 2000.

[114] T. N. Kipf and M. Welling. “Semi-supervised classification with graph convolutional networks”. *arXiv preprint arXiv:1609.02907*, 2016.

[115] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[116] V. Kolmogorov. “Convergent tree-reweighted message passing for energy minimization”. *IEEE transactions on pattern analysis and machine intelligence* 28:10, 2006, pp. 1568–1583.

[117] S. Kornblith, J. Shlens, and Q. V. Le. “Do better imagenet models transfer better?” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 2661–2671.

[118] N. M. Kriege, M. Fey, D. Fisseler, P. Mutzel, and F. Weichert. "Recognizing cuneiform signs using graph based methods". *arXiv preprint arXiv:1802.05908*, 2018.

[119] A. Krishna, B. P. Majumder, R. S. Bhat, and P. Goyal. "Upcycle your ocr: Reusing ocrs for post-ocr text correction in romanised sanskrit". *arXiv preprint arXiv:1809.02147*, 2018.

[120] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. 2012, pp. 1097–1105.

[121] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[122] J. Kukačka, V. Golkov, and D. Cremers. "Regularization for deep learning: A taxonomy". *arXiv preprint arXiv:1710.10686*, 2017.

[123] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. "Deep convolutional inverse graphics network". *Advances in neural information processing systems* 28, 2015, pp. 2539–2547.

[124] S. Kwak, M. Cho, and I. Laptev. "Thin-Slicing for Pose: Learning to Understand Pose without Explicit Pose Estimation". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4938–4947.

[125] *Learning to read by reading. self-supervised learning of cuneiform script*. eng. Heidelberg, 2017, 73 Seiten.

[126] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation applied to handwritten zip code recognition". *Neural computation* 1:4, 1989, pp. 541–551.

[127] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". *Proceedings of the IEEE* 86:11, 1998, pp. 2278–2324.

[128] C.-Y. Lee and S. Osindero. "Recursive recurrent nets with attention modeling for ocr in the wild". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2231–2239.

[129] D.-H. Lee. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks". In: *Workshop on challenges in representation learning, ICML*. Vol. 3. 2. 2013.

[130] F.-F. Li, R. Krishna, and D. Xu. *CS231n: Convolutional Neural Networks for Visual Recognition*. http://cs231n.stanford.edu/slides/2020/lecture_12.pdf. Accessed: 2020–12-05. 2020.

[131] Z. Li and D. Hoiem. "Learning without forgetting". *IEEE transactions on pattern analysis and machine intelligence* 40:12, 2017, pp. 2935–2947.

[132] C.-Y. Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out*. 2004, pp. 74–81.

[133] G. Lin, C. Shen, A. Van Den Hengel, and I. Reid. "Exploring context with deep structured models for semantic segmentation". *IEEE transactions on pattern analysis and machine intelligence* 40:6, 2017, pp. 1352–1366.

[134] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. "Feature pyramid networks for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 2117–2125.

[135] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision.* 2017, pp. 2980–2988.

[136] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. "Progressive neural architecture search". In: *Proceedings of the European Conference on Computer Vision (ECCV).* 2018, pp. 19–34.

[137] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. "SSD: Single shot multibox detector". In: *European conference on computer vision.* Springer. 2016, pp. 21–37.

[138] Y. Liu, Z. Wang, H. Jin, and I. Wassell. "Synthetically supervised feature learning for scene text recognition". In: *Proceedings of the European Conference on Computer Vision (ECCV).* 2018, pp. 435–451.

[139] Y. Liu and L. Jin. "Deep matching prior network: Toward tighter multi-oriented text detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2017, pp. 1962–1969.

[140] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 3431–3440.

[141] J. L. Long, N. Zhang, and T. Darrell. "Do convnets learn correspondence?" *Advances in neural information processing systems* 27, 2014, pp. 1601–1609.

[142] M.-T. Luong, H. Pham, and C. D. Manning. "Effective approaches to attention-based neural machine translation". *arXiv preprint arXiv:1508.04025*, 2015.

[143] L. v. d. Maaten and G. Hinton. "Visualizing data using t-SNE". *Journal of machine learning research* 9:Nov, 2008, pp. 2579–2605.

[144] A. Magueresse, V. Carles, and E. Heetderks. "Low-resource Languages: A Review of Past Work and Future Challenges". *arXiv preprint arXiv:2006.07264*, 2020.

[145] A. Mahendran and A. Vedaldi. "Understanding deep image representations by inverting them". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 5188–5196.

[146] H. Mara and S. Krömker. "Vectorization of 3d-characters by integral invariant filtering of high-resolution triangular meshes". In: *2013 12th International Conference on Document Analysis and Recognition.* IEEE. 2013, pp. 62–66.

[147] S. M. Maul. *Das Gilgamesch-Epos.* CH Beck, 2005.

[148] J. Michael, R. Labahn, T. Grüning, and J. Zöllner. "Evaluating sequence-to-sequence models for handwritten text recognition". In: *2019 International Conference on Document Analysis and Recognition (ICDAR).* IEEE. 2019, pp. 1286–1293.

[149] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems.* 2013, pp. 3111–3119.

[150] T. Mikolov, W.-t. Yih, and G. Zweig. “Linguistic regularities in continuous space word representations”. In: *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*. 2013, pp. 746–751.

[151] A. Mishra, K. Alahari, and C. Jawahar. “Scene text recognition using higher order language priors”. In: 2012.

[152] A. Mishra, K. Alahari, and C. Jawahar. “Top-down and bottom-up cues for scene text recognition”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 2687–2694.

[153] I. Misra, C. L. Zitnick, and M. Hebert. “Shuffle and learn: unsupervised learning using temporal order verification”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 527–544.

[154] T. M. Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research ..., 1980.

[155] G. Mori, C. Pantofaru, N. Kothari, T. Leung, G. Toderici, A. Toshev, and W. Yang. “Pose Embeddings: A Deep Architecture for Learning to Match Human Poses”. *arXiv preprint arXiv:1507.00302*, 2015.

[156] A. Newell, K. Yang, and J. Deng. “Stacked hourglass networks for human pose estimation”. In: *European conference on computer vision*. Springer. 2016, pp. 483–499.

[157] J. C. Niebles, C.-W. Chen, and L. Fei-Fei. “Modeling temporal structure of decomposable motion segments for activity classification”. In: *European conference on computer vision*. Springer. 2010, pp. 392–405.

[158] M. Noroozi and P. Favaro. “Unsupervised learning of visual representations by solving jigsaw puzzles”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 69–84.

[159] M. Noroozi, A. Vinjimoor, P. Favaro, and H. Pirsiavash. “Boosting self-supervised learning via knowledge transfer”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9359–9367.

[160] S. J. Oh, R. Benenson, A. Khoreva, Z. Akata, M. Fritz, and B. Schiele. “Exploiting saliency for object segmentation from image level labels”. In: *2017 IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE. 2017, pp. 5038–5047.

[161] *ORACC - SAAo*. http://oracc.org/saao/. accessed Nov 9, 2018. URL: http://oracc.org/saao/.

[162] E. Pagé-Perron, M. Sukhareva, I. Khait, and C. Chiarcos. “Machine translation and automated analysis of the sumerian language”. In: *Proceedings of the Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*. 2017, pp. 10–16.

[163] S. E. Palmer. *Vision science: Photons to phenomenology*. MIT press, 1999.

[164] D. P. Papadopoulos, J. R. Uijlings, F. Keller, and V. Ferrari. “We don’t need no bounding-boxes: Training object class detectors using only human verification”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 854–863.

[165] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. “BLEU: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318.

[166] R. B. Parkinson, W. Diffie, M. Fischer, and R. Simpson. *Cracking codes: the Rosetta Stone and decipherment*. Univ of California Press, 1999.

[167] S. Parpola, K. Watanabe, A. Livingstone, I. Starr, G. B. Lanfranchi, T. Kwasman, F. M. Fales, J. N. Postgate, H. Hunger, L. Kataja, R. Whiting, S. W. Cole, P. Machinist, A. Fuchs, R. Mattila, M. Luukko, G. Van Buylaere, M. Dietrich, and F. S. Reynolds. *State archives of Assyria*. Vol. I-XX. Helsinki University Press, Helsinki, 1987-2015. URL: http://assyriologia.fi/natcp/saa/.

[168] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems*. 2019, pp. 8026–8037.

[169] J. Pennington, R. Socher, and C. D. Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[170] J. Peters, D. Janzing, and B. Schölkopf. *Elements of causal inference*. The MIT Press, 2017.

[171] D. C. Plaut et al. “Experiments on Learning by Back Propagation.”, 1986.

[172] O. Press and L. Wolf. “Using the output embedding to improve language models”. *arXiv preprint arXiv:1608.05859*, 2016.

[173] A. Radford, L. Metz, and S. Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. *CoRR* abs/1511.06434, 2015. URL: http://arxiv.org/abs/1511.06434.

[174] K. Radner and E. Robson. *The Oxford handbook of cuneiform culture*. Oxford University Press, 2011.

[175] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[176] S. Ren, K. He, R. Girshick, and J. Sun. “Faster r-cnn: Towards real-time object detection with region proposal networks”. *IEEE transactions on pattern analysis and machine intelligence* 39:6, 2016, pp. 1137–1149.

[177] Z. Ren, Z. Yu, X. Yang, M.-Y. Liu, Y. J. Lee, A. G. Schwing, and J. Kautz. “Instance-Aware, Context-Focused, and Memory-Efficient Weakly Supervised Object Detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10598–10607.

[178] D. Rolnick, A. Veit, S. Belongie, and N. Shavit. “Deep learning is robust to massive label noise”. *arXiv preprint arXiv:1705.10694*, 2017.

[179] O. Ronneberger, P. Fischer, and T. Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[180] F. Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[181] L. Rothacker, D. Fisseler, G. G. Müller, F. Weichert, and G. A. Fink. "Retrieving cuneiform structures in a segmentation-free word spotting framework". In: *Proceedings of the 3rd International Workshop on Historical Document Imaging and Processing.* ACM. 2015, pp. 129–136.

[182] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". *nature* 323:6088, 1986, pp. 533–536.

[183] E. Rusakov, K. Brandenbusch, D. Fisseler, T. Somel, G. A. Fink, F. Weichert, and G. G. Müller. "Generating Cuneiform Signs with Cycle-Consistent Adversarial Networks". In: *Proceedings of the 5th International Workshop on Historical Document Imaging and Processing.* 2019, pp. 19–24.

[184] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. "Imagenet large scale visual recognition challenge". *International journal of computer vision* 115:3, 2015, pp. 211–252.

[185] O. Russakovsky, L.-J. Li, and L. Fei-Fei. "Best of both worlds: human-machine collaboration for object annotation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 2121–2131.

[186] D. Saad. "Online algorithms and stochastic approximations". *Online Learning* 5, 1998, pp. 6–3.

[187] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. "MobileNetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 4510–4520.

[188] A. M. Saxe, J. L. McClelland, and S. Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". *arXiv preprint arXiv:1312.6120*, 2013.

[189] J. Schmidhuber. "Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook". PhD thesis. Technische Universität München, 1987.

[190] C. Schnober, S. Eger, E.-L. D. Dinh, and I. Gurevych. "Still not there? Comparing traditional sequence-to-sequence models to encoder-decoder neural networks on monotone string translation tasks". *arXiv preprint arXiv:1610.07796*, 2016.

[191] A. G. Schwing and R. Urtasun. "Fully connected deep structured networks". *arXiv preprint arXiv:1503.02351*, 2015.

[192] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. "Grad-CAM: Visual explanations from deep networks via gradient-based localization". In: *Proceedings of the IEEE International Conference on Computer Vision.* 2017, pp. 618–626.

[193] R. Sennrich and M. Volk. "Iterative, MT-based sentence alignment of parallel texts". In: *Proceedings of the 18th Nordic Conference of Computational Linguistics (NODALIDA 2011).* 2011, pp. 175–182.

[194] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. “Taking the human out of the loop: A review of Bayesian optimization”. *Proceedings of the IEEE* 104:1, 2015, pp. 148–175.

[195] E. Shechtman and M. Irani. “Matching local self-similarities across images and videos”. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE. 2007, pp. 1–8.

[196] K. Simonyan, A. Vedaldi, and A. Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps”. *arXiv preprint arXiv:1312.6034*, 2013.

[197] K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. *arXiv preprint arXiv:1409.1556*, 2014.

[198] P. Siva and T. Xiang. “Weakly supervised object detector learning with model drift detection”. In: *2011 International Conference on Computer Vision.* IEEE. 2011, pp. 343–350.

[199] Ö. Sümer, T. Dencker, and B. Ommer. “Self-supervised Learning of Pose Embeddings from Spatiotemporal Relations in Videos”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV).* 2017, pp. 4298–4307.

[200] M. Sundermeyer, R. Schlüter, and H. Ney. “LSTM neural networks for language modeling”. In: *Thirteenth annual conference of the international speech communication association.* 2012.

[201] I. Sutskever, O. Vinyals, and Q. V. Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems.* 2014, pp. 3104–3112.

[202] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[203] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 1–9.

[204] M. Tan, R. Pang, and Q. V. Le. “Efficientdet: Scalable and efficient object detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2020, pp. 10781–10790.

[205] P. Tang, X. Wang, S. Bai, W. Shen, X. Bai, W. Liu, and A. Yuille. “Pcl: Proposal cluster learning for weakly supervised object detection”. *IEEE transactions on pattern analysis and machine intelligence* 42:1, 2018, pp. 176–191.

[206] C. C. Tappert, C. Y. Suen, and T. Wakahara. “The state of the art in online handwriting recognition”. *IEEE Transactions on pattern analysis and machine intelligence* 12:8, 1990, pp. 787–808.

[207] S. Tian, S. Lu, and C. Li. “Wetext: Scene text detection under weak supervision”. In: *Proceedings of the IEEE International Conference on Computer Vision.* 2017, pp. 1492–1500.

[208] S. Tinney and E. Robson. *Oracc Open Data: A brief introduction for programmers, Oracc: The Open Richly Annotated Cuneiform Corpus*. http://oracc.museum.upenn.edu/doc/opendata/. 2018. URL: http://oracc.museum.upenn.edu/doc/opendata/.

[209] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler. "Joint training of a convolutional network and a graphical model for human pose estimation". *Advances in neural information processing systems* 27, 2014, pp. 1799–1807.

[210] X. Tong and D. A. Evans. "A statistical approach to automatic OCR error correction in context". In: *Fourth Workshop on Very Large Corpora*. 1996.

[211] A. Toshev and C. Szegedy. "DeepPose: Human Pose Estimation via Deep Neural Networks". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1653–1660. DOI: 10.1109/CVPR.2014.214.

[212] J. Uijlings, S. Popov, and V. Ferrari. "Revisiting knowledge transfer for training object class detectors". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1101–1110.

[213] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. "Selective search for object recognition". *International journal of computer vision* 104:2, 2013, pp. 154–171.

[214] D. Ulyanov, A. Vedaldi, and V. Lempitsky. "Deep image prior". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9446–9454.

[215] S. Vanséveren. *Unicode Cuneiform fonts for Macintosh and Windows*. https://www.hethport.uni-wuerzburg.de/cuneifont/. accessed May 20, 2018. URL: https://www.hethport.uni-wuerzburg.de/cuneifont/.

[216] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[217] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. "scikit-image: image processing in Python". *PeerJ* 2, 2014, e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453.

[218] K. Wang, B. Babenko, and S. Belongie. "End-to-end scene text recognition". In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 1457–1464.

[219] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng. "End-to-end text recognition with convolutional neural networks". In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. IEEE. 2012, pp. 3304–3308.

[220] X. Wang and A. Gupta. "Unsupervised Learning of Visual Representations Using Videos". In: *ICCV*. 2015, pp. 2794–2802. DOI: 10.1109/ICCV.2015.320.

[221] C. Wigington, C. Tensmeyer, B. Davis, W. Barrett, B. Price, and S. Cohen. "Start, follow, read: End-to-end full-page handwriting recognition". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 367–383.

[222] R. J. Williams and D. Zipser. "A learning algorithm for continually running fully recurrent neural networks". *Neural computation* 1:2, 1989, pp. 270–280.

[223] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. “The marginal value of adaptive gradient methods in machine learning”. In: *Advances in neural information processing systems*. 2017, pp. 4148–4158.

[224] Z. Wojna, A. N. Gorban, D.-S. Lee, K. Murphy, Q. Yu, Y. Li, and J. Ibarz. “Attention-based extraction of structured information from street view imagery”. In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 1. IEEE. 2017, pp. 844–850.

[225] Z. Wojna, A. N. Gorban, D.-S. Lee, K. Murphy, Q. Yu, Y. Li, and J. Ibarz. “Attention-based extraction of structured information from street view imagery”. In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 1. IEEE. 2017, pp. 844–850.

[226] D. H. Wolpert. “The lack of a priori distinctions between learning algorithms”. *Neural computation* 8:7, 1996, pp. 1341–1390.

[227] Y. Wu and K. He. “Group normalization”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[228] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le. “Self-training with noisy student improves imagenet classification”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10687–10698.

[229] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[230] Z. Xie, Z. Sun, L. Jin, Z. Feng, and S. Zhang. “Fully convolutional recurrent network for handwritten chinese text recognition”. In: *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE. 2016, pp. 4011–4016.

[231] K. Yamauchi, H. Yamamoto, and W. Mori. “Building A Handwritten Cuneiform Character Imageset”. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[232] Q. Ye and D. Doermann. “Text detection and recognition in imagery: A survey”. *IEEE transactions on pattern analysis and machine intelligence* 37:7, 2014, pp. 1480–1500.

[233] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. “How transferable are features in deep neural networks?” In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[234] A. M. Zador. “A critique of pure learning and what artificial neural networks can learn from animal brains”. *Nature communications* 10:1, 2019, pp. 1–7.

[235] A. R. Zamir, T. Wekel, P. Agrawal, C. Wei, J. Malik, and S. Savarese. “Generic 3d representation via pose estimation and matching”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 535–553.

[236] M. D. Zeiler and R. Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[237] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. “Understanding deep learning requires rethinking generalization”. *arXiv preprint arXiv:1611.03530*, 2016.

[238] R. Zhang, P. Isola, and A. A. Efros. “Colorful image colorization”. In: *European conference on computer vision*. Springer. 2016, pp. 649–666.

[239] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. “The unreasonable effectiveness of deep features as a perceptual metric”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.

[240] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang. “EAST: An Efficient and Accurate Scene Text Detector”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 5551–5560. arXiv: 1704.03155.