DISSERTATION

submitted
to the
Combined Faculty of the Natural Sciences and for Mathematics
of the
Ruberto-Carola Heidelberg University, Germany
for the degree of
Doctor of Natural Sciences

Put forward by

Uta Büchler
Born in Offenbach am Main, Germany
Oral examination:

Visual Representation Learning with Minimal Supervision

Advisor: Prof. Dr. Björn Ommer

Abstract

Computer vision intends to provide the human abilities of understanding and interpreting the visual surroundings to computers. An essential element to comprehend the environment is to extract relevant information from complex visual data so that the desired task can be solved. For instance, to distinguish cats from dogs the feature 'body shape' is more relevant than 'eye color' or the 'amount of legs'. In traditional computer vision it is conventional to develop handcrafted functions that extract specific low-level features such as edges from visual data. However, in order to solve a particular task satisfactorily we require a combination of several features. Thus, the approach of traditional computer vision has the disadvantage that whenever a new task is addressed, a developer needs to manually specify all the features the computer should look for. For that reason, recent works have primarily focused on developing new algorithms that teach the computer to autonomously detect relevant and task-specific features. Deep learning has been particularly successful for that matter. In deep learning, artificial neural networks automatically learn to extract informative features directly from visual data. The majority of developed deep learning strategies require a dataset with annotations which indicate the solution of the desired task. The main bottleneck is that creating such a dataset is very tedious and time-intensive considering that every sample needs to be annotated manually. This thesis presents new techniques that attempt to keep the amount of human supervision to a minimum while still reaching satisfactory performances on various visual understanding tasks.

In particular, this thesis focuses on self-supervised learning algorithms that train a neural network on a surrogate task where no human supervision is required. We create an artificial supervisory signal by breaking the order of visual patterns and asking the network to recover the original structure. Besides demonstrating the abilities of our model on common computer vision tasks such as action recognition, we additionally apply our model to biomedical scenarios. Many research projects in medicine involve profuse manual processes that extend the duration of developing successful treatments. Taking the example of analyzing the motor function of neurologically impaired patients we show that our self-supervised method can help to automate tedious, visually based processes in medical research. In order to perform a detailed analysis of motor behavior and, thus, provide a suitable treatment, it is important to discover and identify the negatively affected movements. Therefore, we propose a magnification tool that can detect and enhance subtle changes in motor function including motor behavior differences across individuals. In this way, our automatic diagnostic system does not only analyze apparent behavior but also facilitates the perception and discovery of impaired movements.

Learning a feature representation without requiring annotations significantly reduces human supervision. However, using annotated dataset leads generally to better performances in contrast to self-supervised learning methods. Hence, we additionally examine semi-supervised approaches which efficiently combine few annotated samples with large unlabeled datasets. Consequently, semi-supervised learning represents a good trade-off between annotation time and accuracy.

Zusammenfassung

Computer Vision hat das Ziel die menschliche Fähigkeit, visuelle Umgebungen zu verstehen und zu interpretieren, an Computer weiterzugeben. Ein essenzieller Bestandteil ist hierbei, relevante Informationen von visuellen Daten zu extrahieren, sodass die anvisierte Aufgabe gelöst werden kann. Um zum Beispiel Hunde von Katzen unterscheiden zu können, ist das Merkmal "Körperform" relevanter als "Augenfarbe" oder die "Anzahl der Beine". Im klassischen Computer Vision ist es üblich maßgeschneiderte Funktionen zu entwickeln, die spezifische Merkmale wie Kanten oder Punkte von visuellen Daten extrahieren. Um allerdings Aufgaben zufriedenstellend lösen zu können, wird die Kombination mehrerer Merkmale benötigt. Die Vorgehensweise beim Einsatz von klassischen Computer-Vision-Methoden hat daher den Nachteil, dass jedes relevante Merkmal manuell definiert und an den Computer weitergegeben werden muss wann immer eine neue Aufgabe angegangen wird. Aus diesem Grund enthalten die jüngst publizierten Methoden hauptsächlich neue Algorithmen, die dem Computer beibringen relevante Merkmale eigenständig zu extrahieren. *Deep Learning* besitzt auf diesem Gebiet besonders großes Potential. In Deep Learning lernen künstliche neuronale Netze informative Merkmale automatisch von visuellen Daten zu extrahieren. Die Mehrheit der entwickelten Deep-Learning-Methoden benötigt einen Datensatz mit Annotationen, welche die gewünschte Lösung der Aufgabe vorgeben, sodass das neuronale Netz in der Lage ist, die relevanten Merkmale zu finden. Dies bedeutet allerdings, dass jedes Beispiel im Datensatz manuell annotiert werden muss, was sehr zeit- und kostenintensiv ist. In dieser Doktorarbeit wird nach neuen Deep-Learning-Techniken geforscht, die signifikant weniger manuelle Überwachung benötigen, aber trotzdem zufriedenstellende Ergebnisse in verschiedenen visuellen Aufgabenstellungen liefern.

Der Schwerpunkt dieser Thesis lieg insbesondere auf Algorithmen des eigenüberwachten Lernens, bei dem das neuronale Netzwerk mit einer Ersatzaufgabe, anstatt der ursprünglichen Zielaufgabe, trainiert wird. Die Ersatzaufgabe wird dabei so formuliert, dass keine manuellen Annotationen benötigt werden. Die von uns entwickelte Methode erzeugt ein künstliches Überwachungssignal, indem wir die Anordnung visueller Strukturen durcheinander bringen und dem neuronalen Netzwerk die Aufgabe stellen, die ursprüngliche Struktur wiederherzustellen. Zur umfassenden Auswertung der Fähigkeiten des Verfahrens wenden wir es nicht nur auf verbreitete Computer Vision Probleme wie die Erkennung von Bewegungsaktivitäten an, sondern auch auf biomedizinische Szenarien. Viele medizinische Forschungsprojekte benötigen eine Vielzahl manueller Prozesse, die die Dauer der Entwicklung von Behandlungsmöglichkeiten verlängern. Daher zeigen wir anhand der Forschungsprojekte, bei denen die Motorik von neurologisch beeinträchtigten Patienten analysiert wird, dass eigenüberwachtes Lernen dabei helfen kann, mühsame visuelle Prozesse in medizinischer Forschung zu automatisieren. Um eine detaillierte Analyse des motorischen Verhaltens eines Patienten durchführen zu können, ist es entscheidend, die negativ beeinträchtigten Bewegungen zu erkennen und zu identifizieren. Aus diesem Grund haben wir eine Methode entwickelt, die sogar geringe Veränderungen detektieren und künstlich

verstärken kann, sodass sie besser sichtbar sind. Dabei ermöglicht unser Verfahren außerdem ein personenübergreifendes Detektieren von motorischen Unterschieden.

Das automatisierte Erlernen einer aussagekräftigen Repräsentation visueller Daten, ohne Annotationen zu benötigen, reduziert den manuellen Aufwand signifikant. Das Verwenden von vollständig annotierten Datensätzen führt allerdings beim Stand der Forschung noch häufig zu besseren Ergebnissen im Vergleich zu eigenüberwachten Methoden. Infolgedessen beschäftigt sich diese Thesis zusätzlich mit halb-überwachten Verfahren, bei denen eine große Menge an nicht-annotierten Daten mit wenigen annotierten Daten effizient kombiniert werden. Halb-überwachte Verfahren stellen daher einen guten Kompromiss zwischen Annotationszeit und Leistung dar.

Acknowledgements

At first, I would like to express my sincere gratitude to my advisor Prof. Dr. Björn Ommer. I am especially thankful for his constant support during my Ph.D., his willingness to patiently discuss ideas and results for countless hours and for staying until the end of every submission deadline. I also would like to thank PD Dr. Karl Rohr for his interest in my work and for accepting to review my thesis.

A special thanks to all my colleagues at the CompVis research group and the HCI with whom I have shared many moments of happiness, frustration and inspiring discussions inside and outside of work including Nikolai, Tobias, Michael, Artsiom, Patrick, Johannes, Ekaterina, Dmytro, Alexander, Fabrizio, Barbara and Pamela. I am especially grateful for the strong and hopefully lifelong friendships that emerged during my time in Heidelberg with Timo, Sabine, Lisa and Miguel.

Thanks to my family for their love and encouragement and for always believing in me not only throughout my Ph.D. but my whole life. Without their support I wouldn't be where I am today.

Last but not least, I would like to thank Biagio who was first a colleague but quickly became part of my family. Our heated discussions and diverse perspectives on ideas was a vital part of my accomplishments.

Contents

Chapter 1

Introduction

1.1 Computer Vision

Humans perceive their environment mainly through the five basic senses: sight, hearing, smell, taste and touch. The sensing organs such as eyes, ears and hands send information to the brain which ultimately processes the input. This process is necessary to understand and explore our surroundings. It helps to gain knowledge and to find connections to possibly new encounters and challenges. Especially the human visual system greatly supports humans in understanding the surrounding environment. Scientists have been trying to develop technical devices that assist us in exploring the world. These devices often imitate the ability of objects or behavior occurring in the natural world such as planes that emulate birds or cameras that mimic eyes. Another example is how humans or animals process visual information which they have absorbed through their eyes in order to make sense of the world. This is imitated by computers, which process digital images or videos that are recorded by cameras. The goal of *Computer Vision (CV)* is then to develop algorithms that help computers to understand the content of visual data and to infer information about the environment. In other words, CV attempts to reproduce the abilities of human vision, *i.e.* to help computers to see. Figure 1.1 graphically describes this process.

Computer vision emerged in the early 1970s [158] and is a sub-field of artificial intelligence which aims at simulating human intelligence in machines. At the beginning of CV, researchers believed that processing the visual input with a computer should be an easy task and represents only a small step along the way to developing an artificial intelligence [158]. However, they quickly realized that learning to understand and interpret the visual surroundings is a far more complex task than one would assume. In fact, visual problems which seemed trivial at the beginning are still not fully solved nowadays.

Computer vision methods are developed for a wide variety of applications including security (e.g. surveillance) [32, 75, 154, 186, 118], autonomous driving [109, 188, 111, 58] or healthcare [15, 81, 175, 5]. In order to tackle these applications, there exists a long list of ongoing research areas such as object recognition, posture estimation, person identification, multi-object detection, crowd counting, depth estimation, 3D reconstruction, flow estimation, action recognition, tracking, action segmentation and many more.

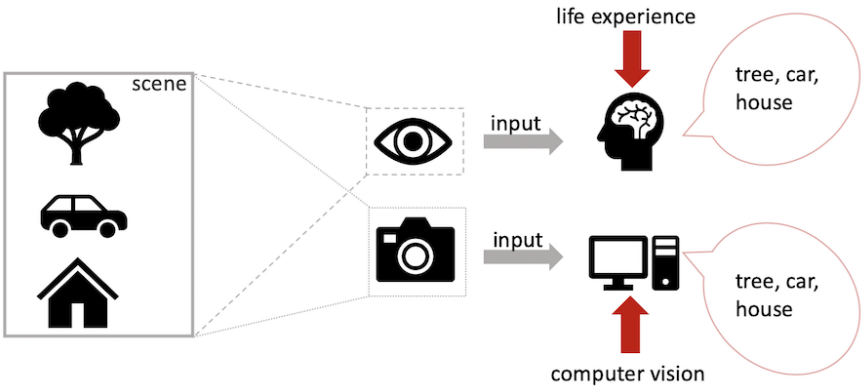


Figure 1.1: Human vision vs. computer vision. Humans capture their surroundings through eyes and send the visual information to the brain which subsequently processes the input. Due to several months or years of exploration and life experience, humans are able to understand their environment and, for instance, recognize the objects in a displayed scene. Computer vision is used to provide a similar ability for computers: given an image of a scene recorded by a camera, CV helps a computer to process the input and eventually infer information about the scene.

One of the most crucial steps to solve these problems is to find a task-oriented representation that best describes the scene and that ignores unimportant factors. In order to identify a person, for instance, humans focus on very specific features of a person's face such as the shape of the eyes or nose; if the person is sitting or standing is rather unimportant. However, if the aim is to recognize the type of movement a person is performing, the posture becomes more important than facial features. Thus, also the performance of computer vision methods heavily depends on the choice of data representation. Traditional feature extractors such as SIFT (Scale-Invariant Feature Transform) [102] or HOG (Histogram of Oriented Gradients) [30] focus mainly on low-level characteristics such as edges or corners. These human engineered features might be successful in distinguishing bananas from apples but are less effective if we want to differentiate between apples and oranges due to their almost identical shape. Moreover, every new task or dataset requires us to manually specify which features the computer should look for. Therefore, the CV community has devoted a big part of its research to methods that teach a computer to autonomously extract relevant and task-specific features. This procedure is also called *representation learning* and is part of the broader topic of machine learning algorithms. Recently, one specific approach has been particularly successful in learning powerful visual representations: *Deep Learning*. A deep learning model is based on an artificial neural network that learns to progressively extract higher level features such as object parts by building them out of simpler characteristics such as edges. Similar to cameras or planes, an artificial neural network is inspired by living organisms. Deep learning draws some insights from the structure and functionality of a brain. However, as humans often need years of experience to naturally understand the environment also deep learning models require a lot of training data to reach an adequate performance.

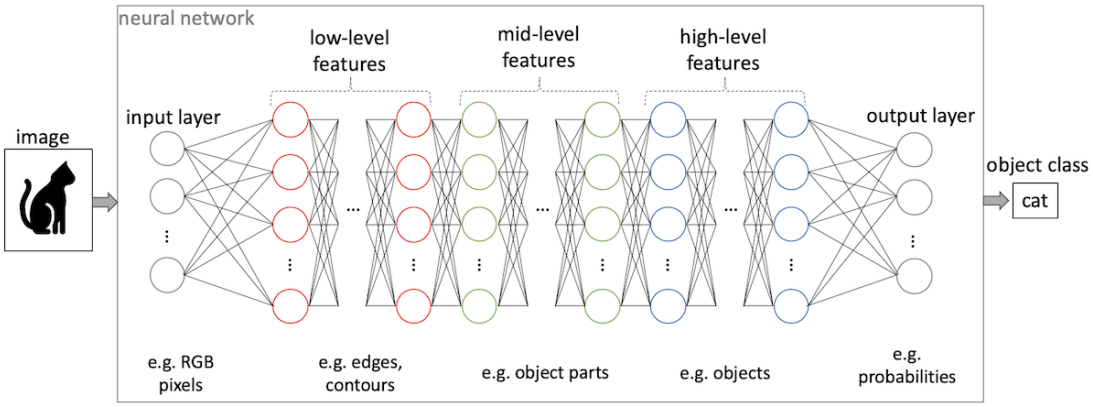


Figure 1.2: Depiction of the structure of a neural network. Low-level features such as edges or contours represent simpler (visual) concepts and are processed in early layers. Mid-level features like object parts are extracted by combining the information passed from previous layers. Finally, high-level features that, for instance, describe the full object, are passed to a final output layer in order to make a decision such as the class of the object shown on the input image. Each column represents one layer that consists of several interconnected neurons (circle).

We will discuss this problem further in the next two sections and provide a more detailed description of deep learning.

1.2 Deep Learning

Recent successes in machine learning and the development of increasingly more powerful computers have brought us closer to the goal of creating an artificial intelligence. Machine learning represents a set of algorithms that interprets data, learns from them, and infers the learned concepts to perform intelligent decisions. Deep learning (DL) is a subset of machine learning algorithms that is partly inspired by the structure and functionality of a brain (visual cortex). Before deep learning, machine learning algorithms such as support vector machines (SVM) [28] or decision trees were highly dependent on manual feature engineering, i.e. hard-coded knowledge. Instead of relying on humans to specify what features to look for, deep learning algorithms teach a computer to acquire their knowledge from raw data.

The idea is to allow a computer to understand the environment in terms of a hierarchy of concepts, meaning that complicated concepts can be learned by building them out of simpler ones. The expression "deep" is used because many simple concepts, build on top of each other, are utilized until a rather complex issue can be solved [61]. Mapping, for instance, an image of a cat directly from pixel intensities to an object class results in a very complicated function. In an artificial neural network, this problem is approached by breaking the complex mapping into many nested simpler functions, where each function is represented by a different layer of the model. Each layer consists of several nodes (artificial neurons) which are densely interconnected. An individual node

might be connected to various nodes in the predecessor layer, from which it obtains information, and several nodes in the following layer, to which it passes information. The connections between the nodes are modeled as weights that define the amount of information passed from layer to layer. Figure 1.2 illustrates the structure of an artificial neural network. In order to provide a good mapping from the input (e.g. images) to the desired output (e.g. object classes) the weights of all layers need to be trained. This is performed by using an optimization algorithm that searches for the set of weights that maximizes the performance on a training dataset. A more detailed background of neural networks and a description of the different layer types are introduced in Chapter 2.

The concept of deep learning exists since the 1940s but was rather unpopular for several years. One reason being the deficient hardware available at that time given that the training process is computationally very costly [61]. Inspired by the Neocognitron network [57] which was presented in 1980 by K. Fukushima, Yann LeCun *et al.* introduced convolutional neural networks (common abbreviations: CNN or ConvNet) [95] in 1989. CNNs are a particular type of artificial neural networks for processing data with a grid-like topology such as image data (for more information on CNNs see Section 2.1). Even though CNNs significantly reduce the computational complexity during training and showed impressive results on some tasks such as the MNIST digit image classification problem, they remained rather unpopular until the early 2000s. In 2009 Russakovsky *et al.* [142] released *ImageNet*, a large object recognition dataset with more than 1 million real images and 1000 object classes. The goal was to move from simple classification problems such as MNIST to real-world problems. Thanks to this large-scale dataset and a novel CNN architecture called *AlexNet* [90], deep learning experienced a breakthrough in 2012. AlexNet reduced the state-of-the-art error rate on ImageNet by a large margin. Since then neural networks gained increasing popularity and many deeper architectures such as VGG [149] or ResNet [67] have been proposed. Neural networks have also shown their potential on various other tasks including video understanding problems. Deep learning is nowadays the leading approach for solving vision tasks and represents a big step forward towards artificial intelligence. In addition, DL advances various interdisciplinary projects involving, for instance, art history [107, 152, 21] or autonomous driving [109, 111, 58, 25].

1.3 Learning a Representation with Limited Supervision

The success of neural networks highly depends on the data. AlexNet, for instance, contains 61 million parameters that need to be trained on the desired task. Without a large-scale dataset like ImageNet it would not be possible to achieve satisfactory results. The deeper a network the more potential it has to solve complex problems. Nevertheless, this also requires more data to train millions or even billions of parameters. Fortunately, visual data is cheap to acquire due to the huge number of images and videos available on the internet. However, training a neural network requires additionally annotations (also: labels) which

define the content of the images or videos (for instance the class of an object displayed on an image). Gathering these annotations is very tedious, cost- and time-intensive considering that every image needs to be labeled manually by a person. Occasionally also privacy, safety or ethic issues prevent us from annotating large datasets. Therefore, besides advancing supervised methods (training the network with manual annotations), researchers have been focusing on developing approaches that learn a useful feature representation with no or only few labels. The methods that do not require any manual annotations are called *unsupervised*. There are several fields in computer vision concerned with developing approaches which are using only few labels such as

- **Weakly-Supervised Learning.** Trains a neural network with lower quality labels that are easier to acquire (e.g. using only bounding box annotations to learn object boundaries [83]) or uses a neural network that is trained on a different fully labeled dataset (or task) as initialization.

- **Zero-Shot Learning.** Training is performed on a small number of classes with the aim of generalizing to new, unseen classes.

- **Semi-Supervised Learning.** Efficiently combines the large number of unlabeled samples with a small amount of labeled samples.

The above list is only a selection from this scope of work. There exist many more expressions describing specific annotation setups. This thesis focuses mainly on unsupervised learning, specifically self-supervised, and partly on semi-supervised learning. Note, that unsupervised and self-supervised learning are used interchangeably in this thesis.

Training a neural network with limited resources has not only the advantage of reducing the labelling effort. It also reflects the way animals and humans learn. According to Horace B. Barlow, a visual neuroscientist, the brain is able to extract knowledge from the massive amount of sensory data perceived through our sensing organs without receiving any direct rewards or punishments [9]. This suggests that the perceived data already incorporates a considerable number of supervisory signals. In machine learning, this property is exploited by self-supervised methods which learn a powerful feature representation without requiring manual labels. Self-supervised approaches train the network indirectly by solving a surrogate task where its labels can be extracted automatically from the data (for more information please see Chapter 3).

Learning with limited annotations is especially helpful for interdisciplinary projects that might address rather atypical vision tasks. Instead of requiring experts from other fields such as biology or art history to spend time on annotating millions of samples, self-supervised learning enables the direct usage of unlabeled datasets. The benefit of applying self-supervised learning methods to interdisciplinary projects has been, for instance, demonstrated in medical imaging [159, 23] or autonomous driving [113, 27]. In this thesis, we particularly show in Chapter 4 that self-supervised learning can efficiently assist neuroscientists in analyzing the motor function (motor behavior analysis) of patients suffering from a neurological disease.

1.4 Objective

Teaching a neural network to understand the content of images or videos is a challenging task. Especially in unsupervised learning only lately methods are starting to show satisfactory results on benchmark datasets. Thus, the principal objective of this thesis is to develop novel deep learning approaches for learning feature representations by employing either none or only a small amount of manual annotations. The performances of the learned representations are evaluated and compared to previous works on several image and video understanding tasks including an interdisciplinary project. The remainder of this section specifies the various sub-goals of this thesis.

Self-supervised learning provides a powerful tool for learning meaningful feature representations without requiring labels. This thesis aims at developing a novel self-supervised learning method to learn an image *and* video representation simultaneously. The goal is to learn image features that are applicable to a wide variety of image understanding tasks such as object classification and human posture estimation and to improve action recognition using the resulting video representation.

Besides applying the novel methods to common computer vision problems, this thesis additionally aims at facilitating the research in interdisciplinary projects using self-supervised machine learning. In particular, this dissertation includes a collaboration with neuroscientists who analyze the movement of animals and humans suffering from neurological diseases that negatively affect the motor function. Self-supervised methods can support researchers by automatically analyzing the behavior before and after medical treatments. Moreover, the overall goal of this collaboration is to develop a diagnostic support system that can discover even small changes in motor function in order to find an optimal treatment. Thus, this thesis aims at developing a tool that not only analyzes behavior but also compares and quantifies even subtle deviations. The discovery of small differences in movements is not only advantageous in medical scenarios but also, for instance, in sports for comparing and identifying sub-optimal movements. Therefore, the developed method should be robust enough to function also in less restricted scenarios with videos which are recorded outside of a medical lab.

Self-supervised learning represents an impressive alternative to supervised learning if no labels are available. However, self-supervised methods are not yet as powerful as supervised methods and therefore lead often to lower performances. Thus, this thesis additionally aims at investigating the improvement in performance when combining a large dataset of unlabeled samples with few labeled examples. In particular, this thesis aims in developing a new semi-supervised method for improving the accuracy on action recognition.

1.5 Contributions

The contributions of this thesis are the following:

- An overview of the most influential self-supervised image- and video-based approaches published from 2015 until 2020.

- A new surrogate task for self-supervised learning is developed in this thesis. The surrogate task exploits time information by firstly permuting video frames and secondly asking the network to reconstruct the original frame ordering. In combination with an LSTM network the model learns a fine-grained image and sequence representation without requiring any manual annotations. The representation can be used to address several tasks such as action recognition and human pose estimation.

- A framework for combining two related permutation tasks that capture spatial and temporal information from images and videos, respectively, is proposed. Given that the two unsupervised tasks are related, the network can be trained simultaneously on two data types without requiring a pre- or post-processing step for adjusting them to each other. The resulting shared representation contains information extracted from images and videos and can therefore be used to tackle a wide variety of visual understanding tasks.

- In this thesis, self-supervised learning is applied to common benchmark datasets of the computer vision community but also to an interdisciplinary project which is based on a close collaboration with neuroscientists. The self-supervised method described above is used to learn a fine-grained behavior representation of diverse subjects (animals and humans) which are suffering from a neurological disease. Biagio Brattoli has equally contributed to this idea.

- Many processes in motor behavior analysis are still performed manually. Given the behavior representation learned via the self-supervised method, this thesis additionally provides a pipeline for automatically analyzing the behavior/motor function of impaired subjects. The proposed method can predict the fitness of a subject, classify diseases, analyze the rehabilitation progress and compare the behavior across diverse subjects. Evaluations are performed on two different types of species (rodents and humans). Implementing and applying the pipeline to rodents has been performed in collaboration with Biagio Brattoli (equal contribution).

- A new approach for discovering and comparing subtle differences in posture is proposed. Small behavioral changes of impaired subjects in comparison to healthy subjects are easily overlooked by humans due to the different appearances. The model proposed in this thesis is able to facilitate the perception of impairment by magnifying subtle posture deviations between an impaired and healthy subject. For that matter, the approach includes at first an unsupervised training of a generative network that learns to separate posture from appearance (background, color of clothes etc.). Then during inference, the generative model is able to detect and magnify subtle posture deviations across subjects to finally generate images that display the magnified differences. The resulting magnifications increase the visibility of impairment for humans and can therefore simplify the interpretation of symptoms.

- Magnifying posture deviations across subjects requires a strong separation

of posture and appearance if the recording setup changes from subject to subject (e.g. different locations). Therefore, a novel disentanglement loss is introduced to enforce a stronger partition of posture and appearance.

- In order to improve the generation quality of magnified images, a new loss is proposed. Different to the previously mentioned magnification, which is only applied after training the generative model, the new loss enables the model to incorporate the magnification process already into the training. This guarantees a higher image generation quality and more realistic magnifications.

- A novel semi-supervised learning approach for action recognition is proposed. The presented approach learns powerful features from partly labeled video datasets by adapting successful methods in semi-supervised learning for images to videos and by exploiting recent advances in unsupervised video representation learning.

- This thesis introduces five new datasets for behavior analysis and magnification. These datasets have been collected by or in collaboration with Anna-Sophia Wahl, Martin E. Schwab, Lenard Filli, Fritjof Helmchen and Michael Dorkenwald.

1.6 Thesis Organization

Chapter 2 introduces the concept of artificial neural networks including their structure, different layer types and common approaches for optimizing their performances. This chapter additionally summarizes established network architectures that have been successfully applied to a variety of different visual understanding tasks.

Chapter 3 first introduces self-supervised representation learning and provides a summary of the most influential approaches. Previous works on self-supervised learning mainly addressed image understanding tasks. In this chapter, we describe our self-supervised framework of permuting video frames where the resulting representation can be used to solve video understanding tasks. Then, we propose an extension of that method by training a multi-task neural network on spatial and temporal permutations for better generalization. We evaluate the performance of both methods on action recognition, pose estimation, image classification, image detection and image segmentation. The approaches introduced in this chapter are based on two publications [15, 16] at CVPR 2017 and ECCV 2018.

Chapter 4 presents an interdisciplinary project which highly benefits from unsupervised machine learning approaches to automatize manual processes. Clinical studies for analyzing the motor function of subjects that are suffering from a neurological disease have so far included many manual processes. To ease the work of researchers and doctors, we propose to apply self-supervision to learn a fine-grained representation of behavior without requiring manual annotations. This chapter first introduces previous (supervised) works on behavior analysis and presents the experimental setups of the clinical studies on which we evaluate

our approach. Then we describe our method for learning a fine-grained posture and behavior representation. In order to provide a detailed diagnostic system that is able to discover even small indications of impairment, we additionally present an approach that magnifies subtle posture differences between an impaired and healthy subject. Our magnification method facilitates the perception and comparison of impairment across different subjects. We demonstrate the applicability of our complete diagnostic system (including our learned representation and our magnification method) on two different species for diverse behavior analysis tasks such as disease classification and rehabilitation evaluations. Our diagnostic support system is currently under review at a journal.

Chapter 5 describes a more robust magnification method for amplifying posture deviations across subjects. Besides the medical scenario introduced in the previous chapter, our robust magnification approach is also applicable to more complex video setups such as outdoor recordings with different lightning and backgrounds. We first introduce two novel losses for disentangling posture from appearance and for learning to magnify posture deviations. Then, we evaluate and compare our magnification results with previous works using three datasets with different settings. This chapter is based on our publication at CVPR 2020 [43].

Chapter 6 introduces a semi-supervised learning (SSL) method for action recognition. Self-supervised learning minimizes the amount of human supervision but does not yet reach as high performances as supervised methods. Therefore, we propose a novel semi-supervised method for action recognition that efficiently combines few labeled samples with a large dataset of unlabeled samples. Our approach consists of an unsupervised pre-training, fine-tuning and pseudo-labelling based post-training phase. After describing our novel approach, we evaluate its performance on two benchmark datasets with up to 400 classes and demonstrate the importance of all stages by providing several ablation studies.

Chapter 7 concludes the thesis with a summary and final discussion.

Chapter 2

Background

Artificial neural networks (ANN) have been tremendously successful during the last few years in learning powerful feature representations for addressing various visual understanding tasks. Therefore, this thesis is mainly concerned with improving and developing new ANN based methods. This chapter contains a short summary of the most important concepts of ANNs. If the reader is already familiar with neural networks, this chapter can be skipped since it does not contain any methodological details about the approaches developed in this thesis.

2.1 Artificial Neural Networks

An artificial neural network (or often simply called *neural network*) consists of an hierarchy of concepts starting from simpler data representations to more complicated ones. As introduced in Section 1.2, the aim is to approximate a complex function $y = f^*(x)$ that maps an input x (e.g. an image) to the desired output y (e.g. an object class) by breaking it into many simpler nested functions $f^{(l)}$ with $l = 1, ..., N_L$ and N_L the amount of functions used, i.e.

$$f^*(x) \approx f(x;\theta) := f^{(N_L)}(f^{(N_L-1)}(...f^{(2)}(f^{(1)}(x;\theta^{(1)});\theta^{(2)})...;\theta^{(N_L-1)});\theta^{(N_L)}), \tag{2.1}$$

where $\theta = (\theta^{(1)}, \theta^{(2)}, ..., \theta^{(N_L)})$ represents the learnable parameters that result in the best function approximation. This type of ANN is also called *feedforward neural network* since the information flows only in one direction. The process of predicting y given x using the feedforward neural network is commonly named *forward pass*. Figure 1.2 illustrates the concept of a feedforward neural network graphically. In practice, the nested functions $f^{(l)}$ are implemented as *layers*, whereas the first layer is represented by $f^{(1)}$, the second by $f^{(2)}$ and so on and the depth of a neural network is defined by N_L. Note, that $\theta^{(1)}, ..., \theta^{(N_L)}$ are matrix valued and their dimensionalities represent the width of the corresponding layers. The first layer is also called *input layer*, the last layer *output layer* and all the layers in between are named *hidden layers* since the training data does not specify their desired output. Each hidden layer consists of a set of neurons that are connected with neurons from adjacent layers but not with neurons from the same layer.

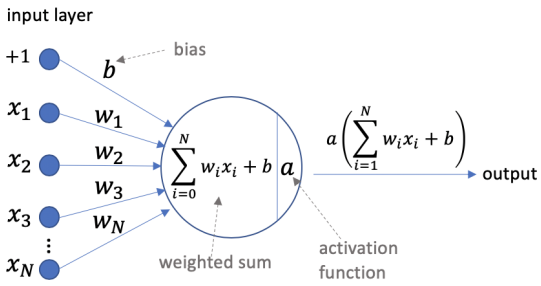


Figure 2.1: Depiction of a Perceptron.

In order to obtain a good function approximation $f(x;\theta)$, we require a strategy that finds the best values for θ (also called *weights*) given a set $\{x_i, y_i\}_{i=1}^{N}$ of input-output pairs. This procedure is called the *training* of a neural network and consists essentially of a forward and backward pass. The latter represents the actual "learning" of the ANN and is responsible for updating the weights accordingly to increase the performance/decrease the error. During the forward pass, the input is fed into the network and passed through all layers until the final layer outputs a prediction score per input sample. To perform the backward pass, we require a loss function (Section 2.3) that evaluates the prediction performances on the training set given the true labels y_i. The computed loss is then back-propagated from the last layer up until the first layer by using an optimization algorithm that adjusts the weights in the direction of improvement. In Section 2.4 we introduce commonly used optimization algorithms which search for the set of weights that maximizes the performance.

2.1.1 Perceptron

Neural networks that map the input directly to the output (no hidden layers) are often referred to as *single-layer neural networks* and are only able to learn linearly separable patterns. A perceptron ranks among supervised single-layer neural networks and represents a mathematical model of a biological neuron. Figure 2.1 illustrates a perceptron and its internal functionality. Similar to a biological neuron, a perceptron controls the strength of influence of one neuron on another. This is practically implemented by using learnable weights w_i. The final signal is computed by summing up the weighted input values. If the resulting signal is above a specific threshold, the neuron outputs (also called to 'fire') its signal. However, the value of the signal can be anything between $-\infty$ and ∞. Therefore, it is necessary to define the bounds (also 'firing rate') of the neuron's signal. The firing rate is practically modelled by using an activation function a. The most elementary form of a would be a binary step function or threshold function that simply decides if the neuron is firing or not,

$$a(x;\theta) = \begin{cases} 1 & \sum_{i=0}^{N} w_i x_i > 0 \\ 0 & \sum_{i=0}^{N} w_i x_i \leq 0. \end{cases} \tag{2.2}$$

with $\theta = (w_1, ..., w_N)$. This activation function might work well for a binary problem. However, if more than two classes are involved, we require multiple

neurons that could all output 1 or 0, making it impossible to determine the only correct class. Therefore, several alternative activation functions (especially nonlinear activation functions) have been proposed over the years. In Section 2.2 we discuss the most common and widely used activation functions.

The weights w_i of a perceptron are trained by applying an optimization approach such as stochastic gradient descent given a training set of input-output pairs (see Section 2.4).

A perceptron can implement all elementary logical functions, such as AND, OR or NOT. However, more complex functions such as XOR do not represent a linear separable pattern and can therefore not be modelled by a single perceptron. In this case we require a multi-layer perceptron (MLP) with a nonlinear activation function which is also often seen as the *regular neural network*.

2.1.2 Regular Neural Networks

Regular neural networks contain at least one hidden layer and are able to represent nonlinear functions. In contrast to convolutional neural networks (subsequent section), multi-layer perceptrons only contain *fully-connected (FC)* layers. As the name already suggests, an FC layer signifies that each node is connected to every node in adjacent layers. Their output can therefore be computed by a simple matrix multiplication. As a side note, also a perceptron is considered fully connected since every input node is connected to the output node(s) and one fully-connected layer is composed of one or multiple perceptrons. The output of a two-layer neural network (one hidden layer and one output layer), for instance, can be mathematically expressed in the following way

$$y_{pred} = f(x;\theta) = \theta^{(2)} \cdot a(\theta^{(1)}x) \tag{2.3}$$

with $\theta^{(1)}$ and $\theta^{(2)}$ the weights of the hidden layer and output layer, respectively and a a (nonlinear) activation function.

Unfortunately, a regular neural network does not scale well to full images. A two-layer neural network that receives images with a size of 200x200x3 already contains more than 240,000 weights which need to be trained. In the next section we describe an alternative and more efficient layer type that exploits the properties of images to reduce the number of parameters and the immense computational costs.

2.1.3 Convolutional Neural Networks

As a regular neural network, a convolutional neural network (CNN) contains an input layer, output layer and one or multiple hidden layers. The main difference between a regular and convolutional neural network is the type of layers. A conventional CNN is composed of diverse layer types, namely

- Convolutional layers,
- Pooling layers and
- Fully-connected layers.

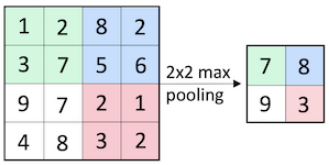


Figure 2.2: Example of a 2x2 max pooling operation.

The following paragraphs provide a brief description of the different layer types. Since fully-connected layers have been already discussed in the previous subsection, this topic will not be covered subsequently.

Convolutional Layers. As the name suggests, a convolutional (conv) layer consists of a set of learnable convolutional filters. In contrast to a fully-connected layer, the neurons in a conv layer are only connected to a small region of the previous layer. Thus, the number of parameters in a conv layer is smaller than in an FC layer. This property comes from the fact that the same convolutional filter is applied to all locations of the input features. Moreover, each filter has only a small receptive field leading to a small number of parameters per filter. The neurons in a conv layer are arranged in 3 dimensions: *width, height,* and *depth*. The width and height characterize the size of the convolutional filter and the depth is defined by the size of the input volume (e.g. if the input is an image with 3 channels the depth of the first conv layer would be 3). Practically, each filter is convolved across the width and height of the input in a sliding window manner. This procedure produces a 2D feature map which provides the response of the chosen filter at every spatial location. A conv layer usually contains several convolutional filters where each of them produces a separate 2D activation map. Stacking these activation maps results in a 3D output volume. Since the filters are applied to spatially neighboring input features, the original structure of the input (e.g. the x-y structure of an image) is maintained. Thus, besides the advantage of reducing the amount of learnable weights, a conv layer additionally preserves the spatial information. In this way, a CNN is able to identify spatial structures such as edges or objects through the application of the corresponding filters (e.g. a sobel filter for edge detection). The optimal values of the filter parameters for a particular task and dataset are learned during the training of the CNN. We would like to refer the reader interested in learning more about convolutions to related literature such as [61] for more details.

Pooling. The main purpose of a pooling layer is to decrease the number of parameters and computational costs by reducing the spatial dimensionality of the feature maps (a depth slice of the input). The pooling operation (also called subsampling or downsampling) operates independently on every feature map. The most common pooling operations are max pooling, average pooling and sum pooling. The operations are performed by applying a max, average or sum filter to a 2D sub-region of an input feature map. If a max filter has, for instance, a size of 2×2, the operation computes the max over 4 numbers. Figure 2.2 illustrates an example of a 2×2 max pooling operation applied to a 4×4 feature map. In practice, max pooling has proven to be more effective than applying an average or sum filter. It is also worth noting that besides reducing the number of parameters, a pooling operation additionally increases

the receptive field and aggregates multiple low-level features in the neighborhood. Therefore, downsampling also causes a local rotational and positional invariance advantageous for various tasks such as object recognition.

Overall Structure of a Basic CNN. Every CNN contains at least one convolutional layer and a pooling layer is periodically inserted in-between successive conv layers. In order to indicate if the output of a neuron is relevant or not, an activation function (more details in Section 2.2) is applied to the convolved feature map either before or after the pooling operation. The final layer of a conventional CNN is a classifier in the form of a fully-connected layer. Thus, every neuron in the last layer is connected to all neurons from the previous layer. The final output represents the predicted class scores of the given input, i.e. the dimensionality of the output is defined by the number of classes occurring in the dataset. To summarize, a CNN gradually transforms the original input (e.g. an image) layer by layer from the initial values (e.g. pixel intensities) to the final class scores. The parameters of the conv and FC layers are learned during training while the activation function and the pooling operation are fixed functions. In the following, we briefly discuss additional strategies and layer types proposed during the last few years for improving the overall performance of CNNs.

Regularization. One of the major problems of CNNs is that they are prone to overfitting due to their complexity. Overfitting occurs when a function approximation is too closely fit to a small set of data points and fails in generalizing to unseen data. In this case, the CNN might output perfect predictions for the training set but is not able to predict the classes of a left-out testing set. Standard regularization methods essentially push some of the parameters towards zero to reduce the model complexity and increase the generalization abilities for ultimately improving the performances on various tasks. The most common and established approaches proposed over the past few years are L2/L1 regularization, Dropout [151] and data augmentation. The *L2 regularization* penalizes the squared magnitude of all weights by adding the term $\frac{1}{2}\lambda w^2$ with w the weights in the network and λ the regularization strength. This type of regularization heavily penalizes peaky weights and encourages the CNN to employ all inputs consistently. Practically, the L2 regularization linearly decays every weight towards zero during training. The *L1 regularization* adds the term $\lambda|w|$ to the objective and has the property to direct some of the entries in a weight vector very close to zero. Thus, the weight vector becomes very sparse, meaning that only the most important inputs are considered. In practice, the L2 regularization often outperforms the L1 regularization. There exists also a combination of both regularizations called *elastic net regularization*. *Dropout* [151] is one of the most effective and most simple regularization approaches and is usually placed after an FC layer. A dropout layer sets individual neurons with a probability of p to zero during training. p is commonly set to 0.5 but represents a hyperparameter that might need to be adjusted for reaching its maximum potential. Dropout forces the network to have redundant representations and prevents neurons from being highly dependent on others (co-adaption). The idea behind *data augmentation* is to prevent overfitting by artificially increasing the size of the training set without requiring manual annotations. Common data augmentations for images are cropping, padding, horizontal flipping, scaling and color jittering. Nearly every

CNN model is nowadays trained with data augmentation.

Besides the previously mentioned regularization methods, many more strategies have been proposed during the last few years. Among them are early stopping, DropConnect [176], DropBlock [59], fractional max pooling [63], stochastic depth [70] and Cutout [35]. For more information please see [61] or similar literature.

Normalization. Normalization can be performed on the input data itself or the hidden layers of an ANN by adjusting and scaling the values. This paragraph addresses techniques of the latter. Normalization provides many advantageous and is presently a standard strategy used in CNNs. For instance, besides preventing weights to explode by restricting them to a certain range, normalization also increases the overall training speed and prevents an internal covariate shift (distribution of the activations is constantly changing). Common normalization techniques are local response normalization [90], batch normalization [73], layer normalization [7] and group normalization [184]. In the following, we will describe *batch normalization* (short: batchnorm) since it is presently the most established normalization technique. Batchnorm increases the stability of a CNN by normalizing the values of a previous activation based on the samples in a batch (a collection of several training samples during one forward pass). In particular, for every value x_i the approach subtracts the batch mean $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$ and divides by the batch variance $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$,

$$\hat{x_i} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{2.4}$$

with m the number of samples in a batch. However, normalizing a unit in this way reduces the expressive power of a neural network. In batchnorm, the expressive power of a CNN is maintained by introducing two learnable parameters to each layer: γ for scaling the unit and β for shifting the value. This results in the following final output

$$y_i = \gamma\hat{x_i} + \beta. \tag{2.5}$$

The learnable parameters allow the new values y_i to have any mean and standard deviation and are optimized during the training step. A batchnorm layer is usually inserted after a fully connected layer or a conv layer and before the activation function.

The following sub-sections introduce popular network structures for solving different types of problems.

2.1.4 Fully-Convolutional Networks

Fully-convolutional networks are used for segmenting the objects shown on an image (semantic segmentation). For that reason, we require a class prediction for every pixel instead of the entire image. Moreover, it is important to keep the original structure of the input (x-y locations of pixels) which demands for a model that preserves the spatial information. A convolutional neural network includes a fully-connected layer which in contrast to convolutional layers obscures

the spatial coordinates. However, the functional form of an FC and conv layer is identical since both layers compute dot products. The only difference between these two-layer types is that conv layers are only connected to a local region in the input. Therefore, Long *et al.* propose in [101] to simply transform (also called convolutionalize) the FC layers of a CNN into conv layers by setting the filter size exactly to the size of the input. A fully-convolutional network has the advantages that it contains less parameters, the computational costs are reduced, and it can process samples with a variable image size. However, due to the poolings and convolutions (downsampling), the output of a fully-convolutional network does not have the same dimensionality as the input image causing a rather coarse segmentation mask. Thus, Long *et al.* [101] propose an alternative method called FCN that performs a gradual upsampling of activations by using deconvolutions. This approach is slightly more complex than simply convolutionalizing FC layers but produces more fine-grained segmentation masks.

2.1.5 Deep Generative Models

The aim of a generative model is to approximate the true distribution a finite dataset was sampled from and to ultimately employ the resulting model for various kinds of tasks. Instead of learning to predict a class given a particular data point, a generative model learns a joint distribution over all data points without requiring any human supervision (unsupervised). Possible application scenarios are density estimation, sampling for generating new data points given the learned distribution and unsupervised representation learning. Two of the most commonly employed generative model approaches in deep learning are (variational) autoencoders and generative adversarial networks (GAN). The following paragraph will first introduce autoencoders, followed by a description of variational autoencoders. To learn more about GANs please see related literature such as [62] or [61].

Autoencoder. The target values of an *autoencoder* (AE) are equal to the input values which means that an AE f tries to learn an approximation of the identity function,

$$\hat{x} = f(x;\theta) \tag{2.6}$$

with x the input to the AE, $\hat{x}$ the output and θ its weights. However, the actual purpose of an AE is not to simply copy the input to the output, but to extract useful features of the underlying data distribution. Producing an output (almost) identical to the input is merely a tool for training the neural network without requiring any manual labels. An AE consists of two main phases: the reduction step (encoding) and the reconstruction phase (decoding). During the reduction step the network learns to capture the most salient features (also: latent attributes) of the dataset by mapping the input (e.g. pixel intensities) into a smaller dimensional embedding space. In fact, if the neural network would be constructed as a linear model, the reduction phase would result into a similar dimensionality reduction as discovered in PCA. However, thanks to the nonlinear activation functions, an autoencoder offers a more powerful nonlinear generalization of PCA. During the reconstruction phase, the AE attempts to generate a

Figure 2.3: Illustration of the structure of an autoencoder (left) and variational autoencoder (right). *Left*: An encoder E maps the input image x to a lower dimensional latent space to obtain $z = E(x)$ which is then fed into the decoder D to reconstruct the image. *Right*: The encoder of a variational autoencoder outputs the mean and standard deviation of a normal distribution per latent attribute from which the latent representation z is sampled. Please note, that the trapezoids (blue, rose) contain in practice several layers including convolutional and pooling operations.

representation as close as possible to the input given the reduced encoding. The network parts of an AE responsible for reducing the dimensionality and for reconstructing the input are called *encoder* and *decoder*, respectively. Considering the previous characterization, Equation 2.6 is updated as follows

$$\hat{x} = D(E(x;\theta^{(E)});\theta^{(D)}) \tag{2.7}$$

with E the encoder, D the decoder and $z := E(x;\theta^{(E)})$ is called the latent representation. Figure 2.3(left) illustrates a simplified version of an autoencoder. The encoder and decoder consist usually of several hidden layers including also convolutional operations. To facilitate the task of the decoder, it is common to include *skip connections* between the encoder and decoder to connect specific layers across the two network parts. The skip connections enable the decoder to additionally receive, besides the latent representation, low-level or mid-level features (which are extracted in earlier layers of the encoder). The weights of an AE are trained by measuring the reconstruction error (see Section 2.3 for loss functions) and by backpropagating the error using one of the optimization methods described in Section 2.4.

Variational Autoencoder. The encoder of an AE, as described above, outputs a single value for each encoding dimension to describe the latent attributes. The encoder of a variational autoencoder (VAE), on the other hand, provides a probability distribution for each latent attribute. Thus, during the decoding phase we do not simply input the embedding provided by the encoder into the decoder, but we randomly sample from each latent distribution to generate the input vector for the decoder. In this case, the VAE is forced to build a continuous and smooth latent space representation and directly learns the parameters of the probability distribution instead of a compressed representation. During training, the encoder outputs the mean and standard deviation for a normal distribution $z \sim \mathcal{N}(\mu, \sigma^2)$, from which the latent representation z is sampled and input into the decoder D for reconstruction. In practice, μ and σ are implemented as fully-connected layers. Figure 2.3(right) illustrates the coarse structure of a VAE. Besides the reconstruction loss already used for AEs, a VAE is additionally optimized by minimizing the Kullback-Leibler distance between the encoder output and a normal distribution. The additional loss guarantees the sampling distribution to be normal.

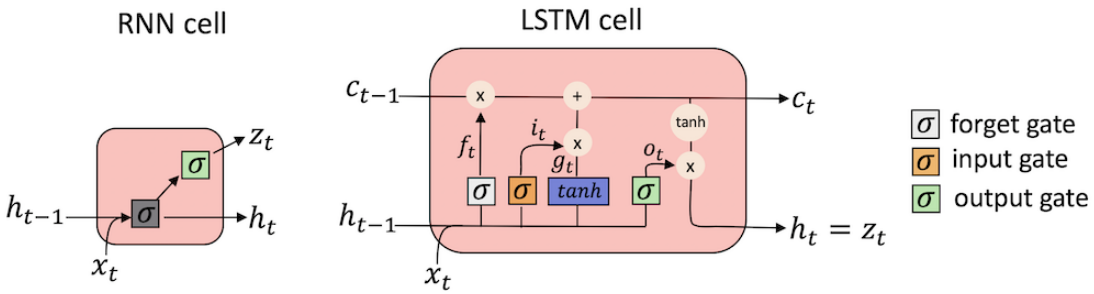


Figure 2.4: Depiction of a basic RNN cell (left) and an LSTM cell (right). The integration of a memory system and a forget gate in the LSTM cell enables the network to explicitly learn when a previous hidden state should be forgotten and when it should be updated using the new input. Therefore, LSTMs are able to better capture the information as the duration of the dependencies increases.

2.1.6 Recurrent Neural Networks

Recurrent neural networks (RNN) allow to reason based on previous events and are especially beneficial for processing long sequences (from videos) or lists, i.e. if temporal or volumetric context is essential for solving the underlying task. Each cell in an RNN layer is connected with its successive cell to pass the present information to the next time step. Unfortunately, basic RNNs have difficulties to capture the information in real-world problems as the duration of the dependencies increases. Yoshua Bengio *et al.* explores this topic in [12]. Luckily, long short-term memory networks (LSTMs) that have been proposed in 1997 by Hochreiter and Schmidhuber [69] are capable of learning long term dependencies due to their prevention of the vanishing gradient problem (weights are not changing their values). In fact, LSTMs have started to show its abilities in speech recognition in 2007 by outperforming traditional models [53] and they have been fully established in the vision community in 2015, where Donahue *et al.* [39] have proposed an end-to-end trainable LSTM model for visual understanding tasks.

RNNs learn temporal dynamics by mapping input sequences to hidden states which are then mapped to outputs in a recurrent manner. Considering, for instance, the aim of learning sequence specific features for human actions, an RNN links a posture x_t at time point t in consecutive frames by means of hidden states h_t and a non-linear activation function σ

$$\begin{aligned} h_t &= \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \\ z_t &= \sigma(W_{hz}h_t + b_z). \end{aligned} \tag{2.8}$$

By learning the parameters W and b we obtain an output representation z_t. Basic RNN cells have a very simple structure and only contain a non-linear activation function unit (see Figure 2.4 left). LSTM cells, on the other side, incorporate memory units that allow the network to explicitly decide when to forget previous

hidden states and when to update them given the new information x_t,

$$\begin{aligned} i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\ f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\ g_t &= \mathbf{tanh}(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\ h_t &= o_t \odot \mathbf{tanh}(c_t). \end{aligned} \tag{2.9}$$

Therefore, LSTMs have the possibility to remove or add information to the state of the current cell using regulated structures. These structures, also called gates, enable an LSTM layer to maintain information in memory for a longer period of time in contrast to RNNs. The structure of an LSTM cell is illustrated in Figure 2.4(right).

In practice, an LSTM layer is commonly attached to a CNN network right before the first fully-connected layer. During training and testing, the CNN is used to separately extract the spatial information from several video frames and the RNN receives the stacked feature maps of all video frames as input to obtain spatiotemporal features.

2.1.7 3D Convolutional Neural Networks

Similar to recurrent neural networks, 3D convolutional neural networks (3D CNN) are useful if temporal or volumetric context is important for solving a particular task such as action recognition.

In standard CNNs, convolutional operations are applied on 2D feature maps for capturing the spatial information on images. When working with videos, it is important to additionally consider the temporal component. Therefore, besides RNNs, researches have first proposed to stack several frames along the channel dimension before feeding them into the network and to apply basic 2D convolutions on the stacked input. The input dimensionality of, for instance, 5 RGB frames stacked along the channel dimension would be then $w \times h \times 5 \cdot 3$ with w and h the width and height of the input images, respectively. The disadvantage of this strategy is, however, that the temporal information of the input signal is lost right after the first convolution is applied since the output is only 2 dimensional. 3D CNNs, on the other hand, incorporate motion information by performing 3D convolutions and 3D pooling operations. The input frames are stacked along a 4th dimension (input dimensionality: $w \times h \times 3 \times 5$) and the 3D operations produce an output volume instead of a 2D feature map. In this way, the temporal information of the input signal is preserved throughout the network until a fully-connected layer is reached. Similar to standard CNNs, there exist several proposals of different architectures for 3D CNNs. The most popular 3D CNN architectures are introduced in Section 2.5.

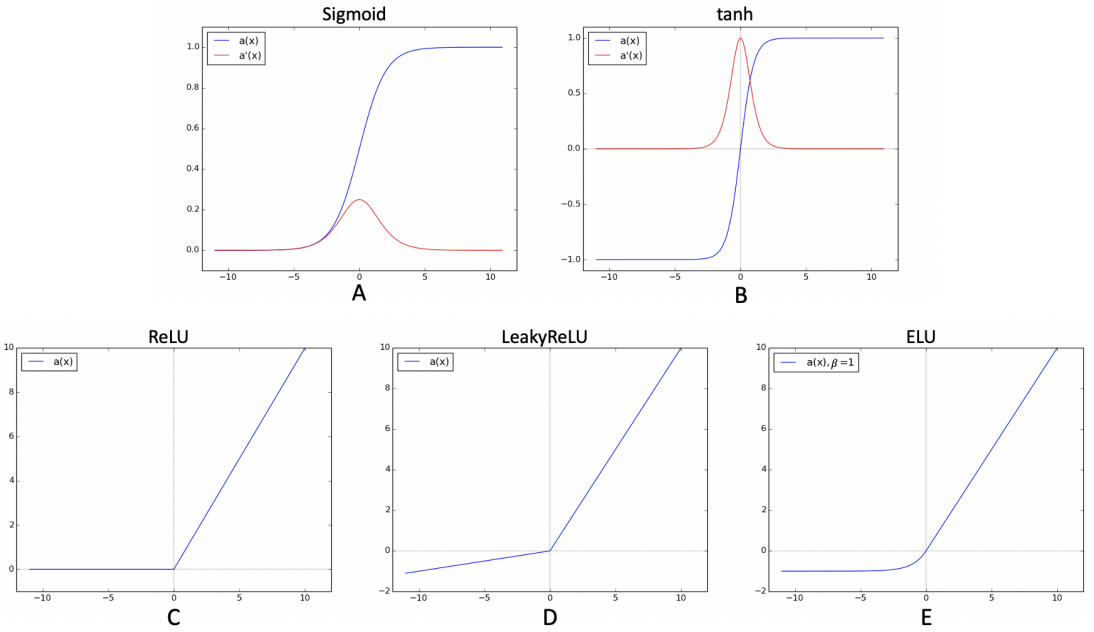


Figure 2.5: Illustration of the most common activation functions. The red graphs represent the derivations of the corresponding activation functions shown in blue.

2.2 Activation Functions

An activation function determines whether a neuron should be activated (fired) or not depending on its relevance for the final prediction of the model. Moreover, a nonlinear activation function introduces non-linearity into the model making it possible to adjust to more complex tasks. In the following, we describe the most common nonlinear activation functions for neural networks and list their advantages and disadvantages. Figure 2.5 displays the graphs of the different activation functions.

1. **Sigmoid**

   $$a(x) = \frac{1}{1 + e^{-x}}$$

   The sigmoid function squashes a value between $[0, 1]$ and represents a saturating firing rate of a neuron. As apparent from the red line in Figure 2.5A, the derivation (which is used during the update step as described in 2.4) of saturated neurons is 0 and can therefore cause the network to get trapped, i.e. the values of the weights are not changing. This is also called the vanishing gradient problem. Moreover, since the sigmoid function is not zero-centered, all gradients of the weights are either all negative or positive if the input is always positive.

2. **Tanh**

   $$a(x) = \tanh x$$

   tanh squashes the output between $[-1, 1]$, is zero-centered and is often used for binary classification tasks. On the positive side, tanh maps negative

inputs to a strongly negative output and zero inputs are mapped close to zero. However, also with tanh as activation function, saturated neurons have a gradient close to 0 causing the network to be susceptible to the vanishing gradient problem.

3. **ReLU**

   $a(x) = \max(0, x)$

   With ReLU as activation function, all negative values are set to 0 which allows the network to easily obtain a sparse representation for better predictive power and less overfitting. Moreover, since the positive inputs never saturate, the network does not suffer from the vanishing gradient problem. Another advantage is that ReLU is cheap to compute which leads to a smaller computational complexity and causes in practice a faster convergence. Unfortunately, ReLU causes the problem of "dead neurons", meaning that if neurons are not activated initially, they will never be activated during the learning process.

4. **LeakyReLU / PReLU**

   $a(x) = \max(0.01x, x)$

   LeakyReLU has similar properties than ReLU with the difference that LeakyReLU does not suffer from the "dead neuron" problem since the gradient is never 0. Moreover, LeakyReLU is more balanced and might therefore learn faster than ReLU. However, the disadvantage of LeakyReLU in contrast to ReLU is that the output is not sparse anymore.

   In LeakyReLU, the slope is pre-defined to 0.01. PReLU is a variant of LeakyReLU that allows the neural network to determine the slope (learnable parameter).

5. **ELU**

   $$a(x) = \begin{cases} x, & x \geq 0 \\ \beta(e^x - 1), & x < 0 \end{cases}$$

   As PReLU, also ELU contains a learnable parameter β that defines the slope on the negative side. In comparison to LeakyReLU/PReLU, ELU contains a negative saturation regime, meaning that very negative numbers slowly reach a gradient of 0 which leads to a sparser representation as in LeakyReLU. It has therefore all the advantages of ReLU and LeakyReLU except that the computational complexity is higher.

6. **Softmax**

   $$a(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

   The softmax function is primarily used after the last layer of a neural network which is acting as classifier. Softmax squashes the output scores of a unit j into a $[0, 1]$ range and the total sum over all K units is equal to 1. This allows us to express the input as a discrete probability distribution.

Considering all the previously explained activation functions, ReLU might be the best activation function to start with whereas sigmoid offers the least advantages.

2.3 Loss Functions

A loss function (also: cost function) evaluates the performance of an algorithm on the given dataset. It measures the divergence between a prediction produced by an algorithm and the ground-truth label. The overall loss represents the average of all losses calculated individually for every sample in a dataset,

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i \tag{2.10}$$

with N the number of samples. The following paragraphs outline the most common loss functions for different types of tasks.

Classification. In classification, we assume that every input sample x_i has exactly one correct label y_i such as the class of an object shown in an image. The underlying dataset might consist of samples from only two different groups (binary) or multiple classes. The most common cost function for classification tasks is the cross-entropy loss. It originates from information theory and measures essentially the difference between two probability distributions p and q,

$$H(p,q) = -\sum_{x} p(x)\log(q(x)) \tag{2.11}$$

where p represents the true distribution and q the estimated distribution. Given the above equation, the cross-entropy loss $\mathcal{L}_i$ for one sample x_i is then as follows,

$$\mathcal{L}_i^{CE} = -\sum_{c=1}^{C} t_{i,c}\log(f(x_i;\theta)_c) \tag{2.12}$$

with $t_i \in \{0,1\}^C$ a one hot encoding vector containing zeros everywhere except on the position of the input's true class $\tilde{c} \in \{1,...,C\}$ and $f(x_i;\theta)_c$ the output score of the neural network for class c given x_i as input. It is important to mention that, before computing the cross-entropy loss, a softmax activation is applied to the output of the last fully-connected layer in order to normalize the FC output into a probability distribution. Note, that the output of the last FC layer is often called "logits" (unnormalized log probabilities of the classes). The combination of softmax activation and cross-entropy loss is often referred to as *Softmax loss* (but many calls it simply cross-entropy loss even though a softmax activation is included). In the case of multi-class classification problems, the cross-entropy loss is also often called the *categorical cross-entropy loss* or *multi-class cross-entropy loss*. For binary problems ($C = 2$), the equation above simplifies as follows,

$$\mathcal{L}_i^{BCE} = -t_{i,1}\log(f(x_i;\theta)_1) - (1-t_{i,1})\log(1-f(x_i;\theta)_1) \tag{2.13}$$

with $t_{i,1} = 1$ if $\tilde{c} = 1$, otherwise 0. Instead of using softmax as activation function, the binary cross-entropy loss is used in combination with a sigmoid activation function.

Regression. A regression function predicts a continuous, real valued quantity instead of a label. The mean square error (MSE)/L2 loss or mean absolute error (MAE)/L1 loss are the most common cost functions for regression tasks. The L2 loss for a single sample x_i is computed by using the squared L2 norm between the predicted value $\hat{y}$ and the target variable y,

$$\mathcal{L}_i^{MSE} = ||\hat{y}_i - y_i||_2^2 \tag{2.14}$$

and the L1 loss for x_i is calculated by taking the L1 norm between $\hat{y}$ and y,

$$\mathcal{L}_i^{MAE} = ||\hat{y}_i - y_i||_1. \tag{2.15}$$

Optimizing a network for a regression task is often more difficult than for a classification task. Therefore, it is recommended, if possible, to transform the regression task into a classification task by quantizing the output into bins.

Image Generation. Typical cost functions used for training a model that is able to generate images such as a (variational) autoencoder are the reconstruction loss, perceptual loss and KL-divergence loss.

The *reconstruction loss* measures the error between the pixel intensities of the generated image $\hat{x}$ and the input image x by using either binary cross-entropy (BCE) or squared L2 norm,

$$\mathcal{L}_i^{rec-BCE} = -\sum_{k=1}^{M} x_k \log(\hat{x}_k) + (1 - x_k)\log(1 - \hat{x}_k), \tag{2.16}$$

$$\mathcal{L}_i^{rec-L2} = \frac{1}{2}\sum_{k=1}^{M} (\hat{x}_k - x_k)^2, \tag{2.17}$$

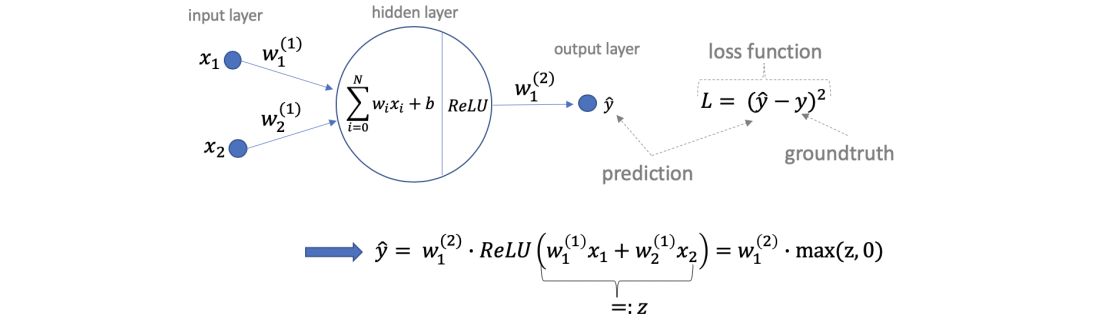
with M the number of pixel values in x.

The perceptual loss [78] is often used as surrogate for the reconstruction loss and compares high-level discrepancies between images such as content or style differences. It is based on a neural network that has been already trained on the task of recognizing objects (e.g. trained on ImageNet) and is therefore capable of identifying high-level features. The perceptual cost function computes the difference (L2 norm) between the features of the original image x and the features of the generated image $\hat{x}$ given the pre-trained network as feature extractor. Using this loss function for training a generative model encourages the output image $\hat{x}$ to be perceptually similar to the target image x but does not force them to match exactly [78].

Similar to the cross-entropy cost function, the Kullback-Leibler (KL) divergence measures essentially the similarity between two distributions,

$$D_{KL}(p||q) = \sum_{j=1}^{N} p(x_j) \log\left(\frac{p(x_j)}{q(x_j)}\right) \tag{2.18}$$

with p the real distribution and q the approximation. It is specifically useful when performing a direct regression as required for a variational autoencoder when minimizing the distance between the encoder output and a normal distribution.



Back-propagation

$$\frac{\partial L}{\partial w_1^{(2)}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1^{(2)}} = \frac{\partial (\hat{y} - y)^2}{\partial \hat{y}} \cdot \frac{\partial (w_1^{(2)} \cdot \max(z, 0))}{\partial w_1^{(2)}} = \frac{\partial (\hat{y} - y)^2}{\partial (\hat{y} - y)} \cdot \frac{\partial (\hat{y} - y)}{\partial \hat{y}} \cdot \max(z, 0) = 2(\hat{y} - y) \cdot \max(z, 0)$$

$$\frac{\partial L}{\partial w_1^{(1)}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \max(z, 0)} \cdot \frac{\partial \max(z, 0)}{\partial z} \cdot \frac{\partial z}{\partial w_1^{(1)}} = 2(\hat{y} - y) \cdot w_1^{(2)} \cdot \frac{\partial \max(z, 0)}{\partial z} \cdot x_1$$

$$\frac{\partial L}{\partial w_2^{(1)}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \max(z, 0)} \cdot \frac{\partial \max(z, 0)}{\partial z} \cdot \frac{\partial z}{\partial w_2^{(1)}} = 2(\hat{y} - y) \cdot w_1^{(2)} \cdot \frac{\partial \max(z, 0)}{\partial z} \cdot x_2$$

with $\frac{\partial \max(z, 0)}{\partial z} = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$

Figure 2.6: Example of back-propagation applied to a multi-layer perceptron (MLP) with 1 hidden layer for computing the gradients of the weights w_i. The MLP receives a two-dimensional vector x as input and outputs a prediction scalar $\hat{y}$ (top). Given the loss function (top right) the gradients of the parameters $w_1^{(1)}, w_2^{(1)}$ and $w_2^{(1)}$ are calculated using back-propagation and the chain-rule of calculus (bottom). The colors signalize the affiliation between the different terms.

2.4 Optimization

Optimization algorithms are responsible for minimizing the losses of an ANN by updating the weights accordingly. Due to the non-linearity and the depth of a neural network, loss functions become non-convex. Even though the ultimate objective of a neural network training is to find the global optima, researchers have found that finding a local optimum with a reasonable low error is sufficient. Neural networks are commonly trained with iterative, gradient-based optimizers (gradient is used to descend the loss function) with back-propagation as a tool for computing the gradient. Most of the nowadays common optimization methods for ANNs are often improvements or refinements of stochastic gradient descent (SGD). The following paragraphs introduce back-propagation, stochastic gradient descent and its variants.

Back-Propagation. The back-propagation algorithm enables the error produced by the ANN to flow backward through the network to compute the gradient. In particular, back-propagation computes the chain-rule of calculus with a very efficient order of operations. The chain-rule states, that the derivative of $f(g(x))$ is given as $f'(g(x)) \cdot g(x)$, i.e.

$$\frac{dy}{dx} = \frac{dg}{dz}\frac{dz}{dx} \tag{2.19}$$

with f, g real valued functions, $z = g(x)$, $y = f(g(x))$ and x a scalar. The same

holds for vector valued inputs,

$$\frac{\delta y}{\delta x_i} = \sum_j \frac{\delta y}{\delta z_j} \frac{\delta z_j}{\delta x_j}. \tag{2.20}$$

Figure 2.6 shows an example of back-propagation applied to a multi-layer perceptron with 1 hidden layer, ReLU as activation function and mean squared error as loss function with $x \in \mathcal{R}^2$ the input and $y \in \mathcal{R}$ the output. The back-propagation algorithm is used to compute the gradients for the parameters $\theta = (w_1^{(1)}, w_2^{(1)}, w_1^{(2)})$ using the chain-rule. In the next paragraphs we describe common optimization methods that employ the gradients for updating the weights (perform learning) in an ANN.

Stochastic Gradient Descent. Vanilla gradient descent (also: batch gradient descent) uses the gradient of the loss function L with respect to the parameters θ for the entire training dataset,

$$\theta = \theta - \alpha \cdot \nabla_\theta L(\theta) \tag{2.21}$$

with α the learning rate. Updating the weights in this way is, however, not realizable for large datasets since the computation would be very slow and only a fraction of the dataset would fit into memory. *Stochastic gradient descent* (SGD) solves this issue by updating the weights using only one sample from the dataset,

$$\theta = \theta - \alpha \cdot \nabla_\theta L(\theta; x, y) \tag{2.22}$$

with (x, y) an input-output pair of the dataset. Considering the example from Figure 2.6 we can now conduct an update step with SGD and back-propagation after performing a forward pass with the following exemplary input-output pair $(x, y) = ((1, 2), 13)$ and the initial weights $w_1^{(1)} = 1, w_2^{(1)} = 2, w_1^{(2)} = 3$

1. Forward pass

$$\begin{aligned} z &= w_1^{(1)} x_1 + w_2^{(1)} x_2 = 5, \\ \hat{y} &= w_1^{(2)} \cdot \max(z, 0) = 15 \\ L &= (\hat{y} - y)^2 = 4, \end{aligned}$$

2. Back-propagation

$$\begin{aligned} \frac{\delta L}{\delta w_1^{(2)}} &= 2 \cdot (\hat{y} - y) \cdot \max(z, 0) = 20 \\ \frac{\delta L}{\delta w_1^{(1)}} &= 2 \cdot (\hat{y} - y) \cdot w_1^{(2)} \cdot x_1 = 12 \\ \frac{\delta L}{\delta w_2^{(1)}} &= 2 \cdot (\hat{y} - y) \cdot w_1^{(2)} \cdot x_2 = 24 \end{aligned}$$

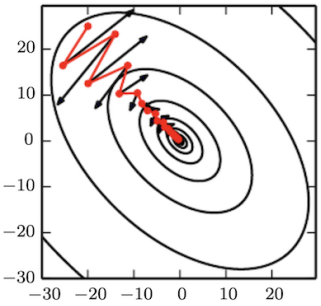


Figure 2.7: Exemplary progress of SGD (black lines) and SGD with momentum (red). Picture is taken from [61].

3. SGD update with $\alpha = 0.001$

$$\tilde{w}_1^{(2)} = w_1^{(2)} \cdot \alpha \frac{\delta L}{\delta w_1^{(2)}} = 3 - 0.001 \cdot 20 = 2.98$$
$$\tilde{w}_1^{(1)} = w_1^{(1)} \cdot \alpha \frac{\delta L}{\delta w_1^{(1)}} = 1 - 0.001 \cdot 12 = 0.988$$
$$\tilde{w}_2^{(1)} = w_2^{(1)} \cdot \alpha \frac{\delta L}{\delta w_2^{(1)}} = 2 - 0.001 \cdot 24 = 1.976$$

In practice, each parameter update is performed using a few samples (mini-batch) from the training set instead of using only one sample. This guarantees a more stable convergence since the variance between different parameter updates is reduced. The learning rate α is a hyper-parameter that needs to be defined beforehand. However, finding a good learning rate is not trivial. A learning rate that is too small leads to a very slow convergence and if α is too large the network might never converge or the loss function might fluctuate around the minimum. In the deep learning community, it is common to specify a learning rate schedule that defines when the learning rate is reduced. The scheduling could be, for instance, based on the amount of iterations.

SGD with Momentum. Even though SGD is a popular optimization strategy, an ANN optimized with SGD converges fairly slow. SGD gets often trapped in sub-optimal local minima and oscillates strongly in areas with long and narrow ravines. *SGD with momentum* tackles this problem by using the moving average gradient instead of the immediate gradient at each update,

$$\begin{aligned} v &= \gamma v + \alpha \cdot \nabla_\theta L(\theta; x, y) \\ \theta &= \theta - v \end{aligned} \tag{2.23}$$

with v the velocity vector and $\gamma \in (0, 1]$ a parameter that regulates for how many iterations the previous gradients are included into the current update. This strategy helps to accelerate SGD in the relevant direction and to reduce the oscillations. Figure 2.7 illustrates the effect of momentum. While approaching a minimum, the immediate gradient decreases and SGD slows down. SGD with momentum, on the other hand, averages the gradient over several time steps

leading to a less drastic deceleration. Moreover, SGD with momentum enables the algorithm to skip over very sharp minima that in many cases represent overfitting.

Besides skipping undesirable sharp minima, SGD with momentum might, at the same time, miss desirable optima if the momentum is too high. *SGD with nesterov momentum* addresses this issue by first estimating the future location and then correcting the velocity vector based on the future location,

$$\begin{aligned} v &= \gamma v + \alpha \cdot \nabla_\theta L(\theta - \gamma v; x, y) \\ \theta &= \theta - v. \end{aligned} \tag{2.24}$$

AdaGrad. The optimization methods described so far use a constant learning rate for all parameters. AdaGrad [46] adapts the learning rate for each parameter individually depending on their importance. The update for a parameter θ_i is performed as follows,

$$\theta_i = \theta_i - \frac{\alpha}{\sqrt{G_{ii}} + \epsilon} \nabla_\theta L(\theta_i; x, y) \tag{2.25}$$

with $G_{ii} = G_{ii} + (\nabla_\theta L(\theta_i; x, y))^2$ the sum of the squares of the gradients w.r.t θ_i up to the current time step and ϵ a small smoothing term to avoid division by zero. Individually adapting the learning rate has the advantage that the global learning rate needs less manual tuning and interacts more as a general indicator. AdaGrad has shown to work very well in settings with sparse gradients. However, due to the growing division term, the learning rate is constantly decreasing which results in slow training up until no additional knowledge is acquired. *AdaDelta* [193] extends AdaGrad by restricting the number of accumulated past gradients to a fixed size.

RMSProp. Similar to AdaDelta, RMSProp has been developed for eliminating the problem of a diminishing learning rate which is observed in AdaGrad. In particular, RMSProp divides the learning rate by an exponentially decaying average of squared gradients,

$$G_{ii} = \gamma G_{ii} + (1 - \gamma) \cdot g_t^2 \tag{2.26}$$

with γ the decay rate chosen usually around 0.9.

Adam. Adam (adaptive moment estimation) [85] is often seen as a combination of RMSProp and SGD with momentum. Besides storing the exponentially decaying average of squared previous gradients (second momentum), Adam also saves the exponentially decaying average of previous gradients (first momentum),

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \end{aligned} \tag{2.27}$$

with $g_t = \nabla_\theta L(\theta; x, y)$ and β_1 and β_2 hyper-parameters that are usually set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$, respectively. v_t is used for scaling the learning rate (like RMSProp) and m_t for accelerating the algorithm (like SGD with momentum) at time point t. Both parameters are initialized with 0 ($m_0 = v_0 = 0$). To prevent a division by a very small number in early iteration, the Adam optimizer contains

additionally a bias correction,

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}. \end{aligned} \tag{2.28}$$

This results in the following update

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.29}$$

The Adam optimizer converges quickly and is, besides SGD with momentum, nowadays the most frequently used optimization approach.

2.5 Established Network Architectures

Since the boom of neural networks, there have been many different architecture proposals where some of them are particularly successful in solving image and video understanding tasks. In this section, we provide a brief description of the most common network architectures in computer vision.

AlexNet. AlexNet [90] is considered one of the most important architectures which advanced the current deep learning era. In 2012, AlexNet won the ImageNet ILSVRC 2012 competition by a large margin and was since then applied to many diverse visual understanding tasks. It consists of in total 8 layers with 5 convolutional layers and 3 fully-connected layers with max pooling layers after the 1st, 2nd and 5th convolutional layer and ReLU activation functions after every conv layer. The network has in total 62.3 million parameters where the FC layers contain most of them. Compared to previous neural network architectures such as LeNet [96], AlexNet contains many more parameters, but thanks to the advancements in GPU capacities at that time it was possible to train the network within 5 to 6 days on the large-scale dataset ImageNet [142] using 2 GPUs. CaffeNet [77] is a variant of AlexNet that essentially has the same architecture, but was build to be trained on only one instead of 2 GPUs (since also AlexNet can be run nowadays on 1 GPU, CaffeNet and AlexNet are practically the same network).

VGG. VGG [149] was introduced in 2014 and outperformed AlexNet in the ILSVRC 2014 challenge. A characteristic of VGG are the relatively small convolutional filters (3×3) on the first few layers. This modification allows the usage of more layers (deeper network) since the small convolutional filters contain less parameters than the ones used in AlexNet. Succeeding architectures such as ResNet [67] adopted the idea of using multiple 3×3 convolutions in series. VGG consists of in total 16 (or 19) layers with 13 (or 16) convolutional layers, 3 fully-connected layers, 5 pooling operations and every hidden layer is equipped with ReLU as nonlinear activation function. In contrast to previous and also following architectures, VGG contains a lot of parameters ($>$ 135 million) leading to long training times.

Inception. The idea of Inception-V1 [156] (also called GoogleNet) which was introduced in 2014, was to go wider instead of only deeper. The concept emerged

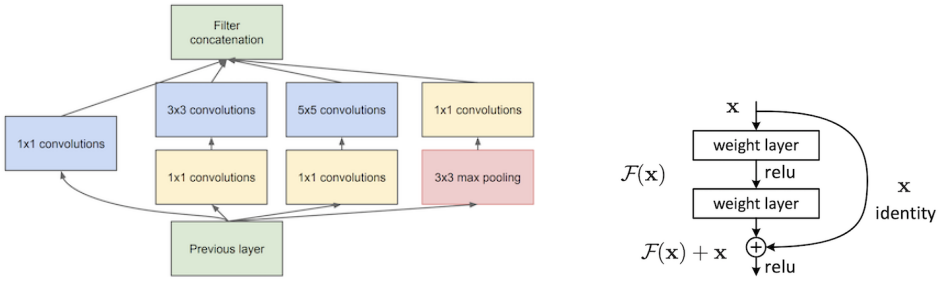


Figure 2.8: *Left*: Illustration of an inception module with dimension reductions. Image taken from [156]. *Right*: Depiction of a residual block in ResNet. Image taken from [67].

from the problem that it is difficult to find the right filter size since objects can occupy either the full image or only a fraction. The developers of GoogleNet introduced an inception module that applies several filters with different sizes on the same level. Figure 2.8 illustrates the structure of the new module. The authors of [156] additionally propose to perform 1×1 convolutions to reduce the input channels and therefore the computational complexity. The original GoogleNet contains 9 of those inception modules with in total 22 layers and a pooling operation at the end of the last inception module. Given that GoogleNet is a relatively deep network, the authors included two auxiliary classifiers in the middle of the network. The total loss is then computed using the two auxiliary losses and the final loss at the end of the network. InceptionV1 was further improved by introducing more updates to increase the accuracy and for reducing the computational complexity. These improved networks are called Inception-V2, Inception-V3, Inception-V4 and Inception-ResNet. More information can be found in [157] and [155].

ResNet. ResNet [67] introduces in 2016 residual building blocks for addressing the vanishing gradient problem. A residual block makes use of shortcut connections between two convolutional layers. Instead of learning the mapping $x \rightarrow \mathcal{F}(x)$, a residual block learns the mapping from $x \rightarrow \mathcal{F}(x) + x$ (identity connection). Figure 2.8(right) illustrates the structure of a residual block. ResNet18 consists of 17 convolutional layers with a filter size of 3×3 (as VGG), 1 fully-connected layer and only 2 pooling operations. Due to the reduced number of fully-connected layers, ResNet18 contains only 11 million parameters. Diminishing the vanishing gradient problem by using identity connections enabled the usage of much deeper neural networks. The original ResNet contains 18 layers, but it is common nowadays to employ ResNet50 or ResNet101 with 50 or 101 layers, respectively.

C3D. C3D [162] and also the networks described in the subsequent paragraphs represent 3D networks with 3D convolutions. C3D has been already introduced in 2015 but gained only recently more attraction due to the higher demand in efficient deep learning algorithms for video understanding tasks. The network consists of 8 3D convolutional layers, 2 fully-connected layers and pooling operations after the 1st, 2nd, 4th, 6th and 8th conv layer. The authors of [162] obtained the best results when using small convolutional filter with the size of $3 \times 3 \times 3$ throughout the entire network. C3D outperformed previous work on

action recognition, action similarity labeling and scene and object recognition at the time of publication.

I3D. I3D [20] has been proposed in 2018 and represents a two-stream 3D inflated network. Instead of developing a new architecture for video understanding, the authors of [20] propose to inflate 2D CNNs that have shown superior performances on image understanding tasks. All 2D convolutional layers and 2D pooling operations are simply inflated to 3 dimensions by endowing them with a temporal dimension. In addition to the proposed inflation, I3D is also tested on a two-stream configuration with one 3D network pre-trained on RGB inputs and the other on optical flow. The streams are trained independently from each other but combined during testing by averaging their predictions. For the experiments, Carreira and Zisserman [20] employ an ImageNet pretrained Inception-V1 as base network.

3D ResNet. As the name suggest a 3D ResNet is the 3D version of a 2D ResNet network. Tran *et al.* propose in [163] two variants. The first version (R3D) simply transforms the 2D operations in a ResNet network into 3D filters. The second version (R(2+1)D) approximates 3D convolutions by separating them into two steps: a 2D convolution for spatial modeling followed by a 1D convolution for extracting temporal information. Experiments have shown that R(2+1)D results in lower errors on action recognition than R3D with the same amount of layers and parameters. This is most likely due to the higher complexity in R(2+1)D given that the amount of activation functions (nonlinearities) can be doubled in R(2+1)D.

Chapter 3

Self-Supervised Representation Learning

In the age of big data, problems have shifted from lacking training data to now having lots of it but lacking tedious manual annotations. Deep learning, which benefits from large volumes of training data, has therefore spurred new interest in unsupervised techniques. Especially self-supervision, where the feature representation of visual data is learned indirectly by solving a surrogate task (also called pretext task), has recently shown great potential. Previous works have successfully addressed image understanding tasks but are still lacking satisfactory performances on the more difficult task of analyzing videos. After providing a description of self-supervised learning and summarizing the most influential self-supervised methods, we propose a new surrogate task that exploits temporal information from videos in this chapter. Our model learns a detailed sequence representation that is used to tackle video understanding tasks. In addition, we introduce a multi-task network for learning a powerful representation on unlabeled images and videos simultaneously. Experiments on image and video understanding tasks demonstrate the wide applicability of our models and their superior performance in comparison to previous methods.

3.1 Self-Supervised Learning in a Nutshell

Convolutional neural networks (CNNs) have demonstrated that they are able to learn powerful visual representations from large amounts of tediously labeled training data [90]. However, since visual data is cheap to acquire but costly to label, there has recently been great interest in learning compelling features from unlabeled data. For that matter, self-supervised learning (SelfSL) [116, 98, 54, 10, 122, 36, 114, 123, 103, 127, 94, 29, 60, 145] is nowadays a popular technique to learn visual representations without requiring manually labeled training data. The desired visual features are indirectly learned by solving a surrogate task where its target values can be obtained automatically from the data. In other words, we still train a CNN with a supervised loss function, however, that function does not evaluate the performance on the desired task (due to the missing labels), but on an artificially constructed surrogate task. In [122], for instance, Noroozi and Favaro randomly permute parts of an image like a jigsaw puzzle and ask the net-

work to recover the original configuration. The index of the applied permutation serves as the target value and is automatically defined without requiring manual annotations. Even though the network has not been directly trained on a desired task, e.g. image classification, the learned representation carries good semantic and structural knowledge of objects. Thus, SelfSL exploits visual data like images [180, 36, 94, 29, 197, 170, 129, 38, 60] or videos [116, 98, 54, 103, 128, 179, 180, 15], but also text [127] or audio [126] as source of information. The representation learned through a surrogate task (pre-training) can be afterwards transferred to a variety of challenging visual understanding tasks such as image classification [122, 124, 60, 123, 16], human pose estimation [11, 10, 114, 15], image segmentation [122, 124, 16, 14, 60] or action recognition [16, 98, 116, 187, 54]. The transfer is performed by re-training (fine-tuning) the obtained representation on the desired task (also called *downstream task*) using all labels of the corresponding dataset. We expect the fine-tuned model to show higher performances on the downstream task than a model that has only been trained on the final task since the pre-training provides prior knowledge about the data structure. The more semantic and structural knowledge is already captured during the pre-training, the higher the performance boost on the final task.

3.2 Overview of Recent Works

Lately, a considerable amount of SelfSL techniques have been introduced in combination with deep learning. They have proven to be an effective tool for learning powerful features with unlabeled data in several areas such as language modelling or computer vision. In the vision community, self-supervised deep learning has been established in 2015 where mainly image understanding tasks such as image classification, object recognition or geometry estimation were addressed [44, 36, 179, 2]. Due to their success many more SelfSL methods have been published since then and applied to a bigger variety of downstream tasks including video understanding issues. This thesis is mainly concerned with methods that analyze videos. However, since self-supervised methods for image understanding have greatly influenced the research for video analysis this section firstly provides a short summary of the most influential image-based self-supervised works followed by video-based approaches. A curated list of famous self-supervised papers can be found on github[1]. The figures displayed in this section provide assistance to understand the described methods better and are all taken from the corresponding papers.

3.2.1 Image-Based Methods

Common tasks for images addressed in self-supervised learning are image classification, object recognition, pose estimation, image segmentation, object detection and geometry estimation using the benchmark datasets Olympic Sports [121], Leeds Sports [80], NYU Depth [120], ImageNet [142], Pascal VOC [50, 51], CIFAR-10/CIFAR-100 [89] and Places205 [200]. We will now summarize five of

[1] `https://github.com/jason718/awesome-self-supervised-learning`

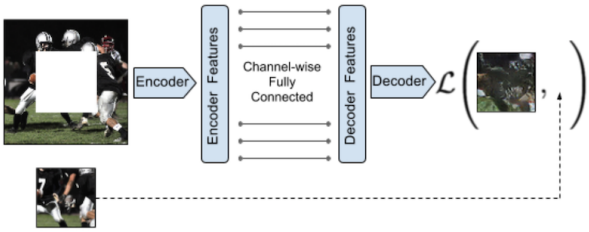


Figure 3.1: Model overview of **Inpainting** [129]. The encoder receives an image with missing parts as input and the decoder generates the missing part. The weights are updated using an L2 loss given the generated and original patch.

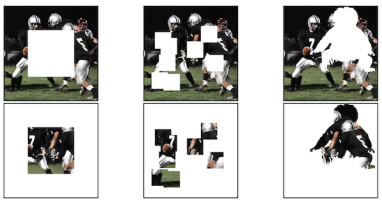


Figure 3.2: Depiction of the different region masks applied to input images for **Inpainting**.

the most influential self-supervised methods which use the learned representation for solving image understanding tasks.

Inpainting [129]. In 2016 Pathak *et al.* [129] proposed a self-supervised method based on a generative model. The pretext task in generative modeling is to reconstruct the original input using an encoder-decoder architecture while learning a powerful latent image representation [169, 197, 40, 42, 129]. Pathak *et al.* propose to learn meaningful features by asking an autoencoder to generate the contents of an arbitrary image region conditioned on its surroundings. An encoder receives an image with missing parts as input and produces a latent feature representation of that input. Then, the decoder's task is to generate the missing region given the latent feature representation. The weights of the autoencoder are updated using an L2 reconstruction loss and the original image region. Figure 3.1 and 3.2 display the training strategy and possible region masks that are applied before inserting the image into the network. In order to succeed at this surrogate task, the network needs to understand the content of the image and produce reasonable assumptions about the missing part of the input. Pathak *et al.* employ the benchmark datasets ImageNet [142] and Paris StreetView [37] for training the network with the unsupervised surrogate task. The transferability of the learned features to image understanding tasks are evaluated by fine-tuning the encoder features (AlexNet architecture) on the desired downstream task. Inpainting [129] has shown state-of-the-art results at the time of publication in image classification and segmentation on Pascal VOC.

Jigsaw [122, 124]. Jigsaw [122] by Noroozi and Favaro and their subsequent method Jigsaw++ [124] have been very successful and widely cited approaches for self-supervised learning. Moreover, our methods introduced in Section 3.3 and 3.4 are influenced by this strategy. As the approaches presented in [36, 123, 119, 29, 84], Jigsaw and Jigsaw++ belong to the type of methods that extract multiple patches from an image and ask the network to predict their relationship. In particular, Noroozi and Favaro propose in [122] to use Jigsaw puzzles for developing a visuo-spatial representation of objects in the context of CNNs. An image is divided into 9 tiles (3x3 grid) which are shuffled afterwards, given a random permutation from a pre-defined set of permutations, and fed into a siamese network with AlexNet [90] architecture. The fully-connected (fc)

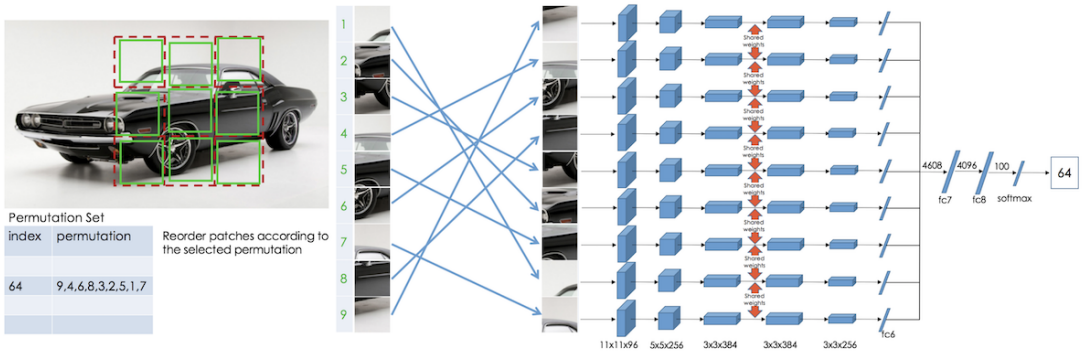


Figure 3.3: Model overview of Jigsaw [122]. An image is divided into 9 parts and fed into the siamese network after randomly shuffling the order of the patches. The features of the patches are concatenated, and a classifier predicts the index of the applied permutation.

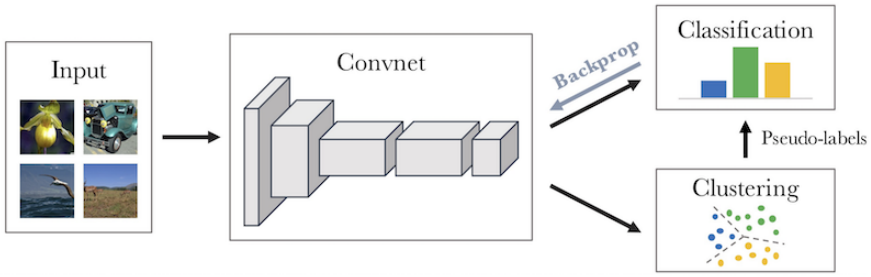


Figure 3.4: Model overview of **DeepCluster** [17]. The network is trained on a multi-class classification task using the cluster assignments as pseudo-labels. The method alternates between updating the network weights and re-clustering the training set given the present feature representation of the network.

features of all 9 patches are concatenated and given as input to the second fc layer. The task of the network is then to correctly predict the index of the applied permutation using a final fc layer as classifier. Figure 3.3 shows the generation of a puzzle and the network architecture. After training the network for 350k iterations with the jigsaw puzzle task, Noroozi and Favaro evaluate the learned features by fine-tuning on image classification, detection and segmentation using the Pascal VOC dataset and on the ImageNet classification task. The method achieved state-of-the-art results at the time of publication (2017). Instead of fine-tuning the self-supervised model on the downstream task, Noroozi and Favaro proposed one year later in [124] to decouple the structure of the self-supervised model from the final task-specific, fine-tuned model. In particular, they first train a (bigger) network on a self-supervised method such as Jigsaw and employ the trained features for assigning pseudo-labels (pseudo object classes) to every image of the ImageNet dataset via clustering. Then, the target network (AlexNet architecture) is pre-trained using a multi-task classification loss and the previously assigned pseudo-labels. Until this point no supervision is required. For the final evaluation, the target network is trained on the desired downstream task such as image classification or segmentation using the labels provided by the datasets.

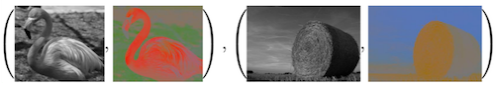


Figure 3.5: Visualization of the **Colorization** approach [196]. The model receives a gray-scale image as input and predicts the color for every pixel.

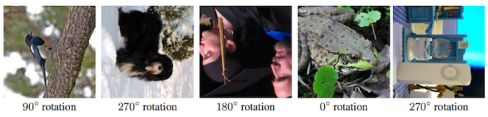


Figure 3.6: Exemplary depiction of the applied transformation to input images for training **RotNet** [60].

DeepCluster [17]. Several approaches [17, 190, 185, 18] have employed clustering to learn an unsupervised image representation. In 2018, Caron *et al.* presented DeepCluster [17] as a novel unsupervised approach based on clustering with impressive results. Similar to image classification, the SelfSL model in [17] predicts the class of an image. However, instead of using the ground-truth classes (supervised), Caron *et al.* employ cluster assignments as pseudo-labels. The approach alternates between clustering the image descriptors with k-means and updating the weights of the network by predicting the cluster assignments. The features for performing the clustering are extracted from the network which is trained. Figure 3.4 summarizes the training procedure.

Colorization [94]. Given a grayscale image, the goal of colorization [93, 196, 197, 26, 71, 1, 94] is to hallucinate a plausible color for every pixel in that image. In 2016, two concurrent works [196, 93] have proposed to use colorization as a pretext task to learn an image representation without requiring any annotations. Fortunately, training data is easily accessible given the huge amount of unlabeled color images available on the internet. Therefore, recent works [93, 196, 94] have proposed to employ large datasets of colored images to train a CNN by firstly removing color information from the input and secondly asking the network to predict that extracted component (see Figure 3.5). Larsson *et al.* investigate in [94] different types of losses for training the networks on the colorization task. In particular, they consider a regression loss for L*a*b color values as well as a KL divergence loss for hue/chroma histograms. Larsson *et al.* [94] perform experiments with different types of networks: AlexNet, VGG-16 and ResNet-152. The best results on image understanding tasks are achieved with the hue/chroma histograms and the ResNet-152 architecture. The results obtained with the AlexNet structure are also the most competitive among AlexNet models of previous works.

RotNet [60]. Modifying an image geometrically while the semantic content stays unaltered is another way of teaching the network to learn meaningful features without requiring manual annotations. In RotNet [60] the entire input image is randomly rotated by a multiple of 90° (90°, 180°, 270°, 360°) before inserting the distorted image to the network (see Figure 3.6). After a forward pass, a classifier needs to predict the index of the applied rotation. Experiments have shown that RotNet achieved state-of-the-art results at the time of publication (2018) for image classification on ImageNet, Pascal VOC and Places205 and image detection/segmentation on Pascal VOC.

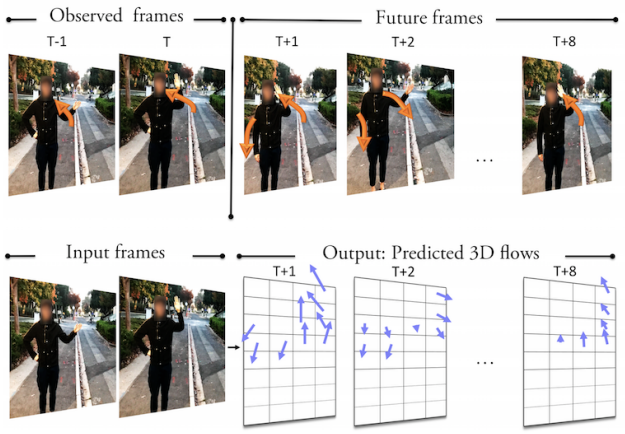


Figure 3.7: Depiction of the **MotionDynamics** strategy proposed in [103]. The model predicts a sequence of basic motions given two input frames.

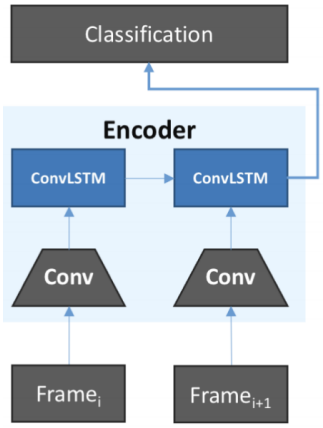


Figure 3.8: Architecture of [103] for transferring the learned representation to the downstream task action recognition.

3.2.2 Video-Based Methods

After the success of applying SelfSL methods to image understanding tasks, several unsupervised video representation learning approaches have been proposed. Temporal information provided in videos such as motion or the ordering of frames is employed to construct novel surrogate tasks. Common video understanding tasks that are presently tackled by SelfSL methods are action recognition, action segmentation or tracking. Moreover, the trained representation is broadly applicable and often also used to address video and image understanding tasks simultaneously. The following paragraphs describe four of the most influential self-supervised methods for video representation learning.

MotionDynamics [103]. Many self-supervised methods employ motion information from videos to train a representation that carries good semantic and structural knowledge of objects and movements [103, 177, 146, 134]. Among them also the approach proposed in 2017 by Luo *et al.* [103] where the network learns to predict a sequence of atomic long-term 3D flows (see Figure 3.7). The network consists of an LSTM based encoder-decoder architecture and receives either a pair of RGB, depth or RGB-D samples as input. The learned latent representation can then be used as motion feature for addressing action recognition. Figure 3.8 demonstrates the network structure of the encoder for recognizing activities. The decoder is omitted for the downstream task.

VideoColorization [171]. Inspired by self-supervised image colorization methods, Vondrick *et al.* [171] teach a network to colorize gray-scale videos (surrogate task) in order to learn a robust feature representation for tackling video understanding tasks. The network consists of a ResNet-18 followed by a 3D spatiotemporal convolutional network. The overview of the model is shown in Figure 3.9. Given a gray-scale reference and gray-scale target image, the CNN computes for every location a feature embedding and the model points from the target frame into the reference frame embeddings using softmax similarity. For

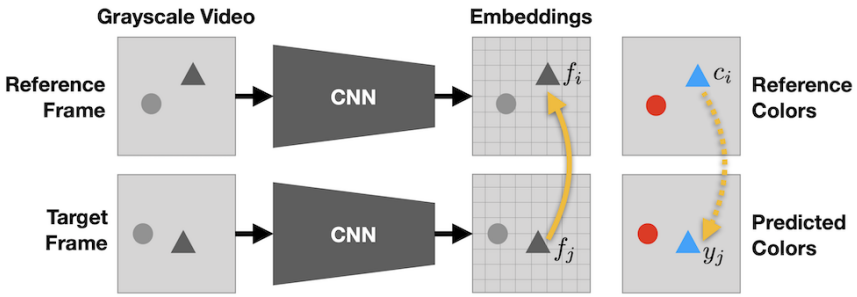


Figure 3.9: Model overview of **VideoColorization** [171]. Given two gray-scale images, the model establishes for every point in the target image a correspondence to a point in the reference frame using softmax similarity. The model is trained by comparing the reference color with the predicted color in the target image.

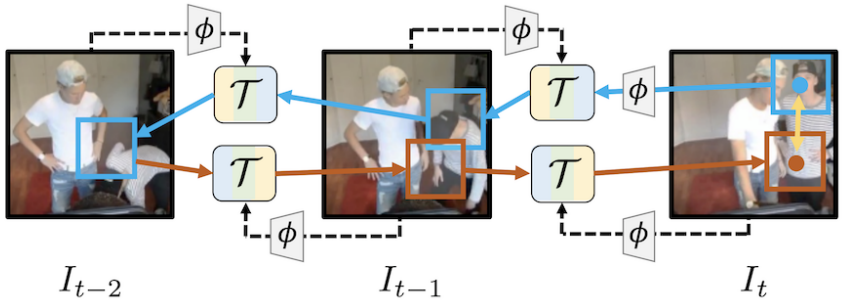


Figure 3.10: Method overview of **CycleConsistency** [181]. A patch is tracked backwards and then forward using a tracker τ and a representation ϕ. The error is computed by comparing the original (right, blue box) with the final tracked patch (right, brown box).

evaluating if the location was tracked correctly, the color of the reference frame is firstly transferred to the associated location in the target frame. Then, the transferred color is compared with the true color of the target frame. The error is computed using a cross-entropy categorical loss after quantizing the color-space into discrete categories. During inference, a fully labeled frame is given as reference and the learned pointer is used to propagate labels throughout the video. For evaluation, Vondrick *et al.* [171] apply their model to video segmentation and pose tracking.

CycleConsistency [181]. In 2019, Wang *et al.* [181] have proposed a self-supervised method for learning visual correspondences by automatically extracting them from videos and turning them into a learning signal. In particular, Wang *et al.* [181] obtain unlimited supervision by tracking patches along a cycle in time (forward and backward) and by using the inconsistency between the starting and ending patches as objective function. Figure 3.10 displays the procedure of the learning method. A random patch is first tracked backwards (right to left, blue box) and then forward (left to right, brown box) by a classic tracking-by-matching method based on a learned encoder ϕ. The training signal is then given by measuring the visual and position-wise differences between the original and final tracked patch. For inference, the first frame of a video is fully labeled, and the encoder's representation is used to compute dense correspondences for propagat-

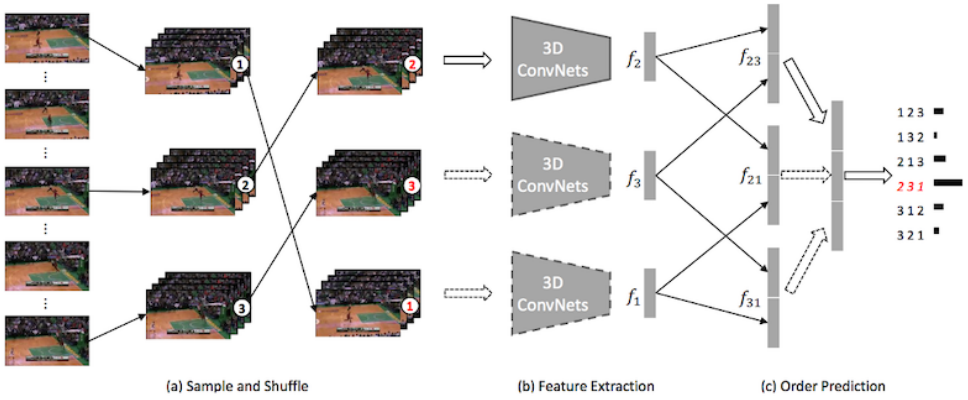


Figure 3.11: Overview of the Clip Order Prediction (**COP**) [187] framework. 3 snippets are extracted from a video, randomly shuffled and inserted into the 3D CNN. After combining the CNN features of all snippets, a classifier predicts the index of the applied permutation.

ing the labels to the rest of the video. The learned model is evaluated on instance and semantic propagation as well as pose keypoint and texture propagation.

COP [187]. Clip order Prediction (COP) [187] is one of the latest approaches that exploit the natural order of frames as source of information to construct a surrogate task [98, 54, 116, 15, 16]. The methods introduced in Section 3.3.1 and 3.4.1 also capitalize on that concept. The idea is related to Jigsaw [122], but instead of shuffling sub-parts of an image, COP [187] permutes several snippets of a video. In particular, the model of [187] is trained by first sampling non-overlapping snippets from a video, where every snippet consists of a pre-defined number of frames. Then, after permuting the snippets in a random order, a 3D CNN is used to extract the features for every snippet (see Figure 3.11). After a pairwise concatenation of the features, fully-connected layers placed on top of the 3D CNN predict the index of the previously applied permutation, i.e. the order prediction is, as in Jigsaw [122], formulated as multi-class classification problem. Thus, the network is updated based on the ability of predicting the applied permutation. Solving this task requires an understanding of the temporal structure of actions and is therefore a valuable pretext task for addressing video understanding tasks. Xu *et al.* [187] employ 3 snippets per sample, leading to $3! = 6$ possible permutations/classes. The model is evaluated by fine-tuning the pre-trained network on action recognition using the benchmark datasets HMDB-51 [91] and UCF-101 [150]. At the time of publication (2019) the approach achieved state-of-the-art results using three different types of 3D CNN architectures (C3D [162], R3D [163], R(2+1)D [164]).

The subsequent sections describe and evaluate our developed self-supervised methods for learning fine-grained video and image representations. Please note that some of the approaches described above have been published after the methods introduced in the following sections.

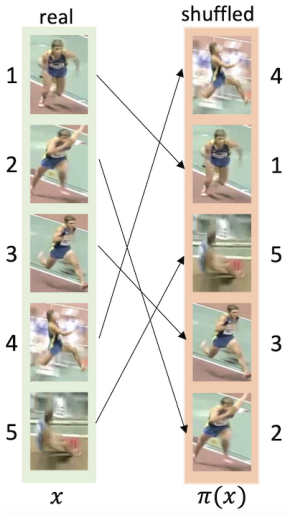


Figure 3.12: Exemplary depiction of temporal shuffling.

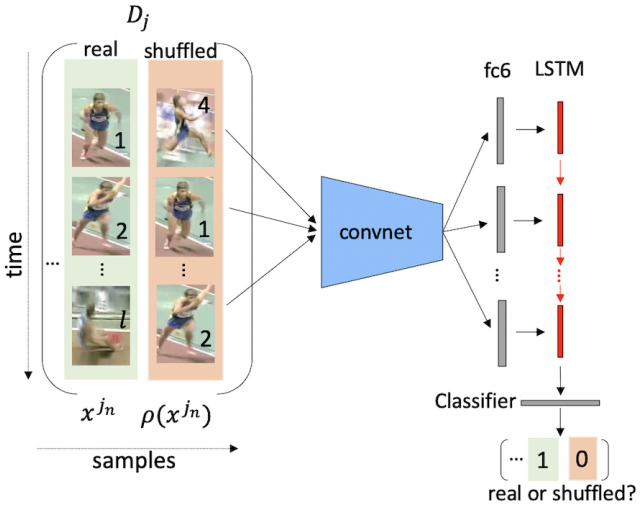


Figure 3.13: Visualization of our LSTM self-supervised training. As input we use real and permuted sequences to learn a representation.

3.3 LSTM Self-Supervision for Videos

The key competence of visual understanding is to recognize structure in visual data. Thus, breaking the order of visual patterns and training a network to recover the structure provides a rich training signal. The framework of permuting the input data and learning a feature representation has been successfully pursued on still images [122, 124, 36, 29, 38] by employing spatial shuffling of images (especially permuting jigsaws). We aim to address not only image but also video understanding tasks and propose to adapt the framework of permuting images to videos. A previous publication by Misra *et al.* [116] employ information from videos by verifying the temporal order of three frames using a triplet siamese network. In this way, the network learns a *single frame* representation, which is, however, not sufficient for determining fine-grained similarities between *sequences* [15]. In order to learn more sequence specific features, we propose to combine learning of single frame features with learning of sequence representations by using a long short-term memory (LSTM) network which we train on permutations of entire sequences.

In the next Section, we present our permutation strategy and evaluate our learned representation in Section 3.3.2 on image and video understanding tasks.

3.3.1 Temporal Permutation

SelfSL represents a type of methods that aims at teaching the network the structure and properties of visual data through a surrogate task. We propose to train the network on distinguishing between real (not permuted) and randomly permuted sequences. Figure 3.12 demonstrates the shuffling of a sequence with

5 frames for the sports activity "Long Jump". Distinguishing between a real sequence and its permutation is challenging, since the model needs to learn the subtle details of the subject's posture together with their change over time. Moreover, a positive and the respective negative sequence are identical except for the order in which postures appear. In order to solve the task satisfactorily, the network needs to disregard background, differences between subjects or lighting changes and solely focus on the change of postures. Thus, our surrogate task is perfectly suited for learning movements and postures, while establishing crucial independence properties.

Training. Let $x = (x_1, x_2, \dots x_l)$ be a sequence with l frames that is to be shuffled by permuting its parts by a random permutation ρ. For training the CNN, mini-batches D_j are composed as

$$\begin{aligned} D_j &= [x^{j_1}, \rho(x^{j_1}), x^{j_2}, \rho(x^{j_2}), \cdots, x^{j_n}, \rho(x^{j_n})] \\ L_j &= [y_1, y_2, \cdots, y_{2n}], \text{ with } y_k = k \bmod 2 \end{aligned} \tag{3.1}$$

where y_k is the label of sequence k in D_j and $2n$ is the number of samples in batch D_j. After a forward pass of D_j, the binary classifier predicts per sample either 1 for real or 0 for shuffled. Figure 3.13 depicts this procedure and the network structure. We optimize the network weights by using the cross-entropy loss and stochastic gradient descent (SGD). For more technical details please see Section 3.5.

Architecture. In order to be comparable with previous work, we employ an AlexNet [90] network structure for the single frame representation with 5 convolutional layers and 1 fully-connected (fc6) layer. Then, the fc6 output of the frames is combined in a recurrent neural network, implemented as LSTM, for learning a sequence representation. The output of the LSTM layer is then processed by a final fully-connected classification layer for predicting real or shuffled.

3.3.2 Experiments

In this section we evaluate our method described above on image and video understanding tasks. We present results on human pose estimation and action recognition. In Chapter 4 we additionally demonstrate the performance of our self-supervised method on an interdisciplinary project (behavior analysis).

Human Pose Estimation. Pose estimation in the vision community means to localize joints (synonym: keypoints) on images/videos such as left hand or right foot and/or to search for a pre-defined pose in the space of all postures. It is traditionally tackled by learning a supervised model on detecting body parts [137, 161] and by adding extra information like the relative position between parts [132] in order to improve the accuracy and to learn similarities. Our approach does not require a model of the body or its parts, it rather extracts characteristic pose information directly from the image using the features learned by our approach.

We test our approach on two benchmark datasets for pose estimation: Olympic Sports [121] and Leeds Sports [80]. The *Olympic Sports dataset* comprises 16 different sport activities with a total of 525 clips and 113,516 frames. Since the athletes in Olympic Sports are not always in the center, we compute person bounding boxes for all videos using the approach of [52] due to its performance

in object and person detection. We firstly initialize the 5 convolutional layers and 2 FC layers using the filters of the powerful CliqueCNN model [11] which has shown competitive performances on pose analysis with unlabeled data. Then we run our self-supervised training procedure as explained in Section 3.3.1 to improve the representation. To be comparable, we use the same experimental setup as in [11]. In particular, [11] created a test set of around 1200 different postures and labeled 20 similar and 20 dissimilar samples for each of the 1200 query postures. Then, for evaluating pose similarity, we use the learned feature representation for sorting the 40 labeled samples based on their similarity to the query posture. The more of the 20 similar postures are among the first 20 samples after sorting, the better the representation of posture. We compare our approach with the baseline methods described in the following.

- **HOG-LDA [65].** For the first baseline method we follow the approach described in [65]: We first extract whitened HOG (histogram of oriented gradients) feature descriptors for all video frames and then train an LDA (linear discriminant analysis) model on clustered postures to learn visual similarities.

- **Exemplar-SVM [108].** The authors of [108] propose to divide a large machine learning problem into several easy-to-solve sub-problems by training one SVM classifier per positive sample (exemplar). Then, all classifiers together form a strong ensemble of Exemplar-SVM classifiers. We employ HOG features for training the classifiers and the negative samples originate from all categories except the one that the exemplar comes from. For evaluating the performance of Exemplar-SVM on posture estimation we use the combination of the trained classifier scores as similarity measure.

- **Exemplar-CNN [45].** Exemplar-CNN is an unsupervised approach that exploits data augmentations to create surrogate classes. A random set of data transformations is applied to each input sample and the CNN's task is to classify the applied augmentation classes. We follow [45] to train the network on the posture estimation datasets. Then, for evaluating its performance, we use the trained network representation for computing similarities among the test samples.

- **CliqueCNN** [11]. In CliqueCNN, Bautista *et al.* propose an unsupervised similarity learning approach that only relies on the highest/lowest similarities given a weak estimator of local distances. Very similar samples are grouped into compact cliques and the batches for training the CNN are constructed so that one batch only contains mutually distant cliques. The approach has shown state-of-the-art performances on human pose estimation at the time of publication.

- **ImageNet [90].** With the notation "ImageNet", we allude to the model trained supervised on the ImageNet classification task. Note, that in contrast to the unsupervised methods explained previously, this model has not been trained on the human pose estimation dataset itself.

Category	HOG-LDA [65]	Exemplar SVM [108]	Exemplar CNN [45]	ImageNet [90]	Clique CNN [11]	Ours
Basketball	0.51	0.63	0.58	0.55	0.70	**0.75**
Bowling	0.57	0.63	0.58	0.55	0.85	**0.87**
Clean&Jerk	0.61	0.71	0.58	0.62	0.81	**0.85**
Discus Throw	0.42	0.76	0.56	0.59	0.65	**0.68**
Diving 10m	0.42	0.54	0.51	0.57	0.70	**0.76**
Diving 3m	0.50	0.57	0.52	0.66	0.76	**0.84**
Hammer Throw	0.62	0.64	0.51	0.66	0.82	**0.88**
High Jump	0.64	0.76	0.59	0.62	0.82	**0.87**
Javelin Throw	0.71	0.72	0.57	0.74	**0.85**	**0.85**
Long Jump	0.60	0.69	0.57	0.71	0.78	**0.85**
Pole Vault	0.59	0.64	0.60	0.64	0.81	**0.83**
Shot Put	0.51	0.67	0.52	0.70	0.75	**0.76**
Snatch	0.64	0.76	0.59	0.67	0.84	**0.89**
Tennis Serve	0.70	0.75	0.64	0.71	0.84	**0.87**
Triple Jump	0.63	0.65	0.58	0.65	0.80	**0.83**
Vault	0.59	0.71	0.63	0.68	0.81	**0.86**
Mean	0.58	0.67	0.56	0.65	0.79	**0.83**

Table 3.1: Transferability of our temporal permutation approach to human pose estimation. We report the average area under the curve for all categories of the Olympic Sports dataset [121] using our approach and previous methods. Our model outperforms the previous state-of-the-art CliqueCNN [11] by a large margin.

In Table 3.1 we show the average area under the Curve (AUC) obtained with our model and the previously described baseline methods (the higher the better). In particular, we measure the AUC of a ROC (receiver operating characteristic) curve where the false positive rate is set in ratio with the true positive rate at different classification thresholds. Table 3.1 demonstrates that we consistently outperform all previous methods and achieve an average gain of 4% over the best performing previous method, CliqueCNN. Evidently, our self-supervised LSTM-based sequence ordering task captures more subtle structures, which becomes apparent when comparing the learned pose similarities of our approach with that of the state-of-the-art in Figure 3.14. The stripe pattern, which highlights the reoccurring postures of gait cycles in "Long Jump", is more clearly pronounced when using our representation in contrast to the features learned by CliqueCNN. Since our model has learned detailed similarities, it can find a large number of consistent nearest neighbors to a query frame. This ability is demonstrated in Figure 3.15 which shows the average over 100 nearest samples for two sports categories of Olympic Sports. It shows, that our approach is able to produce averages that still capture the essence of the pose.

The *Leeds Sports dataset* [80] contains 2000 pose annotated images. Since there are only static images but no videos as would be required for our method, we transfer the previously trained model from Olympic Sports and directly test on the Leeds Sports benchmark. For evaluation we follow the standard protocol

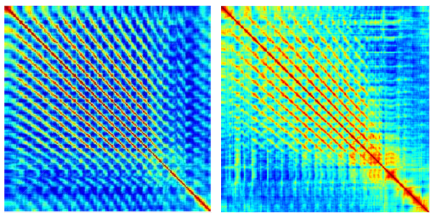


Figure 3.14: Similarity Matrices of the category *Long Jump* using our approach (left) and CliqueCNN [11] (right). Rows/columns are individual frames. *Red* signifies high similarity and *blue* indicates low similarity.

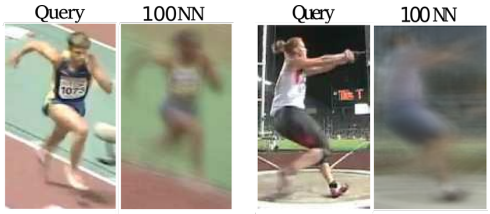


Figure 3.15: Applying the posture representation obtained by our self-supervised LSTM training to find similar samples: We show the average of the 100 nearest neighbors of a query frame from category *Long Jump* (left) and *Hammer Throw* (right).

Parts	HOG-LDA [65]	ImageNet [90]	Clique CNN [11]	Ours	Pose Mach. [137]	Deep Cut[132]	GT
Torso	73.7	76.9	80.1	**82.4**	88.1	96.0	93.7
Upper legs	41.8	47.8	50.1	**53.3**	79.0	91.0	78.8
Lower legs	39.2	41.8	45.7	**48.0**	73.6	83.5	74.9
Upper arms	23.2	26.7	27.2	**30.9**	62.8	82.8	58.7
Lower arms	10.3	11.2	12.6	**16.0**	39.5	71.8	36.4
Head	42.2	42.4	45.5	**48.9**	80.4	96.2	72.4
Mean	38.4	41.1	43.5	**46.6**	67.8	85.0	69.2

Table 3.2: Human pose estimation results on the Leeds Sports dataset [80] using our temporal permutation model trained on OlympicSports and without fine-tuning on Leeds Sports. We report the percentage of correct parts (observer-centric) for our approach, related works and GT similarities (GT). This experiment shows the generalization capabilities of our approach since it has not been trained on the dataset itself.

and measure the observer-centric Percentage of Correct Parts (PCP). A part/limb is considered as correctly detected if the distance between the true limb location and the predicted location is less than half of the limb length. In Table 3.2 we show the PCP acquired by HOG-LDA [65], ImageNet [90], CliqueCNN [11], our approach, two fully supervised method (Pose Machines [137] and DeepCut [132]) and ground-truth (GT) similarities. The GT indicates an upper bound on the performance we can achieve by an unsupervised approach that finds nearest training samples to query frames, but that is not trained on keypoints. Therefore, for each query we select the nearest neighbor using its keypoints and measure the PCP between the ground-truth keypoint annotation of the nearest neighbor and the query. Compared to the best previous unsupervised method (CliqueCNN) we improve by 3.1%. Achieving this gain without fine-tuning on the target dataset shows that our approach can nicely generalize.

Action Recognition. The goal of action recognition (also often called action

	Random	ImageNet	Shuffle&Learn [116]	VGAN [170]	Jigsaw* [122]	Ours
Acc (%)	47.8	67.7	50.2	52.1	51.5	**52.4**

Table 3.3: Evaluating the transferability of our temporal permutation model to action recognition using the human action dataset UCF-101 [150]. We initialize the network with the method shown in the top row until conv5 and fine-tune the weights on the action recognition task. Accuracies [%] are reported for our approach and previous methods. '*': The authors of Jigsaw [122] do not provide results for this task, we replicate their results using our implementation.

classification) is to recognize/classify a movement performed by a subject given some raw input data such as videos or accelerometers data. For evaluating our model on the action recognition task, we use the three splits of the human action datasets UCF-101 [150] with 101 different action classes and over 13k clips. We follow the common evaluation procedure of training the network supervised on the action recognition task (fine-tuning) after initializing with the weights pre-trained on the proposed unsupervised method. The fine-tuning is performed by using single frames as input and the network is trained and tested on every split separately. If not mentioned otherwise, all classification accuracies presented in this paragraph are computed by taking the mean over the three splits. For training and testing we utilize the PyTorch implementation[2] provided by Wang *et al.* [178] for augmenting the data and for the fine-tuning and evaluation step, but network architecture and hyperparameters are retained from our model. Table 3.3 shows the accuracy on UCF-101 after fine-tuning the representations obtained with the following methods: *(i)* random weights *(ii)* a supervised network pre-trained on the ImageNet [142] dataset *(iii)* 3 unsupervised previous methods and *(iv)* our self-supervised approach. The results show, that our features successfully outperform all previous works on action recognition and that our method further reduces the gap to supervised pre-training (*(ii)*).

3.4 Multi-Task Self-Supervision

Training a network on a single task enables us to accurately adjust hyper-parameters until we reach the highest performance on that problem. However, being so focused on one task let us ignore additional information that can be extracted from data and that might help to further improve the performance on our initial problem. Several works [47, 6, 115, 38, 180] have shown that sharing a representation among related tasks leads to a better generalization of the model. Also in SelfSL, previous works [38, 180] have shown that training a network on multiple surrogate tasks can improve the performance on the desired downstream task. However, these methods combine heterogeneous tasks that require an additional technique on top of the self-supervised training to exploit the full potential of their approach. We propose a model that combines two directly related tasks, which are complementary without the need of additional adjustment approaches.

[2] `https://github.com/yjxiong/temporal-segment-networks`

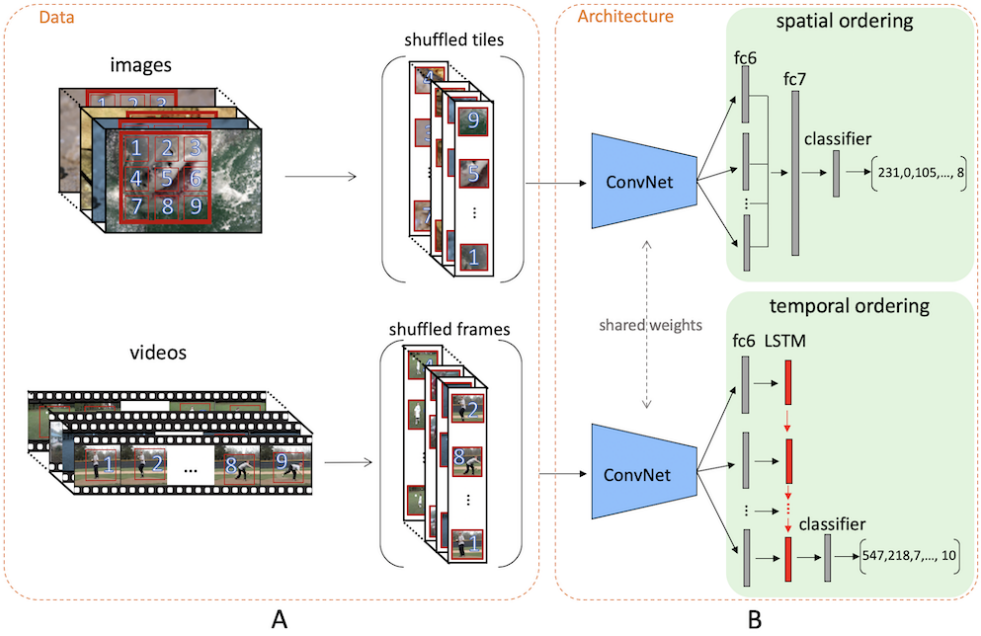


Figure 3.16: Overview of our spatiotemporal model. (A): We train our network on the spatial and temporal ordering task in parallel with two alternating batch types containing either shuffled tiles or shuffled frames. (B): Our network consists of a ConvNet and two classifier branches which aim to predict the correct index of the applied permutations. The training signal of both ordering tasks is used to update the ConvNet weights using SGD and the categorical cross entropy loss. The trained ConvNet weights are then used as initialization to address video and image understanding tasks.

3.4.1 Spatial and Temporal Permutation

Noroozi and Favaro [122] have shown that breaking the order of visual patterns on images and training a network to recover the structure provides a rich training signal. The same conclusion can be drawn from our previously described approach that breaks the temporal structure by shuffling frames. Both methods have proven through extensive evaluations that the learned representation carries good semantic and structural knowledge of objects or actions. Spatial and temporal shuffling are both ordering tasks which only differ in the ordering dimension. Therefore, we propose to address the two directly related ordering tasks jointly in order to extract information from images and videos simultaneously. This enables us to tackle a wide range of different visual understanding tasks from both domains.

We learn a CNN feature representation (AlexNet [90] architecture up to pool5) for images and individual frames of a video using spatiotemporal self-supervision. Training starts from scratch with a randomly initialized network. To obtain training samples for the spatial ordering task, we divide images into a $m \times m$ regular grid of tiles as suggested by [122] (see Figure 3.16 A top). The shuffling of the temporal ordering task if performed on frame level using l frames per video sequence (see Figure 3.16 A bottom). For the following part of this section, we are going to talk about a sample x in general, referring to a sequence of frames

(temporal task) or a partitioned image (spatial task). Similar to Section 3.3.1 we define $x = (x_1, x_2, \dots)$ as the sample that is to be shuffled by permuting its parts by some index permutation $\rho_i = (\rho_{i,1}, \rho_{i,2}, \cdots)$ with

$$\rho_i(x) := \left(x_{\rho_{i,1}}, x_{\rho_{i,2}}, \dots\right). \tag{3.2}$$

In Section 3.3.1 we have proposed to train the CNN on a binary classification problem (real vs shuffled). In contrast, Noroozi and Favaro propose in [122] to train the spatial shuffling with a multi-class classification loss, i.e. the network needs to predict the index i of the performed permutation ρ_i. Therefore, we have also experimented with this training procedure for the temporal permutation task and, in fact, have empirically found that using a multi-class classification loss leads to slightly better results. Moreover, using the same classifier type for both ordering tasks guarantees a more consistent and stable training of our multi-task network. For practical reasons, we perform a pre-processing step, as introduced in [122], that reduces the set of all possible permutations $\mathrm{P}^\star$ by sampling a set $\mathrm{P} \subset \mathrm{P}^\star$ of maximally diverse permutations $\rho_i \in \mathrm{P}$. This reduces the complexity of the classification task significantly since the set of all possible permutations $\mathrm{P}^\star$ contains $l!$ or $(m \cdot m)!$ elements. If, for example, $l = 8$ the total number of possible permutations (and therefore classes to predict) equals $8! = 40320$. The sampled set P is obtained by iteratively including the permutation with the maximum Hamming distance $d(\bullet, \bullet)$ to the already chosen ones until the desired amount of permutations is reached. Both self-supervised tasks have their own set of permutations.

To solve the ordering task of undoing the shuffling based on the features we want to learn, we need a classifier that can identify the permutation. The classifier architecture begins with an fc6 layer. For spatial ordering, the fc6 output of all tiles is stacked in an fc7 layer (see Figure 3.16B top); for temporal ordering the fc6 output of the frames is combined in a recurrent neural network implemented as LSTM [69] (as in Section 3.3.1 and see Figure 3.16B bottom). The outputs of the fc7 and LSTM layer are then processed by two final fc layers (one for each task) which estimate the index of the applied permutation ρ_i. The network is trained in parallel with two batches, one containing n images with spatially permuted tiles and one consisting of n sequences with temporally shuffled frames. Back-propagation provides two gradients, one from the spatial and one from the temporal task, which are passed backwards through the entire network down to conv1. Thus, both batches update the ConvNet weights which are evaluated in the following chapter.

3.4.2 Experiments

In this section we analyze the abilities of our model on image and video understanding tasks and compare with the performance of previous works. In particular, we perform unsupervised and supervised evaluations on image classification, image segmentation, image detection and action recognition.

Nearest Neighbor Search. We employ nearest neighbor (NN) search to perform an unsupervised evaluation of our model. For that matter, we use two different datasets: split1 of the human action dataset UCF-101 and the Pascal

Method	UCF101					Pascal				
	Top1	Top5	Top10	Top20	Top50	Top1	Top5	Top10	Top20	Top50
Random	18.8	25.7	30.0	35.0	43.3	17.6	61.6	75.5	85.5	94.2
Jigsaw[122]	19.7	28.5	33.5	40.0	49.4	39.2	71.6	**82.2**	**89.5**	96.0
OPN[98]	19.9	28.7	34.0	40.6	51.6	33.2	67.1	78.5	87.0	94.6
Ours	**23.4**	**34.6**	**41.0**	**47.3**	**57.1**	**39.4**	**71.7**	81.7	88.4	**96.1**

Table 3.4: Quantitative evaluation of our unsupervised feature representation using NN search on split1 of UCF-101 and Pascal VOC07. The nearest neighbors are computed using the cosine distance of the pool5 features. For UCF-101, 10 frames per video are extracted. Images of the test set are used as queries and the images of the training set as the retrieval targets. We report mean accuracies (%) over all chosen test frames. If the class of a test sample appears within the Top$k$ it is considered correctly predicted. We compare the results gained by *(i)* a random initialization, *(ii)* a spatial approach [122], *(iii)* a temporal method [98], and *(iv)* our model. For extracting the features based on the weights of *(ii)* and *(iii)* we use their published models.

Figure 3.17: Qualitative evaluation of our unsupervised feature representation using NN search on the Pascal VOC07 dataset. For every test sample we show the Top5 NN from the training set (Top1 to Top5 from left to right) using the cosine distance of the pool5 features. We compare the models of *(i)* a supervised training with the Imagenet classification task, *(ii)* our spatiotemporal approach, *(iii)* OPN as a temporal approach [98], *(iv)* Jigsaw [122] as a spatial method and *(v)* a random initialization.

VOC 2007 dataset. For computing the nearest neighbor on UCF-101, we extract 10 frames per video. The Pascal VOC 2007 dataset consists of $9,963$ images, containing $24,640$ annotated objects which are divided in 20 classes. Based on the default split, 50% of the images belong to the training/validation set and 50% to the testing set. We use the provided bounding boxes of the dataset to extract the individual objects and discard patches with less than $10k$ pixels. We use the model trained with our self-supervised approach to extract the pool5 features of the training and testing set. Then, for every test sample we compute the Top k nearest neighbors in the training set by using cosine distance. A test sample is considered as correctly predicted if its class can be found within the Top k nearest neighbors. The final accuracy is then determined by computing the mean over all testing samples. Table 3.4 shows the accuracy for $k = 1, 5, 10, 20, 50$ computed on UCF-101 and Pascal VOC 2007 using our approach, two previous works

Method	Non-Linear	Linear
Imagenet	59.7	50.5
Random	12.0	14.1
RotNet+[60]	43.8	36.5
Videos [179]	29.8	-
OPN* [98]	29.6	-
Context [36]	30.4	29.6
Colorization[196]	35.2	30.3
BiGan[41]	34.8	28.0
Split-Brain[197]	-	32.8
NAT[14]	36.0	-
Jigsaw[122]	34.6	27.1
Ours	**37.9**	**34.7**

Table 3.5: Test accuracy (%) of the Imagenet classification task. A linear [196] and non-linear [122] classifier are trained after the frozen features (pool5) of the methods shown in the left column. (*: indicates our implementation of the model, +: indicates bigger architecture)

[122, 98] and a random initialization. It can be seen, that our model achieves the highest accuracy for almost all k, meaning that our method produces more informative features for object/video classification. Especially the accuracies for video classification are significantly higher in comparison to the other approaches. We additionally evaluate our features qualitatively by depicting the Top5 nearest neighbors in the training set given a query image from the test set (see Figure 3.17). We compare our results with [122, 98], a random initialization, and a network trained on ImageNet (supervised).

In the next few paragraphs we evaluate how well our self-supervised model can transfer to image and video understanding tasks and to different datasets. For that, we initialize all networks with our trained model up to conv5 and fine-tune on the specific task using standard evaluation protocols.

ImageNet [142]. The ImageNet benchmark dataset can be used to test image classification abilities. It consists of ∼1.3M images divided into 1000 object categories. We evaluate our features by training a *(i)* linear [196] and *(ii)* non-linear classifier [122] (two-layer neural network) on top of the frozen conv layers. Table 3.5 shows that our features obtain the best results given all previous unsupervised methods for both the linear and non-linear classifier. The modified AlexNet introduced by [60] is not directly comparable to our model since it has 60% more parameters due to larger conv layers ("groups" parameter of the deep learning framework).

Pascal VOC. The dataset offers to evaluate the transferability of our features by fine-tuning on three different tasks: multi-class object classification and object detection on Pascal VOC 2007 [50], and object segmentation on Pascal VOC 2012 [51]. In order to be comparable to previous works, we fine-tuned the model without batch normalization, using the standard AlexNet with groups in conv2, conv4 and conv5. Previous methods using deeper networks, such as [180, 38], are

Method	Classification[50]	Detection[50]	Segmentation[51]
ImageNet	78.2	56.8	48.0
Random	53.3	43.4	19.8
RotNet[60]+	73.0	54.4	39.1
OPN[98]	63.8	46.9	-
Color17[94]	65.9	-	38.4
Counting[123]	67.7	51.4	36.6
PermNet[29]	69.4	49.5	37.9
Jigsaw[122]	67.6	**53.2**	37.6
Ours	**72.0**	52.4	**41.3**

Table 3.6: Evaluating the transferability of our features on three tasks on Pascal VOC. The network is initialized until conv5 with the method shown on the left column and fine-tuned for (i) multi-label image classification[88], (ii) object detection using Fast R-CNN [139] and (iii) image segmentation[101]. (i) and (ii) are evaluated on Pascal VOC'07, (iii) on Pascal VOC'12. For (i) and (ii) we show the mean average precision (mAP), for (iii) the mean intersection over union (mIoU). Fine-tuning has been performed using the standard AlexNet architecture, without batch normalization and groups 2 for conv[2,4,5]. ('+': significantly larger conv layers)

omitted from Table 3.6. For object classification we follow the evaluation protocol described in [88]. We do not require the pre-processing and initialization method described in [88] for any of the shown experiments. For object detection we train Fast-RCNN following the experimental protocol described in [139]. We use FCN [101] to fine-tune our features on the segmentation task. The results in Table 3.6 show that we significantly improve upon previous works. Our method outperforms even [60] in segmentation, which uses batch normalization also during fine-tuning and uses a larger network due to the group parameter in the conv layers.

Action Recognition. We employ the three splits of two different human action datasets to evaluate our unsupervised pre-trained network on the action recognition task: UCF-101 [150] with 101 different action classes and over 13k clips and HMDB-51 [91] with 51 classes and around 7k clips. As in Section 3.3.1, the supervised training is performed using single frames as input and the network is trained and tested on every split separately. All classification accuracies are computed by taking the mean over the three splits of the corresponding dataset. For training and testing we utilize the PyTorch implementation[3] provided by Wang et al. [178] for augmenting the data and for the fine-tuning and evaluation step, but network architecture and hyper-parameters are retained from our model. Table 3.7 shows that we outperform the previous works on UCF-101 and HMDB-51. During our self-supervised training our network has never seen videos from the HMDB-51 dataset, showing that our model can transfer nicely to another dataset.

[3] `https://github.com/yjxiong/temporal-segment-networks`

Method	UCF-101	HMDB-51
Random	47.8	16.3
ImageNet	67.7	28.0
Shuffle&Learn [116]	50.2	18.1
VGAN [170]	52.1	-
Luo et. al [103]	53.0	-
OPN [98]	56.3	22.1
Jigsaw* [122]	51.5	22.5
Ours	**57.3**	**23.2**

Table 3.7: Test accuracy (%) on the video understanding task action recognition. We initialized the network until conv5 with the approach shown in the left column and fine-tuned on UCF-101 and HMDB-51. '*': Jigsaw [122] do not provide results for this task, we replicate their results using our PyTorch implementation.

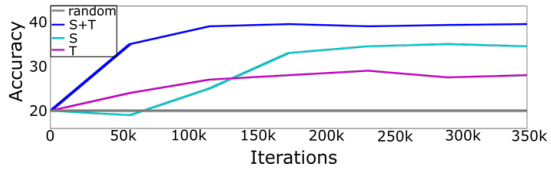


Figure 3.18: Unsupervised comparison of the different parts of our model on Pascal VOC'07 using nearest neighbor search. *(S)*: **S**patial task, *(T)*: **T**emporal task, *(S+T)*: **S**patial and **T**emporal task simultaneously.

Method	S	T	S&T	S+T
Pascal	67.6	64.1	69.8	**72.0**
UCF-101	51.5	52.8	54.2	**57.3**

Table 3.8: Supervised comparison of the different parts of our model on Pascal VOC'07 and UCF-101. (S): **S**patial task, (T): **T**emporal task, (S&T): first solely **S**patial task, followed by solely **T**emporal task, (S+T): **S**patial and **T**emporal task simultaneously.

3.4.3 Ablation Studies

Now we evaluate the spatial and temporal task separately to demonstrate that both components are essential to achieve the final performance.

Unsupervised Evaluation. We show in Figure 3.18 results on the Pascal VOC object classification task for every component of our model without any further fine-tuning. As in the previous section, we perform the unsupervised evaluation using nearest neighbor search by extracting pool5 features and computing cosine similarities. This shows how well the unsupervised features can generalize to a primary task, such as object classification. Figure 3.18 illustrates that the combined spatiotemporal model (S+T) clearly outperforms the networks trained on only one task (by 7% on the spatial and 14% on the temporal model). Furthermore, the combined network shows a faster improvement, which may be explained by the regularization effect that the temporal has on the spatial task and vice-versa.

Supervised Fine-Tuning. In Table 3.8 we present a supervised evaluation of all components. Each model is fine-tuned on the multi-class object classification task on Pascal VOC 2007 and on video classification using UCF-101. The

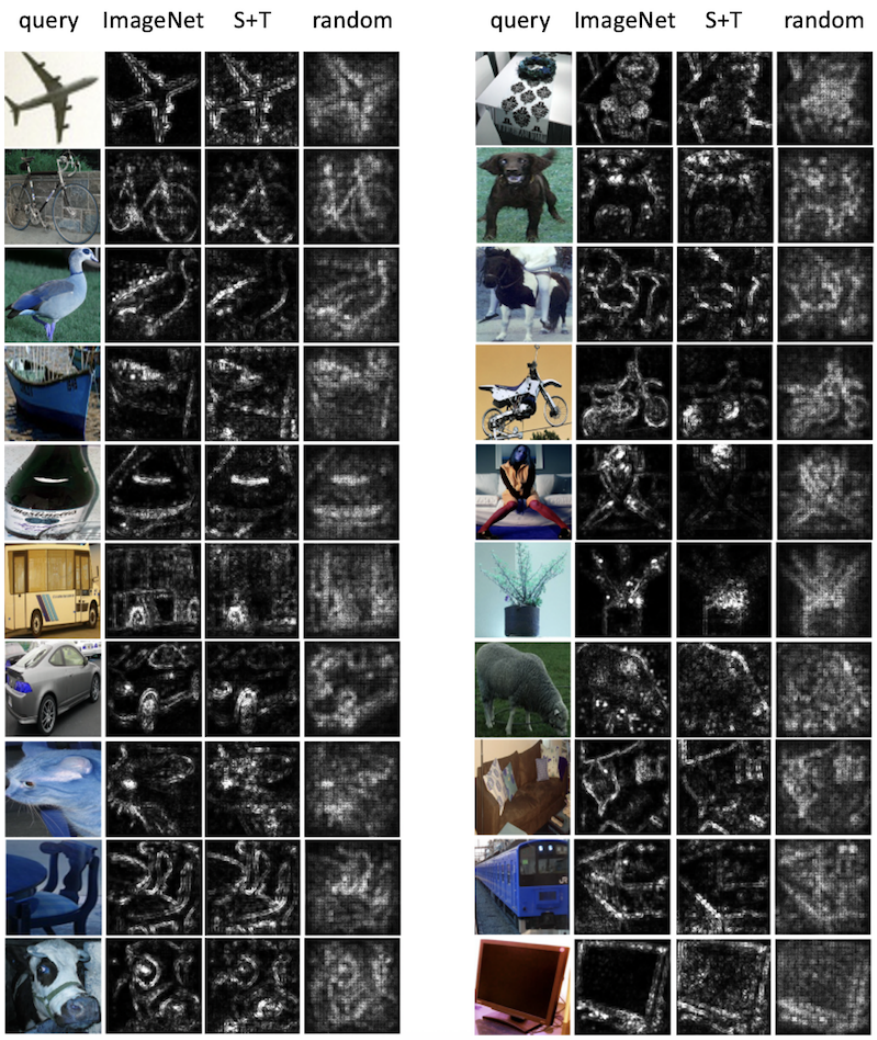


Figure 3.19: Class Saliency Maps of image classification models using networks trained with ImageNet, our spatiotemporal method and random weights as initialization.

results coincide with the unsupervised evaluation and show that the features of the spatiotemporal model (S+T) outperform both single-task models. The combination of the two tasks has been performed in parallel (S+T) and in a serial manner (S&T) by first training the spatial and then the temporal task on top of the spatial features. Training the permutation tasks in parallel provides a large gain over the serial version, showing that the two tasks benefit from each other and should be trained together.

3.4.4 Visualizations

For illustrating how well our representations have captured salient details of an object we apply the visualization procedure of [148]. As in Section 3.4.2, we evaluate our representation by transferring it to the task of image classification on Pascal VOC 2007. The network is initialized up to conv5 using *(i)* supervised training by means of the ImageNet classification task (upper bound), *(ii)* our spa-

tiotemporal self-supervision and *(iii)* no pre-training, i.e., random weights. Each of these initialized networks are then fine-tuned on Pascal VOC. Applying [148] yields a class-specific saliency map which indicates the relationship of individual pixels to the final classification. A good representation should capture essential aspects of the object. Fig. 3.19 shows that our self-supervised approach captures more of the characteristic structures than a randomly initialized network.

3.5 Technical Details

The following information describe the training process more detailed for both presented approaches (Section 3.3.1 and 3.3.1). If not stated otherwise, the details concerning the temporal part in Section 3.4.1 coincide with the specifications of the predecessor approach in Section 3.3.1.

All deep networks are implemented using the PyTorch[4] framework and the experiments were performed on a single Titan X GPU. The still images used for the spatial task are chosen from the training set of the Imagenet dataset [142]. For training our model with the temporal task, we employ the frames from split 1 of the human action dataset UCF-101 [150]. We randomly select 8 frames from each video and randomly crop a patch with the size of 224x224 per frame and resize to 75x75. For the spatial task we divide an input image into 9 non-overlapping parts and each tile has the size of 75x75. As augmentation, we randomly crop each tile/frame, apply a random color jittering to each of them and normalize the tiles/frames separately. We use 1000 permutations ($|\mathrm{P}| = 1000$) for both tasks in Section 3.4.1. For the training of the predecessor method with a binary loss (Section 3.3.1) we employ all possible permutations $\mathrm{P}^\star$ since the amount of permutations is not affecting the number of prediction classes. For updating the network, we use SGD with a starting learning rate of 0.001 which we reduce after 200k iterations by a factor of 10. Our network runs in total for 350k iterations. We use a batchsize of 128 for both spatial and temporal tasks. A batch in Section 3.3.1 contains $n = 128$ different sequences and 256 samples in total since every sequence is represented by its real and randomly shuffled version (as expressed in Equation 3.1). For the spatial classification branch, the fc6-layer has a size of 1024, fc7 has 4096 dimensions. For the temporal task we use an fc6-layer with 512 neurons and one LSTM layer with the hidden dimension of 256.

3.6 Discussion

In this chapter, we have introduced self-supervised representation learning and presented two novel SelfSL methods that efficiently employ visual data without requiring any human annotations. We break the order of visual patterns and ask a neural network to recover the structure. The temporal permutation of frames leads to a detailed posture and sequence representation that is evaluated on human pose estimation and action recognition. To exploit the full potential of both images and videos we have proposed a multi-task representation learning approach that combines two directly related tasks. A neural network is trained

[4] `http://pytorch.org/`

simultaneously on permuting image tiles and shuffling video frames. This enables us to learn a broadly applicable feature representation for addressing various video and image understanding tasks. The learned representation outperforms previous works on image classification, multi-class object classification, action recognition, object detection and object segmentation using four different benchmark datasets. Nearest neighbor search demonstrates that our model is able to comprehend similarities or dissimilarities between related or unrelated events, respectively. This property is further analyzed in the next chapter where we use SelfSL to compare motor behavior videos among subjects that are participating in biomedical studies.

Chapter 4

Unsupervised Motor Behavior Analysis

Motor behavior analysis seeks to understand the voluntary dynamic change of posture of individuals and how it transforms due to disease developments or external influences such as medication. A detailed analysis of posture changes during skilled motor tasks such as walking or grasping can reveal distinct functional deficits and their restoration during recovery. Previous works on behavior analysis require placing physical or virtual markers on the individual to capture the movements. This procedure is very cost- and time-intensive considering the annotation effort for labeling characteristic locations. We propose to use video-based self-supervised learning to automatically learn accurate posture and behavior representations. Our approach does not require any physical or virtual markers and is able to automatically analyze behavior changes without any user interaction or prior expert knowledge. An essential goal of motor behavior analysis is to develop an effective treatment for restoring impaired motor skills. Thus, besides evaluating the overall behavior, it is crucial to discover small differences in motor function between the impaired individual and a reference movement to allow a direct adjustment of treatment. For that matter, we propose a novel deep generative model that is able to detect and magnify subtle posture deviations across individuals.

4.1 Introduction

Movements of individuals are the visible result of several intricate internal processes. Therefore, the research of motor behavior [13, 4, 55, 175, 81, 112, 5, 167, 143] concerns a wide variety of disciplinary perspectives. For a basic understanding of motor function several factors need to be taken into account such as the neurophysiological or psychological state of the individual. Thus, analyzing the visible execution of movements can reveal crucial information about the subject's internal condition without requiring invasive procedures such as brain surgery. Figure 4.1 illustrates this concept.

Especially in biomedical research, motor behavior analysis is an important component for evaluating different treatment options for diseases that affect the motor function such as a stroke or multiple sclerosis (MS). Even though machine

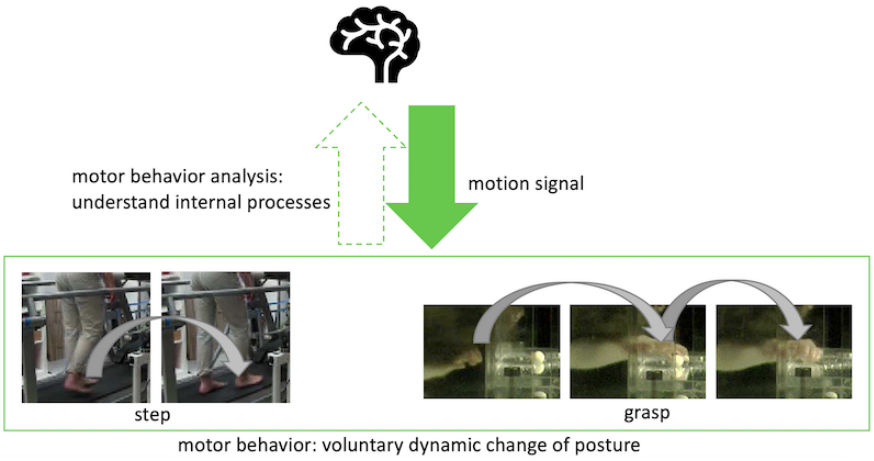


Figure 4.1: Motion signals in the brain trigger visible posture changes. Analyzing the behavior reveals information about the state of the internal processes, e.g. the brain function.

learning has lately shown tremendous successes in automatizing several medical studies [133, 22, 198, 147, 153, 140, 168], a large amount of neuroscience researchers still perform behavior analyzes manually by closely observing and annotating hours of recordings.

There exist two popular approaches for behavior analysis [112, 81] that employ machine learning techniques to provide semi-automated behavior evaluations. However, both rely on annotated keypoints and strong user interactions. In this chapter, we propose a *diagnostic support system* that does not require any manual annotations. In particular, we demonstrate how unsupervised computer vision can be used to automatize motor behavior analysis. We apply our SelfSL approach introduced in Section 3.3.1 to two biomedical studies of humans and rodents that suffer from a neurological disease. Given recordings of several individuals that perform a pre-defined movement such as walking, our SelfSL approach is able to encode the fine-grained details of the shown behavior. The learned posture and sequence representation can be used to automatically observe and quantify the changes in behavior triggered by medication or disease aggravation.

Besides measuring the impairment of patients, a detailed diagnosis also requires understanding how and when the behavior changes in comparison to a reference (e.g. healthy) movement. However, human senses are not construed to detect small variations in behavior, especially when comparing two different subjects. In other fields, scientists have been creating tools to magnify their senses in order to recognize phenomena that are not directly observable, e.g. microscopes or telescopes for increasing the visual senses. We belief that such a tool is also required to improve the perception of behavioral changes. Therefore, we additionally propose a generative model-based approach that amplifies subtle posture differences between an impaired individual and a reference subject. Build upon the fine-grained representation learned in our SelfSL method, our generative model generates new images that exaggerate impaired postures. If, for instance, a patient is not able to fully bend their knee while walking, our magnification

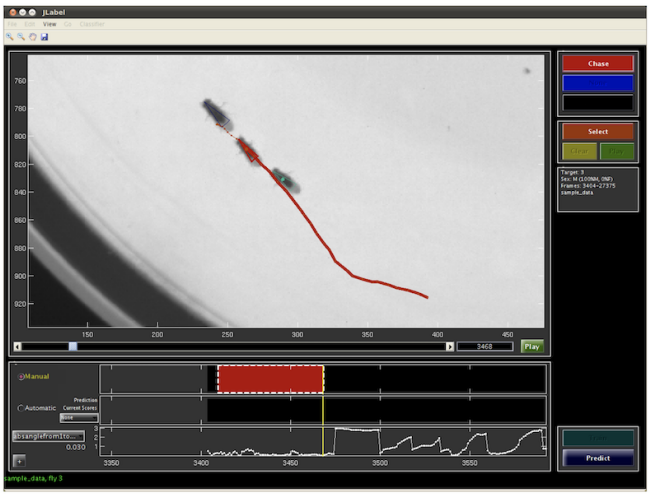


Figure 4.2: **Jaaba** [81] interface screenshot while the user labels the behavior 'chase' given a video of flies. The dashed white box in the bottom of the figure signals the frames that are being labeled. After annotating frames with and without the desired behavior, Jaaba trains a classifier for detecting this behavior. Then, the user can choose the frames that should be analyzed automatically and Jaaba provides a behavior prediction with a confident score per selected frame. The figure is taken from `http://jaaba.sourceforge.net/Training.html`.

approach magnifies this behavior (since it is different from healthy gaits) by producing images that only show an outstretched leg throughout the gait. In this way, subtle impairment which could be easily overlooked is more visible to the user (e.g. a doctor). Our generated images facilitate the user's interpretation of the symptoms, help to actually understand the disorder and ultimately support researchers in developing or adjusting treatments.

After describing previous works in the next section, we introduce in Section 4.3 the experimental setup of two medical scenarios that benefit from such a diagnostic support system. Then, we demonstrate in Section 4.4 how our SelfSL method can be applied to the given scenarios. Our novel generative model for amplifying subtle posture differences is introduced in Section 4.5. We demonstrate the wide applicability of our learned features and generated images through extensive experiments and a comparison with previous works in Section 4.6.

4.2 Previous Work

Placing physical markers on an individual to capture the trajectory of a movement is very tedious and especially problematic when working with small and hairy animals. Therefore, many researches have moved from physical to virtual markers [141, 131, 130] by virtually annotating the interesting regions after recording the movements. In recent works, machine learning is employed to facilitate the evaluation of motor behavior by e.g. training a supervised model that can detect keypoints or estimate postures. The two most popular machine learning based approaches are currently Jaaba [81] and DeepLabCut [112] which are described subsequently.

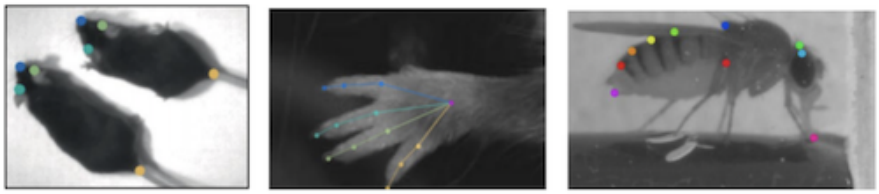


Figure 4.3: Exemplary depiction of keypoint annotations of rodents and flies for training **DeepLabCut** [112]. The examples are taken from [112].

Jaaba. Janelia Automatic Animal Behavior Annotator (Jaaba) is an open-source program for detecting annotated behavior in animals. It collects quantitative statistics that describe the behavior shown in an input video. Given the graphical user interface (GUI) provided by Jaaba, a user first needs to define the behavior they want to detect on some of the input frames and Jaaba attempts to detect the annotated behavior on the remaining frames after training a classifier. An example of the user interface for the task of detecting the behavior 'chase' of flies can be found in Figure 4.2. For training the classifier, Jaaba computes per-frame features that describe the state of the animal. The proposed features can be grouped into locomotion, landmark-based, appearance-based and social categories. The user can select the most meaningful features or develop new per-frame features. The classifier is then trained using the pre-defined features and the GentleBoost learning algorithm [56]. If necessary, the user can request to re-train the classifier after labeling more frames or correcting wrong predictions.

DeepLabCut. DeepLabCut [112] is an open-source software package for quantifying behavior with 3D pose estimation. It is based on the deep learning feature detector architecture of "DeeperCut" [72], an approach for multi-human pose estimation. The network of DeepLabCut consists of a variant of ResNets [67] that are pretrained on ImageNet [142]. Deconvolutional layers are used to up-sample the ResNet features to produce prediction masks for every pre-defined keypoint. For training the model, the user needs to label a few hundred frames with the desired keypoints. Figure 4.3 depicts examples of keypoint annotations for rodents and flies. The number of annotated frames required for achieving satisfactory results is dependent on the quality of videos and the difficulty of the task. The developer of DeepLabCut provide weights of the network trained on $\sim 25,000$ labeled images for human pose estimation. Users can use these weights as initialization before fine-tuning on their desired dataset to reduce the training effort.

The main difference between our approach proposed in the following sections and the previously described methods is the amount of required user interaction. Our approach is completely unsupervised and does not require any annotations. Moreover, our model learns a feature representation of the entire object-of-interest and does not define behavior based on only a few keypoints. We show the benefits of our method in the experimental section where we also compare with the previously described methods.

As a plus, we also propose a generative model that generates new images for facilitating the perception of small differences between a query and reference subject. We are not aware of any previous behavior analysis approaches that

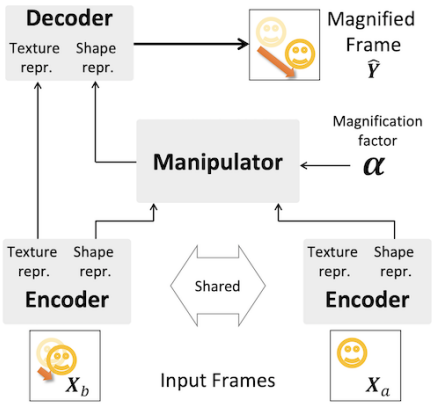


Figure 4.4: Overview of the latest video motion magnification method based on an encoder-decoder architecture. The figure is taken from [125].

provide such a magnification tool. However, there exist previous works that address video magnification in a different context which we will discuss in the following paragraph.

Magnification. Magnification is a valuable tool to enhance differences on images or a set of images, in order to automatically detect and visualize small deformations. Tali *et al.* [33] and Tlusty *et al.* [160], for instance, visualize non-local variations between repeating structures in a single image or for multiple views (e.g. for visualizing irregularities in a stone wall). Previous works on video motion magnification have primarily addressed the amplification of small motions [125, 100, 183, 48, 173, 174, 199, 166] or the deviation from a predefined reference shape [172], but only within the same video [100, 125]. The first attempt of motion magnification [100] computes optical flow between video frames and then amplifies every pixel separately given the optical flow information. Following works [183, 173, 48, 174, 199, 125] do not alter pixels directly, but they decompose the video into an alternative representation, e.g., by using the frequency domain. The desired motion is then selected and used to generate the image. Oh *et al.* [125] proposed the first deep learning based approach to video motion magnification using an encoder-decoder architecture. Figure 4.4 provides an overview of the method. The network is trained with a regularization loss that enforces the shape representation of a color perturbed frame to be the same as the original frame in order to induce a separation of texture and shape. The loss ensures that the magnification only affects shape and not intensity changes. For performing the magnification, Oh *et al.* [125] introduce a specialized non-linear magnification module that is trained using a synthetic dataset. We also amplify differences using video frames as input, however, we amplify the deviation in posture across individuals and videos. Moreover, our approach does not require an additional non-linear module for producing meaningful magnifications. Since this approach is the most recent and most successful publication in video magnification, we provide a comparison of our magnification results with the amplifications generated by Oh *et al.* [125] in the experimental section.

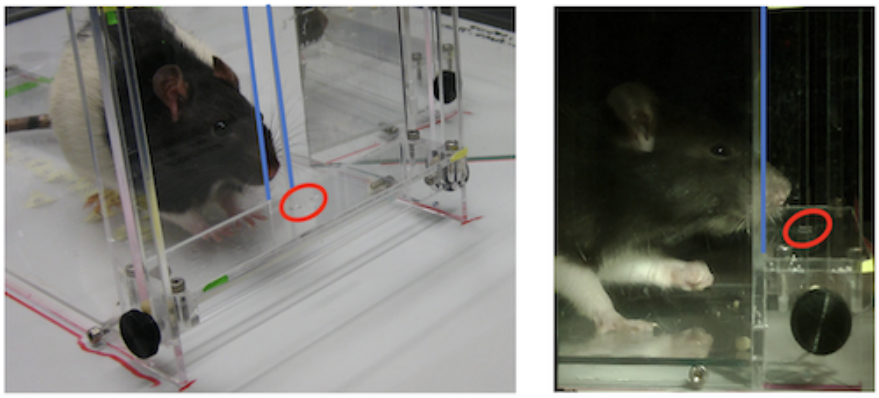


Figure 4.5: Experimental Setup of the Rat Stroke Model. Left: The animal is placed in a Plexiglas box and reaches through a small aperture (blue lines) to grasp for a sugar pellet that is placed manually within the area marked with a red oval. Right: Exemplary frame of a video from a grasping session that is used to analyze the behavior.

4.3 Experimental Setup

We evaluate our diagnostic support system on two species suffering from neurological diseases. We closely collaborated with neuroscientists who executed the medical studies and recorded the subjects with a single camcorder.

4.3.1 Rat Stroke Model

In the first study, we analyzed the recovery of impaired forelimb function in rats that suffered from a photothrombotic stroke in the sensorimotor cortex. To evaluate the impact of the neurological disease and subsequent treatments, the animals performed a skilled forelimb action (single pellet grasping) at several stages during the experiment. They were placed in a Plexiglas box and recorded while grasping a sugar pellet that was manually positioned on a shelf (see red oval on Figure 4.5). The study was performed with in total 36 adult female Long-Evans rats. In the first stage, the animals were trained for up to 5 weeks in the complex single pellet grasping task until at least 60% of the grasps were successful. Then, all rats received a photothrombotic stroke targeting the sensorimotor cortex that corresponds to the preferred paw for grasping. The animals were divided into four treatment groups and the ability to grasp was again examined for every animal 2 days, 7 days, 14 days, 21 days, and 35 days after the stroke. The treatment groups are the following:

1. *Stimulation*: neuronal stimulation of the intact corticospinal tract with blue light from day 3 until day 14 after the stroke (3× a day)

2. *Stimulation & Training*: stimulation as above plus intensive grasping training of the impaired paw during the 3rd and 4th week after stroke (for at least 100 sugar pellets per training session)

3. *Delayed Training*: intensive grasping training of the impaired paw during the 3rd and 4th week after stroke

Figure 4.6: Exemplary depiction of several subjects in HG2DB. The different subjects wear different types of trousers in different colors.

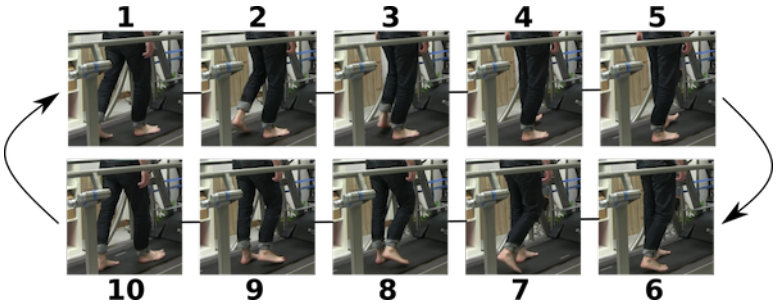


Figure 4.7: One Walking cycle of a subject from HG2DB represented by 10 linearly spaced frames.

4. *No Treatment*

For a more detailed description of the experimental setup and treatment groups please see [175]. To evaluate the success of the performed medications, we analyze the rehabilitation behavior of the different treatment groups using the recordings from all stages. We verify the results achieved with our SelfSL algorithm using manual scores and compare with previous works in Section 4.6.

4.3.2 Human Gait Dataset (HG2DB)

For our second scenario, human subjects with different neurological disorders were recorded while walking on a treadmill. The Human Gait Dataset to Study Dysfunctional Behavior (HG2DB) contains 14 patients diagnosed with multiples sclerosis (MS), 18 patients with hydrocephalus and 10 healthy subjects that are used as reference. The videos were recorded at University Hospital Zurich between 2017 and 2018. All patients showed walking impairment where some were more prominent than others. 24 hours after receiving appropriate treatments, all impaired patients were recorded again to evaluate the change in behavior due to the medication. For further information about the applied treatment of the MS patients please see [55]. The movements were documented with a standard consumer video-camera and the position of the tripod was kept constant for all recordings. Figure 4.6 provides exemplary frames of several subjects of HG2DB and Figure 4.7 shows a full walking cycle divided in 10 linearly spaced frames. For anonymity purposes, we only use the lower part of the body since this is also the area that needs to be analyzed for assessing the walking abilities. As for the rat stroke model, described above, we employ our methods to analyze the behav-

ior of the patients before and after treatment and compare them with healthy individuals.

The videos have been recorded by Linard Filli with whom we also stayed in close contact while performing the behavior analysis experiments on this dataset.

4.4 Self-Supervised Learning for Behavior Analysis

A detailed motor behavior analysis goes far beyond a trajectory analysis [81]. It requires an accurate representation of the entire object-of-interest to capture even small distinctions between various stages of rehabilitation or between a query and reference movement. Focusing on only a few pre-defined keypoints easily leads to forfeiting important elements essential for a fine-grained study of motor function. Moreover, in order to expedite and facilitate the evaluation of behavior for medical experts, the algorithm should circumvent tedious manual annotations and an annotator bias. For that reason, we propose to employ SelfSL to learn a detailed representation of posture and behavior without requiring any labels.

4.4.1 Learning a Fine-Grained Representation

The challenge is to learn an encoding that can compare similar behavior across individuals despite their differences in appearance. Fortunately, our approach described in Section 3.3.1 has shown excellent performances on datasets with diverse subjects and tasks related to motor behavior analysis such as posture estimation or action recognition. Therefore, we train per medical study a deep neural network with the temporal permutation task introduced in Section 3.3.1 and Figure 3.13 using the corresponding recordings. The sequences for training the network are at first randomly extracted from the given videos. Then, we randomly shuffle the sequences, input the real and permuted samples into the CNN and train the network to distinguish between the two sequence types. Since the training data contains sequences from different subjects, the network learns to be invariant to appearance characteristics and only focuses on the execution of the shown movements. After training, we can use simple classifiers on top of the fine-grained representation in order to measure the recovery during rehabilitation or to distinguish between different diseases.

During the grasping process of the rat stroke experiment, the camcorder records half of the animal's body that performs the action (as can be seen in Figure 4.5 right). However, only the paw is essential for assessing motor function since the stroke solely affected the forelimb. Therefore, before training the network with the temporal permutation task, we have implemented an approach that separates the paw from background clutter without requiring any annotations. This procedure is described in the next paragraph. Note, that we do not require any detection mechanism for the human gait dataset since a fixed bounding box per video produces satisfactory extractions of the region-of-interest (lower body).

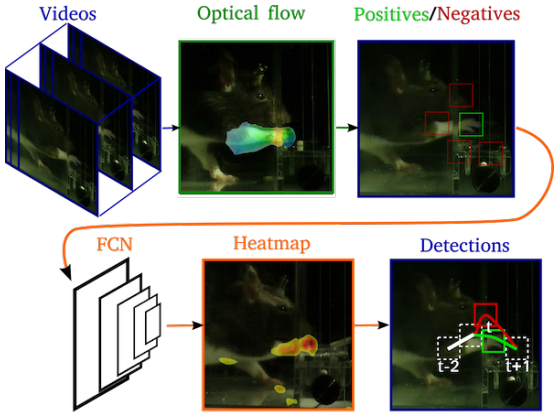


Figure 4.8: Visualization of our detection system, which uses optical flow for initial positive samples and random negatives to train an FCN for extracting candidate regions before applying temporal smoothing (bottom right).

4.4.2 Detection

Finding and tracking a rat's paw during grasping is challenging for a number of reasons. There is no initialization for tracking provided and due to frequent occlusions by other body parts (other paw, arm, nose, etc.) detections are frequently lost. Moreover, paws are small, furry, fast-moving (implying large displacement between successive frames and motion blur due to limited illumination of nocturnal rodents that are distracted by intense light), and appearance varies significantly between subjects as does shape between different hand postures. Learning a representation and detector for paws with these large variations in shape and appearance is therefore demanding—especially since we do not require laborious manual annotations of paws. Therefore, we follow a sequential bootstrapping procedure to train a CNN-based hand detector in an iterative manner, initializing it with motion information to start with the easy to extract paws first and then consecutively learning more complex ones. We initially extract a set of candidate paw regions by computing optical flow [99] and decomposing frames with [182] into a low-rank background model and a sparse set of connected foreground pixels, thus finding strongly moving paws. In addition to these positive samples we add hard negatives randomly sampled from locations around the positives to then train a CNN (AlexNet [90], trained with stochastic gradient descend with cross-entropy loss) to separate both classes. The resulting network which we will call E_π^0 is then turned into a fully convolutional network (FCN^0) by reshaping the last 3 fc-layers. A deconvolutional layer is not necessary since we do not need pixel-accurate segmentation. Paws are then extracted by taking the 5 strongest candidate detections per frame from the FCN^0 scoremap and then selecting the best one by temporal smoothing, i.e., fitting a polynomial to ten consecutive frames and choosing the smoothest trajectory. Figure 4.8 summarizes this procedure. We show in the experimental section that the new detector has better performance than the initial motion detections.

Given the new and improved detections of all sequences we can now train the model (*ConvNet+LSTM*) with the temporal permutation task to learn a behav-

ior and posture representation. Since E_π^0 has implicitly learned a representation of paws we employ the weights of E_π^0 as initialization for the ConvNet. The LSTM and classifier weights are initialized randomly. Training with the temporal permutation task yields not only a behavior representation but also a refined posture network E_π^1 that presumably understands posture better than the paw detector FCN^0. Therefore, we implement a bootstrap retraining to further improve the forelimb detections and consequently also the posture representation. In particular, we use E_π^1 to obtain a new paw detector FCN^1 which in turn yields better detections to retrain the *ConvNet+LSTM* model on the temporal permutation task. The bootstrap retraining of the *ConvNet+LSTM* network and the *FCN* paw detector finally converges after two iterations which is demonstrated quantitatively in the experimental section.

For the human gait dataset, the *ConvNet+LSTM* network is initialized randomly and only trained once with the surrogate task since an adequate region-of-interest can be extracted from the beginning for all videos.

4.5 Magnification of Impaired Behavior

The SelfSL approach enables us to learn a representation for quantifying visible impairment. However, a complete and detailed analysis of behavior also includes to spot even small indications of impairment to better understand the possible degeneration of motor function. For that matter, we require a comparison of the performed action with some reference (e.g. healthy) movement. However, this task cannot always be solved easily, since the comparison of different behaviors is performed across diverse subjects. Appearance differences make it difficult to solely focus on postures and to spot their subtle deviations. Hence, we extend our behavior analysis tool with an additional model that is able to magnify subtle posture differences across subjects to facilitate the perception of deviations for humans.

Model. We require an approach which provides *(1)* a detailed representation of posture that allows to magnify subtle deviations and *(2)* magnifications expressed in the form of images so that humans are able to understand and interpret the output. In fact, the model needs to first explicitly learn an encoding space conditioned on the input image (in which we can manipulate the input posture) and second decode the encoding back to the image space. Hence, an autoencoder (AE) is the architecture of choice. An AE consists of an encoder E and a decoder D. E extracts a lower dimensional representation of the input image x, and D translates the representation back to the input space by generating the reconstructed image $\hat{x}$. We additionally require a model that explicitly separates posture from the remaining image components since we only want to magnify the differences in posture. Hence, we use an autoencoder containing two encoders, E_π for posture and E_α for appearance. Figure 4.9A displays the structure of the network. In detail, E_π and E_α encode an input image x into the low dimensional latent space z_π and z_α, respectively. The decoder D is used to reconstruct x given z_π and z_α as input and outputs the reconstruction

$$\hat{x} = D\big(z_\pi^x, z_\alpha^x\big) \tag{4.1}$$

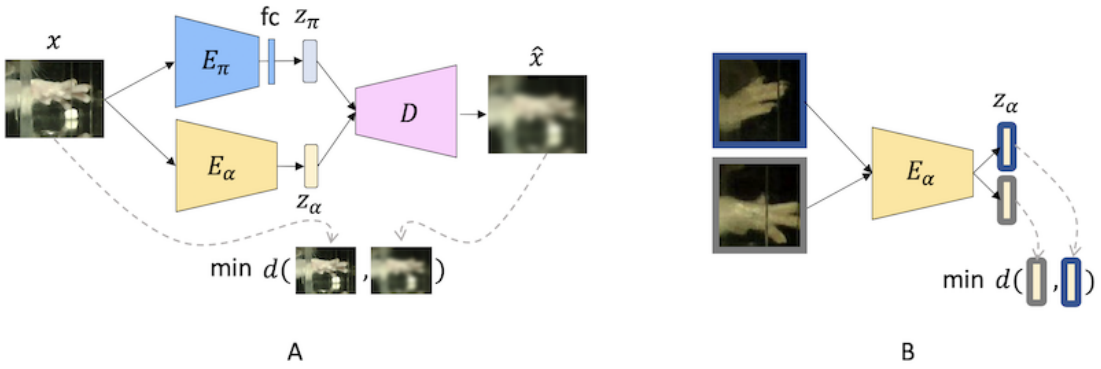


Figure 4.9: Training of our generative model which is employed during inference for magnifying subtle posture differences. *A*: To produce real-looking images, the generative model with two encoders E_π (posture) and E_α (appearance) and one decoder D is trained by minimizing the distance between the input image x and the generation $\hat{x}$. *B*: We additionally introduce an appearance loss that minimizes the distance between the latent representations of images that originate from the same video, i.e. images which have the same appearance.

with z_π^x and z_α^x the latent posture and appearance representation of x, respectively.

Training. In order to generate satisfying magnifications, the network needs to *(1)* produce real-looking images and *(2)* separate posture from appearance. To satisfy condition *(1)*, the generative model is trained by minimizing the reconstruction loss, i.e. distance between the original input image x and the generated image $\hat{x}$

$$\mathcal{L}_{rec} = d(x, \hat{x}) \tag{4.2}$$

with $d(\bullet, \bullet)$ the perceptual distance [78]. The reconstruction loss is visually illustrated in Figure 4.9A.

To ensure that we can manipulate posture without changing the appearance as required for magnification, we introduce two additional training principles that define the role of E_π and E_α. Fortunately, we have already learned a fine-grained posture representation for our medical scenarios in the previous section using the temporal permutation task. For that reason, we use the ConvNet network from Section 4.4 as our posture encoder E_π and thus circumvent the training from scratch for the posture encoder. In fact, we do not update E_π since it is already fully trained on extracting only posture information from the input image. We solely train one fully connected layer placed on top of E_π (as shown in Figure 4.9A). To define the functionality of E_α we introduce an appearance loss

$$\mathcal{L}_{app} = |E_\alpha(x) - E_\alpha(x')|^2 \tag{4.3}$$

which minimizes the distance between the latent representations of two images from the same video. This ensures that input images with the same appearance are mapped to the same point in the latent space z_α. This procedure can be viewed in Figure 4.9B.

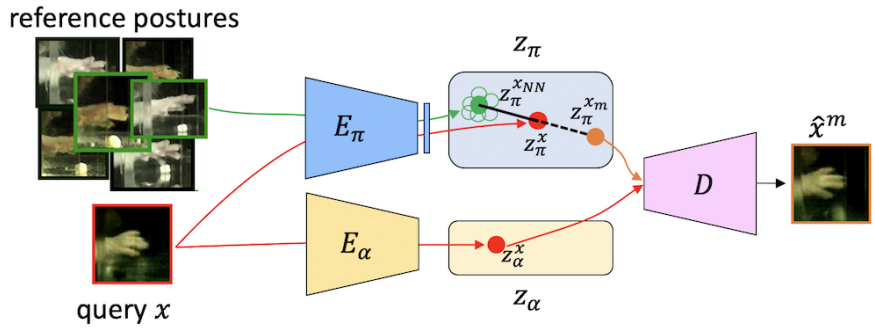


Figure 4.10: Illustration of our magnification strategy. We linearly extrapolate in the latent posture space z_π between a reference posture (filled green dot) and the query posture (red filled dot) to obtain the magnified posture (filled orange dot). The magnified image is then generated by feeding the magnified posture and the latent appearance representation of the query frame (filled red dot in z_α) into the decoder D after concatenating them. The reference postures are in comparison to the query posture more horizontal and the output image displays the exaggeration of this behavior by further rotating the paw.

To summarize, our generative model is updated with the following training objective:

$$\mathcal{L} = \mathcal{L}_{rec} + \gamma \mathcal{L}_{app} \tag{4.4}$$

where γ is a free parameter and we set $\gamma = 10^{-3}$ in our experiments. More technical details about the network structure and hyper-parameters can be found at the end of this chapter (Section 4.7).

Magnification. After the training of the generative model is converged, we employ the network for performing magnifications in the posture space without altering the appearance. Figure 4.10 illustrates our strategy for magnifying the posture differences between a query frame and reference posture. First, we collect several reference postures by determining the nearest neighbors of the query frame x from the set of all healthy postures. We compute the nearest neighbors by using the posture representation of our behavior network. Using the nearest neighbors ensures that all reference frames originate from the same type of posture, e.g. all arise from a right step. Then, we insert the selected frames into the posture encoder and perform magnification between the representation $z_\pi^x = E_\pi(x)$ of the query frame x and the encoding $z_\pi^{x_{NN}}$ of the reference postures using linear extrapolation. In this way, we obtain the magnified representation $z_\pi^{x_m}$,

$$z_\pi^{x_m} = z_\pi^{x_{NN}} + \lambda \cdot (z_\pi^x - z_\pi^{x_{NN}}), \tag{4.5}$$

with $\lambda > 1$ a pre-defined parameter and $z_\pi^{x_{NN}} = \frac{1}{K}\sum_{j=1}^{K} E_\pi(x_j)$ the average posture representation of the K chosen reference frames (nearest neighbors). The linear extrapolation is illustrated in Figure 4.10 via the black line in the posture space z_π that connects the different posture representations. In order to generate the magnified frame $\hat{x}_m$, we extract the appearance encoding z_α^x of the query frame x, concatenate the two encodings (z_α^x and $z_\pi^{x_m}$) and insert the result into the decoder D.

Models	Accuracy(%)
OpticalFlow [99]	40.2
FCN^0	58.0
FCN^1	81.4
FCN^2	**82.1**

Table 4.1: Accuracy of the detections obtained by using optical flow, after one round of training (FCN^0), and after two (FCN^1)/three rounds of retraining (FCN^2).

4.6 Experiments

Now we present several experiments which evaluate the trained representations and analyze the behavior of the two medical scenarios given our models. We further show that our diagnostic support system outperforms previous works.

4.6.1 Paw Detection

For the rat stroke model we first require an approach that detects the object-of-interest, i.e. the grasping forelimb, before learning a posture and behavior representation. We now evaluate the accuracy of the obtained regions and provide more details about the training process of the paw detector.

As described in Section 4.4.2 we extract the first initial candidate paw regions ($\sim 15,000$) by using optical flow [99] and robust PCA [182]. We train a network (E^0_π) with these initial detections and transform it to an FCN (FCN^0) in order to obtain candidate regions for the full dataset ($\sim$ 40,000 densely extracted sequences). Then, we perform the bootstrap retraining of the *ConvNet+LSTM* network and the paw detector until convergence (after 2 iterations). We evaluate the detection performance for every phase on a small test set of manually labeled paw locations from different videos. Table 4.1 shows the detection accuracy of successive rounds of the bootstrap retraining (detections are counted as correct if their intersection over union with the ground-truth is $\geq 50\%$).

4.6.2 Evaluation of the Learned Representation

Next, we qualitatively and quantitatively evaluate the posture and behavior representation learned by our SelfSL method. We use the fc6 features of our ConvNet network (displayed in Figure 3.13) as posture representation and the LSTM features as behavior encoding. If not stated otherwise, we employ the final model of the bootstrap retraining procedure (after 2 iterations) for the rat stroke scenario. For the human gait dataset, we train the *ConvNet+LSTM* network only once since no detection algorithm of the region-of-interest is required for the given videos.

Qualitative Results. The primary goal of our learning procedure is to train a model that understands postures and behavior which includes identifying similar or dissimilar samples. Therefore, we assess the quality of the learned representation by computing the cosine similarities in the representation space

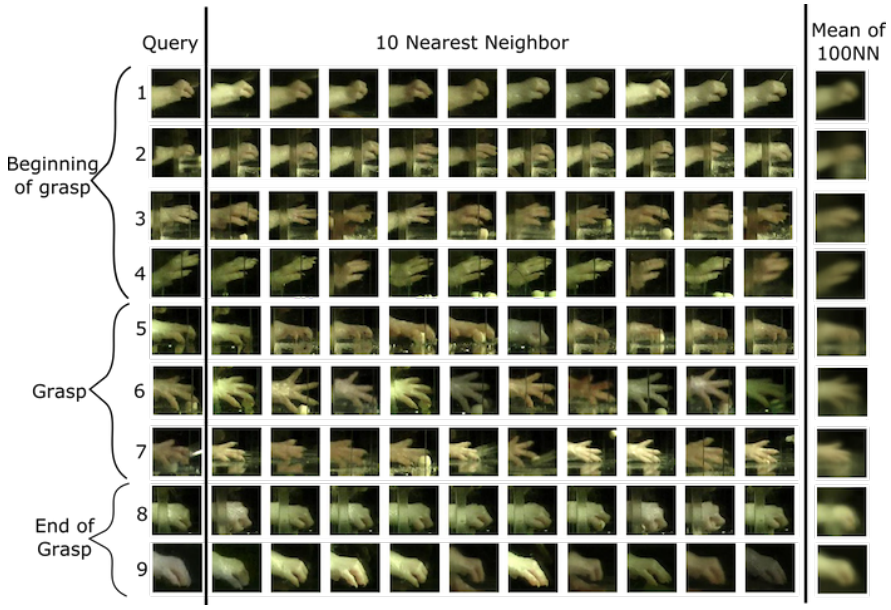


Figure 4.11: Qualitative evaluation of our posture encoding on the rat stroke dataset. We compute the cosine similarity between all samples given their posture encoding and show the nearest neighbors of several query frames. The last column provides an average image of the 100 nearest neighbors.

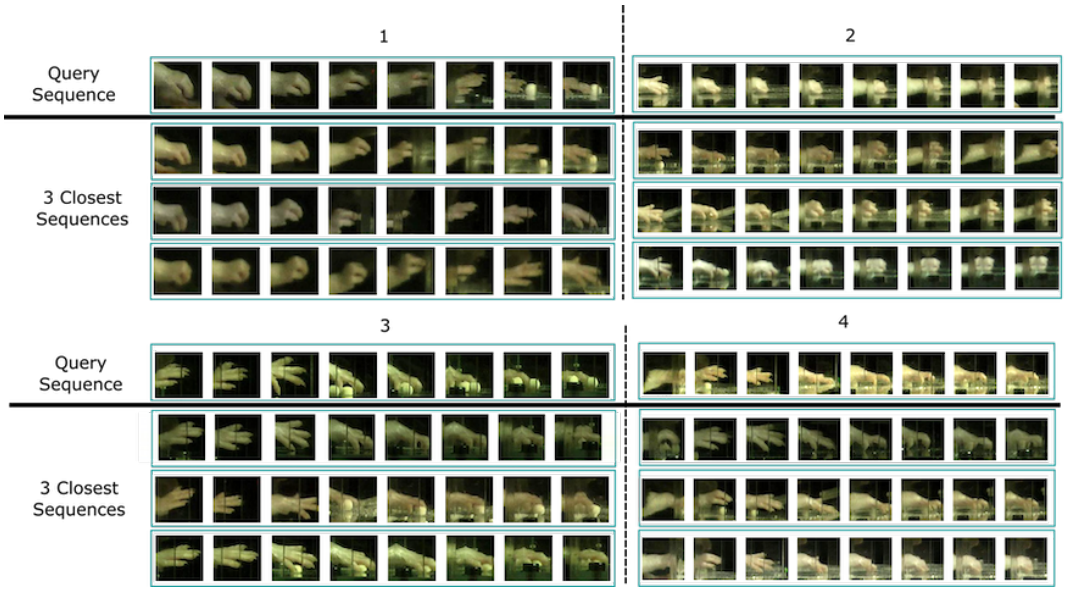


Figure 4.12: Qualitative evaluation of our behavior encoding on the rat stroke dataset. We use the LSTM features for computing the cosine similarities between all sequences and show the 3 closest sequences for 4 query samples.

between all samples of a dataset to determine the nearest neighbors. Figure 4.11 illustrates the nearest neighbors of several query postures of the rat stroke dataset given the learned posture representation, and Figure 4.12 qualitatively evaluates the behavior representation. It is apparent, that both representations are invariant to appearance differences and only encode the desired characteristics, i.e. posture and behavior. The same conclusion can be drawn from Figure 4.13 which

Figure 4.13: Qualitative evaluation of our posture encoding on the human gait dataset. We project our posture representation of 1000 randomly chosen postures to a 2D embedding using tSNE [178]. The walking gate can be reconstructed by following the circle anticlockwise and similar postures are located close to each other. The figure can be best viewed by zooming in on the digital version.

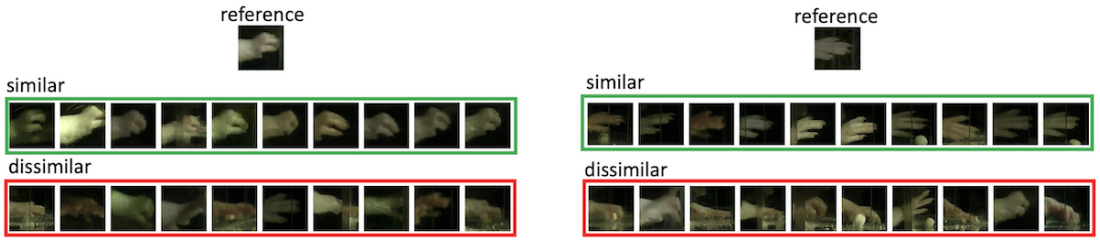


Figure 4.14: Exemplary depiction of two samples of the posture benchmark test set used to evaluate posture representations quantitatively.

Models	Accuracy(%)
ImageNet[90]	65.3
E_π^0	72
E_π^2	**85.6**

Table 4.2: Evaluation of posture representation using our benchmark test set for posture similarity.

Models	Accuracy(%)
Max frame similarity	74.1
Avg frame similarity	75.9
DTW [31]	76.8
ClusterLSTM	64.0
Ours^2	**80.5**

Table 4.3: Evaluation of behavior representation using the benchmark test set for sequence similarity.

shows the projection of 1000 postures of HG2DB from the posture encoding to a 2D embedding space.

Quantitative Results. We quantitatively evaluate the learned representations of the rat stroke dataset using a posture (individual frames) and a behavior (sequences) benchmark, which were manually labeled. Both benchmarks are composed of reference frames/sequences and 10 similar and 10 dissimilar manually selected samples have been labeled for each reference. Figure 4.14 shows two exemplary samples of the posture benchmark. We use 30 reference elements for evaluating the posture representation and 22 for the sequence evaluation. This test set consists of in total 4326 frames. For evaluating pose and behavior similarity, we use the learned feature representations for sorting the 20 labeled samples based on their similarity to the reference posture and sequence, respectively. We then report the relative number of samples which are labeled as similar within the first 10 samples after sorting. Table 4.2 shows the results for single frame posture similarity obtained by a network trained supervised on ImageNet [90], our E_π^0 and E_π^2 (the final posture representation obtained after two iterations). Note that the joint training of behavior and postures substantially improves the representation of individual postures. In Table 4.3 we compare the behavior similarity results achieved by our LSTM ordering task (*Ours*^{2}, last bootstrap retraining) with *Dynamic Time Warping* [31] (DTW), a direct stacking of single frame posture representations and an LSTM baseline model (*clusterLSTM*) based on sequence clustering. In case of stacking, we compute the similarity between two sequences by taking either the average or maximum of the pairwise similarities between single frames originating from the two sequences. *ClusterLSTM* corresponds to an LSTM network trained on a multi-class classification task rather than our proposed self-supervised ordering task. For that matter, we

create clusters of sequences using Dynamic Time Warping as distance measure between the sequences and train the network to separate different clusters from another. Table 4.3 shows that our LSTM ordering task outperforms all the other approaches. The weak performance of *clusterLSTM* underlines that training a behavior representation on discrete groups of sequences is not suited to learn fine-grained behavior similarities. We omit the results of the intermediate iterations (E_π^1, Ours^1) in Table 4.2 and 4.3, since the performances differ only marginally due to convergence of learning (as indicated in Table 4.1).

4.6.3 Fitness Prediction and Comparison with Previous Work

For neuroscientists, the primary goal of behavior analysis is to discover the degree of motor function impairment and to develop new treatments. For the task of single pellet grasping, a standard protocol has been proposed in [4] to judge grasping fitness. Experts assess grasping by scoring ten criteria including the pronation and supination of the paw (its turning) and the opening and closing of the digits. Averaging these scores then yields an indicator for the fitness of the motor function. Rather than trying to replicate the individual decisions that experts make, we propose to circumvent this tedious manual analysis by directly mapping sequences to a final fitness score.

Evaluation Protocols. We compare the resulting scores of our method that learns a non-parametric and unsupervised representation against two established methods for supervised, keypoint-based behavior analysis, DeepLabCut [112] and Jaaba [81]. The methods for comparison are summarized in Section 4.2. We train per approach a classifier for distinguishing healthy from impaired sequences. The training set is composed of 5000 sequences from pre-stroke videos (healthy) and 5000 from 2 days post-stroke videos (impaired). The trained classifiers then predict the healthiness for each grasping sequence on a separate test set of $\sim 90,000$ sequences. Per method, we tested several classifiers including support vector machines (SVM) [28], AdaBoost and multi-layer perceptrons and selected the classifier that obtains the best test accuracy. For our approach we use the trained behavior representation as feature input and train a support vector machine classifier [28]. For DeepLabCut we first fine-tune the provided network weights (pre-trained on human pose estimation) on detecting 14 keypoints on the forelimb (wrist, arm and 3 per finger for 4 fingers) using 1500 manually labeled frames. Then, we use the keypoint predictions of DeepLabCut on the grasping sequences to extract the trajectory and kinematic features. We use these features to finally train and test the classifier that aims to distinguish healthy from impaired sequences. The best results have been achieved with an AdaBoost classifier with Decision Trees as estimators. Jaaba receives the body-part location per video frame as input and automatically computes a set of hand-crafted features to train an in-build classifier.

Fitness Prediction. The animals in the rat stroke experiment were divided into 4 different treatment groups during a 35-day long recovery. We compute for every rehabilitation phase the relative number of sequences that are predicted as healthy given the evaluation protocols described above. In Figure 4.15 we

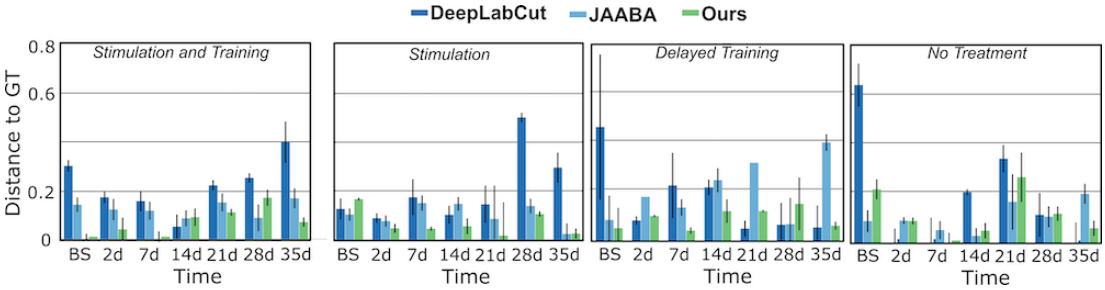


Figure 4.15: Prediction of grasping fitness of the rat stroke dataset during a 35-day long recovery for 4 treatment groups. We show for every rehabilitation phase the distance to the ground-truth scores determined by experts. "BL" are the baseline videos recorded after the training stage and before the stroke. All other phases are recordings after the stroke, e.g. "2d" represent videos from 2 days after the stroke and "35d" from 35 days after the stroke.

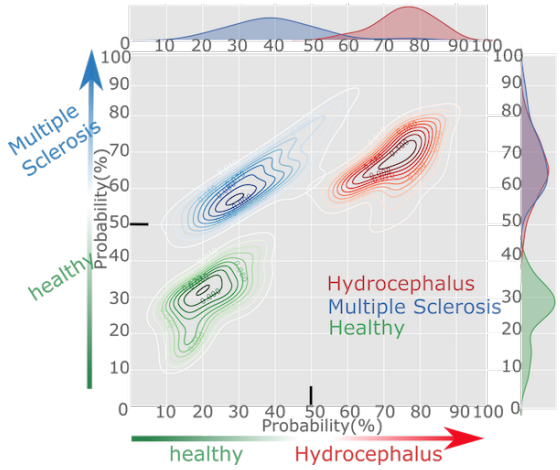


Figure 4.16: Disease classification of the subjects in HG2DB. We train two classifiers (healthy vs multiples sclerosis and healthy vs hydrocephalus) and report the 2D score distribution on test videos. Each axis represents one classifier and shows the probability of a subject suffering from one of the diseases.

show the results achieved with our method, DeepLabCut and Jaaba. Each bar represents the error from the expert ground-truth (GT) scores, as defined in [4]. Our results have a correlation of $0.933 \geq 0.005$ with the ground-truth scores whereas Jaaba and DeepLabCut produce a correlation of $0.820 \geq 0.072$ and $0.695 \geq 0.014$, respectively. This experiment demonstrates that our non-parametric model is able to better emulate the tedious manual expert scores in comparison to previous works.

4.6.4 Disease Classification

The human gait dataset contains patients that are diagnosed with either multiples sclerosis (MS) or hydrocephalus. We now show, that our learned behavior representation is able to not only distinguish healthy from impaired but also patients with different neurological diseases that both affect the motor function.

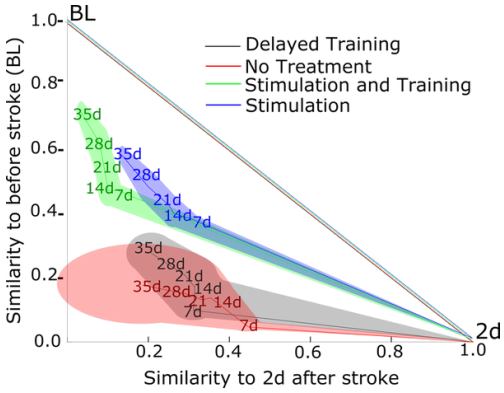


Figure 4.17: Rehabilitation analysis for four treatment cohorts of the rat stroke scenario. We show the similarity to healthy baseline behavior (BL, x-axis) and to impaired behavior (2d, y-axis). The cohorts with optogenetic stimulation show a strong improvement during recovery whereas animals of "Delayed Training" and "No Treatment" show signs of inadequate compensation.

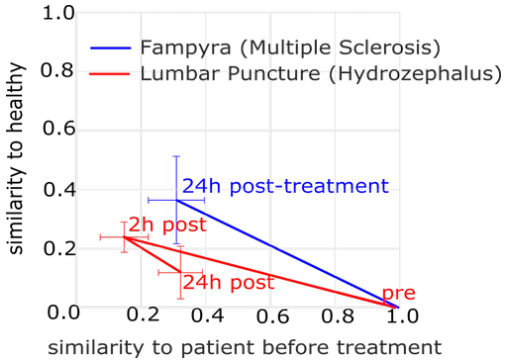


Figure 4.18: Rehabilitation analysis of the human gait dataset. We show the similarity of multiples sclerosis and hydrocephalus patients to healthy (y-axis) and pre-treatment (x-axis) behavior. The treatments for both diseases improve the patient's conditions, however, the treatment for hydrocephalus patients only persists in a short time window.

At first, we train one linear classifier (SVM) per disease by using the recordings from healthy subjects as positive examples and patients with MS or hydrocephalus as negatives ($\sim$ 1200 sequences per class). Then, we compute the scores of both classifiers on some test videos from all classes. Figure 4.16 shows the distribution of scores after concatenating the results of the classifiers per video and applying kernel density estimation. Each axis provides the probability of a subject to be affected by the respective disease or not. All classes are well separated and compact, indicating that our method is able to separate even diseases that both affect the motor function by only extracting information from the different walking styles.

4.6.5 Rehabilitation Analysis

The subjects of both medical scenarios received disease-specific treatments to improve motor function. If long-term recordings are given, behavior analysis can also reveal the subtle changes in motor function during recovery. In this section, we analyze the recordings from diverse recovery stages and compare the outcome of different treatments using our behavior representation.

Rat Stroke Model. The videos of the rat stroke scenario contain recordings from before the stroke was triggered and up until 35 days after stroke. Per treatment group, we relate the behavior to a large set of healthy baseline (BL) kinematics (before stroke) as well as impaired samples from 2 days post stroke (2d). We train a linear classifier (LDA) using $\sim$ 1000 sequences from BL and an-

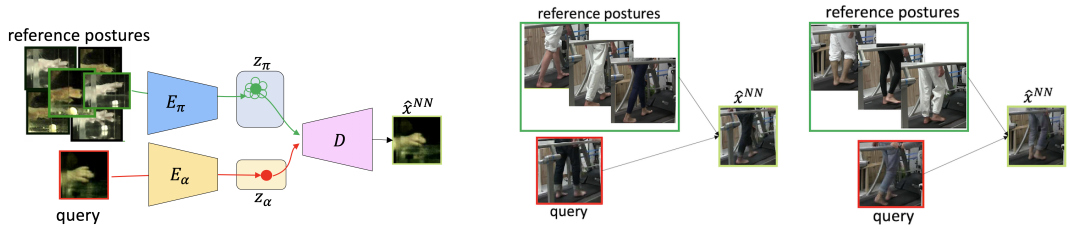


Figure 4.19: *Left:* Depiction of the procedure for resynthesizing healthy reference frames. We first obtain a reference posture by averaging the healthy nearest neighbor postures of the query frame in the posture encoding z_π. Then we combine that encoding with the appearance representation of the query subject and synthesize the healthy reference frame $\hat{x}^{NN}$ using the decoder D. *Right:* Exemplary results of resynthesized healthy reference frames on HG2DB.

other $\sim$ 1000 sequences from 2d behavior. During inference, we use the classifier scores as a measure of impairment for all videos recorded during recovery. Figure 4.17 displays the similarity to BL (y-axis) and 2d (x-axis) per cohort and for each week of recovery. The cohorts with optogenetic stimulation ("Stimulation" and "Stimulation and Training") steadily improve during recovery, reaching around 70% similarity to the baseline behavior and having almost no similarity with the impaired post-stroke behavior. Moreover, the figure not only shows, if the animal behavior gets closer again to their original state at baseline, it also reveals cases of unsuccessful recovery where behavior digresses from 2d, however without becoming more similar to baseline, e.g. the cohort "no treatment". This is a sign of inadequate compensation of impaired motor functions that differs significantly from true recovery.

HG2DB. To evaluate the effect of a treatment on human patients we compare the behavior after treatment with *(1)* healthy and *(2)* pre-treatment behavior for both diseases. We train a classifier using sequences from healthy and pre-treatment and show the development of motor function skills in Figure 4.18 after treatment. The treatment for multiples sclerosis patients (Fampyra) yields a significant improvement on the walking abilities. Lumbar puncture, the treatment for patients affected by hydrocephalus, leads to an improvement within the first 2 hours due to reduction of liquor pressure, but the experiment shows that the motor skills deteriorated again afterwards as expected by expert neuroscientists.

Figure 4.17 and 4.18 show that our behavior encoding is an effective representation for comparing different therapies and for analyzing the improvements in behavior during recovery. In the next sub-section, we analyze the abilities of our generative model for discovering and magnifying subtle posture differences.

4.6.6 Magnification

Section 4.5 introduced a generative model for amplifying subtle differences in behavior between an impaired patient and reference subject. This approach can extend medical diagnostics and reveal subtle deviations due to impairment that even a trained eye might easily overlook. Now, we demonstrate the applicability of our approach on the two medical scenarios and compare with previous work

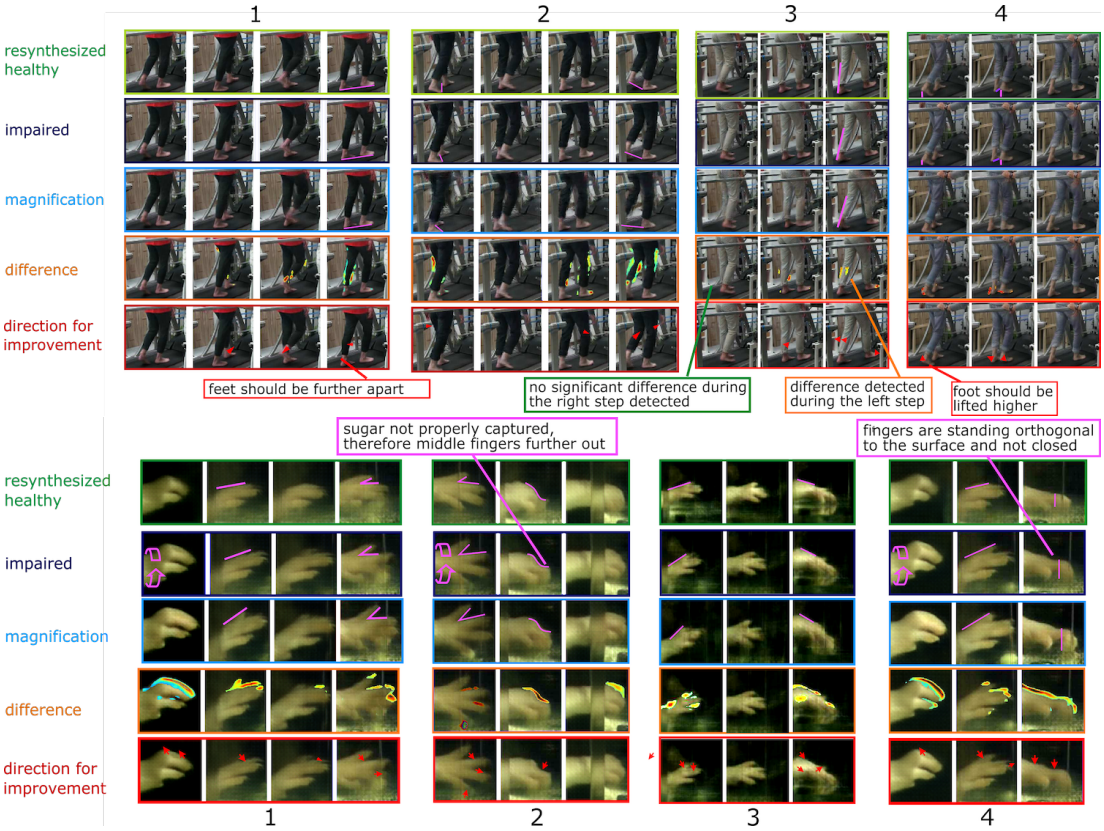


Figure 4.20: Results of our magnification approach on HG2DB (top) and the rat stroke dataset (bottom). We show the healthy reference posture (1st row) generated with the appearance of the impaired subject, the original impaired frames (2nd row), our magnification results (3rd row), a heat map that demonstrates the pixel differences between the healthy and impaired subject (4th row) and the direction for improvement using optical flow. We manually superimposed magenta markers to facilitate the perception of the differences for the reader.

on video motion magnification.

Resynthesized Healthy Reference. Appearance differences increase the difficulty of spotting behavior deviations across subjects. Therefore, to support a direct diagnostic comparison despite the appearance differences of the reference and impaired subject, we first generate an image that shows a healthy reference posture but has the same appearance as the impaired subject. This is only possible since our generative model learned to separate posture from appearance. We graphically show the procedure for generating the healthy reference frame in Figure 4.19 including three exemplary outputs on both medical scenarios. As can be seen, our resynthesized healthy reference frames $\hat{x}^{NN}$ contain the same type of posture as the healthy nearest neighbors and the same appearance as the query subject.

Our Results. In Figure 4.20 we demonstrate the abilities of our magnification approach on HG2DB (top) and the rat stroke dataset (bottom). Our magnification (3rd row) reveals the hardly visible differences in posture between the query (2nd row) and resynthesized reference (1st row). For instance, in the first example of Figure 4.20 our approach recognizes that the impaired subject is

Figure 4.21: Results of our magnification approach in comparison with the outputs generated by Oh *et al.* [125].

not able to perform a proper right step and therefore sets the foot down earlier than a healthy subject leading to a smaller distance between the feet. To help the reader spot subtle deviations, we manually superimposed (magenta) markers in the first three rows. We highlight the differences between healthy and impaired postures by an automatic heat map (4th row) and show how the affected body regions need to move to compensate the impairment (last row). The heat map is determined by computing the L2 distance in pixel space between the impaired and healthy frame. For calculating the *direction of improvement*, we employ the optical flow between the two images.

Comparison. Now we compare our results with the latest motion magnification approach by Oh *et al.* [125] using their published implementation on github [1]. In Figure 4.21 we directly compare our magnified outputs with the generations produced by Oh *et al.* [125]. Our approach (4th row) successfully magnifies the differences between healthy (1st row) and impaired (2nd row), whereas the motion magnification approach [125] (3rd row) is often not able to amplify the differences accordingly. The strategy by Oh *et al.* is not suited for evaluating posture deviations across subjects. Therefore, it outputs rather blurry and unrealistic looking images without magnifying the differences in the walking cycle.

Next, we evaluate how our method and the approach by Oh *et al.* perform if we ask to magnify between healthy subjects in contrast to using healthy and impaired subjects as before. The desired output is that the approach does not detect any behavior differences between healthy subjects and that the magnified generations do not differ from the original input. This would demonstrate that the method does not magnify arbitrary deviations caused by differences in appearance or normal posture variations, but only those due to impaired behavior. In Figure 4.22 we present the differences in pixel space between an original frame and its magnification using impaired sequences (top) or healthy subjects (bottom) as query. For our method, the differences are consistently larger for impaired patients (top) than for healthy ones (bottom). The differences for Oh *et al.* , on the other hand, indicate that this approach magnifies healthy and impaired behavior indiscriminately. This experiment shows, that our approach fulfills the requirement of only magnifying impaired behavior and not arbitrary deviations.

[1] `https://github.com/12dmodel/deep_motion_mag` (accessed August 2019)

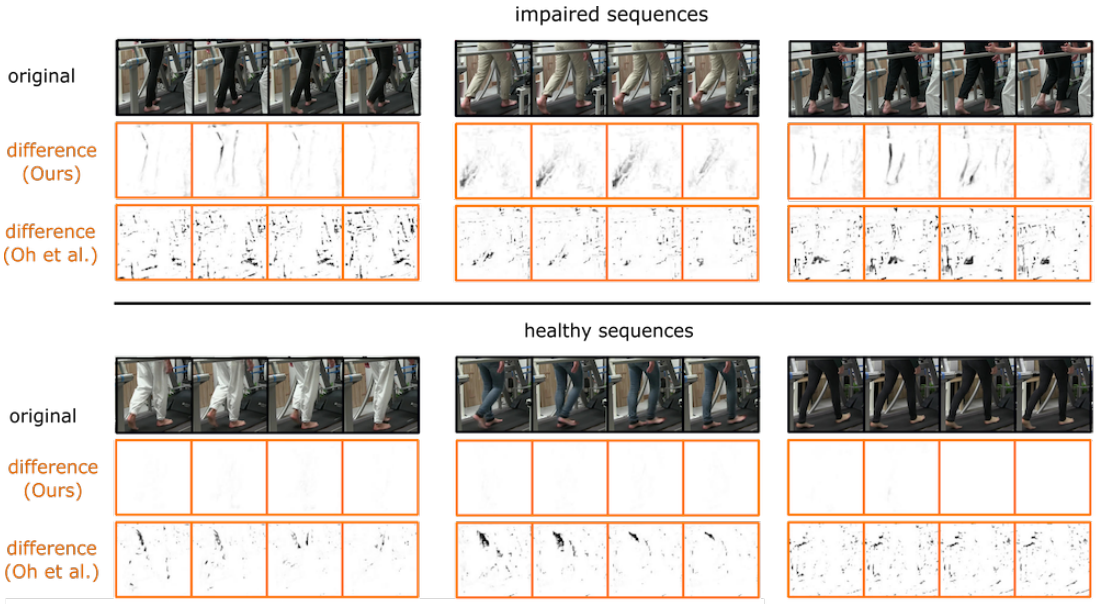


Figure 4.22: Evaluation of the magnifications between healthy subjects (bottom) in contrast to the magnified generations between impaired and healthy behavior (top). A magnification approach for motor behavior analysis should not magnify arbitrary differences, but only the ones we are interested in (e.g. due to impaired behavior). We evaluate this condition by calculating the differences in the pixel space between the original input image and the magnified output. The differences for magnifications between healthy subjects should be very small in comparison to amplifications between healthy and impaired.

4.7 Technical Details

All deep networks are implemented using the PyTorch[2] framework and the experiments were performed on a single Titan X GPU.

Posture and Behavior representation. The *ConvNet+LSTM* network corresponds to the model introduced in Section 3.3.1 with a foundation architecture based on AlexNet [90], i.e. five convolutional layers (*conv1* to *conv5*) with intermediate max-pooling and one fully connected layer (*fc6*). The temporal information is incorporated by using a long short-term memory layer with 1024 hidden nodes. For our experiments the *fc6* features are used as posture representation and the LSTM as behavior encoding. A final fully-connected layer acts as classifier for predicting if a sequence is shuffled or not. For training the network we use 24 sequences per batch and include a randomly permuted version each (in total 48 samples per batch). We use a binary cross-entropy loss and update the network using stochastic gradient descent with weight decay and a starting learning rate of 0.01 which we reduce after 200k iterations by a factor of 10. Our network runs in total for 350k iterations. We train in total two networks, one for each dataset: For the rat stroke model we use $\sim 100,000$ sequences for training with an average length of 8 frames per sequence and for the human gait dataset we employ $\sim 170,000$ sequences with an average length of 27 frames. The input frames have a size of 227×227 and as augmentation we randomly crop each frame and apply a random color jittering (the same augmentation for all frames belonging to the same sequence).

Generative model. The encoder-decoder architecture is composed of three networks: the posture encoder E_π, the appearance encoder E_α, and the decoder D. E_π is the same network as the "*ConvNet*" part described above until *fc6*. E_α is composed of 5 layers, each layer includes a convolutional layer, followed by batch normalization and LeakyReLU. The decoder has 6 layers: firstly, a fully connected layer with ReLU which receives the appearance and posture encoding as input (concatenated); the output is then reshaped and input into 5 transpose convolution layers. Each transpose convolution layer contains upsampling, a convolution operation with kernel size 3, batch normalization, and LeakyReLU. The network is updated using Adam solver, with a learning rate of $5 \cdot 10^{-4}$, while keeping the default values for the other hyper-parameters. The model is trained for 50 epochs.

4.8 Discussion

In this chapter we have demonstrated the applicability of unsupervised machine learning to motor behavior analysis. We have used our temporal permutation task to learn a fine-grained posture and behavior representation without requiring experts labelling hours of recordings. The utility of the resulting features is evaluated on two biomedical scenarios with rodents and humans. We have shown that our representation is able to *(i)* analyze the rehabilitation based on different treatments, *(ii)* distinguish diseases and *(iii)* reproduce manual fitness scores of

[2] `http://pytorch.org/`

experts. In addition to it, our approach compares favorably against supervised behavior analysis approaches.

This chapter has introduced additionally a magnification approach that facilitates the perception of subtle impaired behavior. Our generative model discovers the posture deviations between impaired patients and healthy reference subjects and generates new images that display the exaggerated posture differences. We have showed that our approach produces realistic magnifications of impaired behavior that is almost invisible to the naked eye. Comparing to the latest motion magnification approach has demonstrated the importance of our disentangled representation in order to enable a meaningful magnification across subjects.

The medical scenarios introduced in this chapter were recorded indoors and in a very restricted setup with similar lightning conditions. In the next chapter we present a robust magnification approach that is able to emphasize posture deviations across subjects in less restricted recording setups.

Chapter 5

Robust Magnification

Magnifying posture deviations across subjects is a new and challenging task. In the previous chapter we have shown its importance especially for analyzing the behavior of patients that suffer from a neurological disease. The videos of the medical scenarios were recorded in a very controlled setup with similar lightning and camera settings. In this chapter we propose a robust magnification approach that enables us to compare videos recorded in different setups as, for instance, occurring in sports for comparing and identifying sub-optimal movements. To transfer appearance across subjects onto a magnified posture, we introduce a novel loss for disentangling appearance and posture in an autoencoder. Moreover, we incorporate the magnification process already into the training so that the model learns to produce realistic images even for strong magnifications. Experiments confirm that our approach improves upon previous works.

5.1 Introduction

Motion magnification techniques aim to detect and amplify small motion in order to facilitate its perception. Previous approaches [100, 183, 48, 173, 174, 166, 199, 125] successfully magnify the variations of a specific target object within the same video. They are addressing, for instance, the magnification of the "wobbling" motion of a pupil during the idle state or the breathing movement of a baby's belly (to detect breathing interruptions of babies during sleep). These methods can handle intra-video appearance variability but fail at the differences across subjects and videos. In the previous chapter we have demonstrated the importance of a magnification approach that can selectively amplify subtle posture differences across subjects. Motor behavior analysis highly benefits from such an advanced method. Exaggerating the often subtle deviations between impaired behavior and healthy reference movements facilitates the interpretation of the symptoms and assists researchers and doctors to understand the disorder. The videos of the medical scenarios analyzed in the previous chapter were recorded in a very controlled setup: indoors with similar lightning and background and the cameras were placed roughly at the same position. However, not only biomedical scenarios with very controlled setups benefit from magnifying posture deviations. In sports, for instance, analyzing an athlete's execution of a specific action to discover the mistakes and to adjust the movement accordingly is crucial to increase the overall

performance. This problem, however, requires a magnification method that also functions outdoors and that it is able to compare behavior recorded with different setups.

Therefore, we propose in this chapter a novel unsupervised and more robust magnification model that is not restricted to indoor recordings and that can be used for numerous other applications such as sports. As before, we aim to explicitly disentangle posture and appearance in an autoencoder. However, in contrast to our previous method, we propose a novel loss that better enforces disentanglement despite larger appearance deviations. Moreover, the new loss enables us to train all weights of the generative model from scratch without requiring a pre-trained posture representation. Magnification typically aims at generating new, exaggerated postures that are not in the training set. Therefore, it is difficult to produce real-looking images especially if the dataset contains videos with high variety in appearance. Consequently, we need to integrate the magnification process already into the training of the autoencoder to generate realistic images even for strong amplifications. In contrast to the most recent video magnification approach [125] whose training works on synthetic data, we introduce an approach that can directly train on inferred magnifications of real data without requiring supervision.

Experiments demonstrate that our method leads to more detailed and realistically looking magnifications. It also improves previous performances in terms of quality and on the downstream task of discovering posture deviation due to impairment.

5.2 Robust Magnification across Subjects

Before describing our approach for robust magnification, we redefine the problem of magnifying posture deviations. Then, we present our unsupervised approach for separating posture and appearance to assure that the magnification only alters the posture and not the appearance. Finally, we describe our method that enables us to directly train the magnification on real data.

5.2.1 Problem Definition

Given a frame x^q of a query video showing a subject performing a particular action, the objective is to amplify the differences of x^q to a reference frame x^r and to generate the magnified image

$$\hat{x}^m = m(x^q | x^r, \lambda) \tag{5.1}$$

with m a magnification function, λ the amplification intensity and x^r a frame from a different video and subject. As in the previous chapter, we aim to magnify only the posture deviations while leaving the appearance unaltered. Thus, we require an autoencoder (AE) with two encoders, E_π for posture and E_α for appearance, and a decoder D to generate images. A query image x^q is then reconstructed as follows

$$\hat{x}^q = D\big(E_\pi(x^q), E_\alpha(x^q)\big). \tag{5.2}$$

The same separation is applied for x^r. Given the AE with two encoders, we can update Equation 5.1 for generating the magnified image $\hat{x}^m$ as follows,

$$\hat{x}^m = D\Big(m_\pi\big(E_\pi(x^q)|E_\pi(x^r), \lambda\big), E_\alpha(x^q)\Big) \tag{5.3}$$

with m_π a magnification function in the posture space which is further defined in Section 5.2.3. In order to obtain a meaningful magnification $\hat{x}^m$ we require a training procedure which teaches the network to distinguish posture from appearance and to produce realistic outputs. In the next two sub-sections we propose our novel training strategy which enables a robust magnification of posture deviations across subjects.

5.2.2 Disentanglement for Magnification

Magnifying posture deviations involves the comparison of subjects with different appearances. To transfer the posture from x^r to x^q it is crucial to obtain a posture encoding that does not contain any subject-specific information. Furthermore, we require a pure appearance representation of x^q for generating the magnified frame.

The posture and appearance encoders are considered to be disentangled if the posture encoding is invariant to appearance changes and vice-versa. The state-of-the-art in motion magnification [125] induces an invariance to intensity changes in the pose encoder by introducing a regularization loss. The objective enforces the pose representation of a color perturbed frame to be the same as the original frame. We also apply a color transformation τ to the input image x^q, but we additionally alter the posture by choosing a random frame $x^{q\prime}$ from the same video as x^q ($x^{q\prime}$ contains the same appearance as x^q, but a different posture). We insert $\tau(x^q)$ into the posture encoder E_π, $x^{q\prime}$ into the appearance encoder E_α and generate the reconstruction with the decoder D (see Figure 5.1 A). A perfect reconstruction is only possible if the AE extracts the posture information from $\tau(x^q)$ and the appearance information from $x^{q\prime}$. We train our model by minimizing the following reconstruction loss:

$$\mathcal{L}_{\text{rec}} = d(\hat{x}^q, x^q) \tag{5.4}$$

with $\hat{x}^q = D(E_\pi(\tau(x^q)), E_\alpha(x^{q\prime}))$ the reconstruction and $d(\bullet, \bullet)$ the perceptual distance [78]. Oh *et al.* [125] also employ the reconstruction loss but require an additional regularization objective to enforce invariance.

Despite the color transformation, the input $\tau(x^q)$ to the posture encoder still contains appearance information such as the background scene or the type of clothes worn by the subject. This would allow the decoder to find a lazy solution by mainly leveraging the posture encoding to reconstruct x^q as good as possible without considering the appearance encoding $E_\alpha(x^q)$. In contrast to motion magnification of single objects, the magnification of posture deviations transfers postures across subjects with different appearances. Hence, it requires a stronger separation of posture and appearance. For that reason, we introduce a novel loss discouraging our model to correctly reproduce the image if one of the two encodings is ignored. In practice, we generate "fake" images by exchanging the

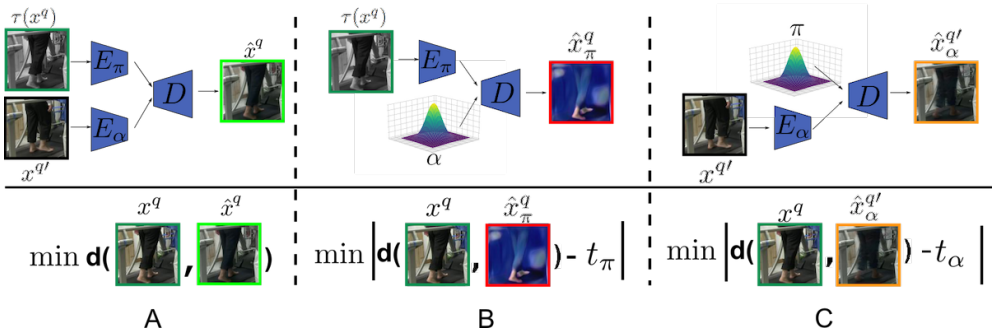


Figure 5.1: Disentanglement for Magnification. *A*: The image x^q is reconstructed by using the color transformed image $\tau(x^q)$ as input to the posture encoder and a random frame $x^{q\prime}$ (same video as x^q) as input to the appearance encoder. The reconstruction loss minimizes the distance x^q and the reconstruction $\hat{x}^q$. *B* and *C*: To enforce meaningful information in both encoders, we teach the network to generate deficient images when only one of the two encodings is used. We exchange one of the two encodings with Gaussian noise, producing "fake" images and require a distance $t_\bullet$ between the original and the fake image.

encoding of either appearance or posture with random Gaussian noise. Then we teach the network that an image reconstructed without one of the two encodings (fake image) is lacking an important component and should therefore not be able to fully represent the original input image. We define the reconstruction with fake posture as

$$\hat{x}_\alpha^{q\prime} = D\big(\mathcal{N}(0,\sigma), E_\alpha(x^{q\prime})\big) \tag{5.5}$$

and with fake appearance as

$$\hat{x}_\pi^{q} = D\big(E_\pi(\tau(x^q)), \mathcal{N}(0,\sigma)\big). \tag{5.6}$$

The generation with fake images is visually illustrated in Figure 5.1 B and C. We enforce a distance between the input and fake image to be close to a target value $t_\alpha, t_\pi > 0$. These values represent the lower bound on how close $\hat{x}_\alpha^{q\prime}$ and $\hat{x}_\pi^{q}$ are allowed to approach the original input x^q during training. We update our model using the loss

$$\mathcal{L}_{\text{dis}} = \big\|d(x^q, \hat{x}_\pi^q) - t_\pi\big\|_1 + \Big\|d(x^q, \hat{x}_\alpha^{q\prime}) - t_\alpha\Big\|_1 \tag{5.7}$$

with $d(\bullet,\bullet)$ being the perceptual distance. Note, that for the distance to $\hat{x}_\alpha^{q\prime}$ we compare with x^q since x^q and $x^{q\prime}$ contain the same appearance and therefore $\hat{x}_\alpha^{q\prime}$ should be equal to $\hat{x}_\alpha^q$ after optimizing the network.

At this point, both terms of $\mathcal{L}_{\text{dis}}$ are optimized independently from each other and one might be easier to minimize than the other. However, to successfully generate $\hat{x}^q$ we require both the posture and appearance encoding to be equally advanced. We found empirically that if one encoder outperforms the other, the weaker one has problems to catch up. Therefore, we balance the encoders by relating the target values t_α and t_π with the reconstruction quality of the opposite terms,

$$t_\pi = d(x^q, \hat{x}_\alpha^{q\prime}) + \gamma_\pi, \tag{5.8}$$

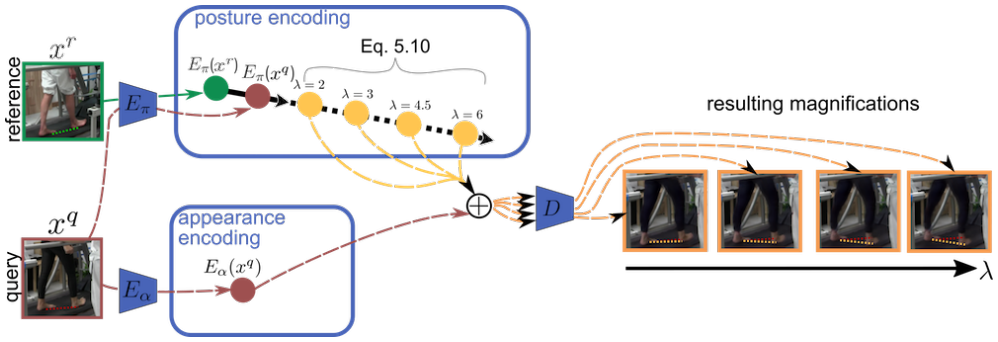


Figure 5.2: Illustration of the magnification process. We amplify posture deviations between a query frame x^q and a reference frame x^r by extrapolating the posture differences in E_π. D receives the appearance encoding of x^q concatenated with the amplified posture encoding as input and generates the resulting magnifications. We show the output using 4 different amplification factors λ.

$$t_\alpha = d(x^q, \hat{x}^q_\pi) + \gamma_\alpha. \tag{5.9}$$

with γ_π and γ_α being fixed margins. By coupling the margins to another, we force each encoder to make equal progress. If, for example, the reconstruction quality of $\hat{x}^{q\prime}_\alpha$ increases (and therefore the distance to x^q decreases), t_π decreases as well according to Equation 5.9 and forces therefore $d(x^q, \hat{x}^q_\pi)$ in Equation 5.7 to be smaller than $\hat{x}^{q\prime}_\alpha$ by a margin of γ_π. Overall, this leads to a loss which is less sensitive to the fixed margins.

In the next section, we introduce our approach that enables us to directly train the magnification of posture deviations on real data.

5.2.3 Learning to Magnify

The magnification in the posture space usually leads to novel poses. However, it is difficult for a generative model to produce postures never seen during training, especially with a high variety in appearance. In particular, we require a model that is *(i)* able to precisely transfer the magnified posture m_π to the pixel space and *(ii)* sensitive to small encoding differences. Thus, the magnification should be included directly into the training process. Since ground-truth magnifications are not available, we cannot simply employ the reconstruction loss. Oh *et al.* [125] tackled this problem by creating a synthetic dataset to simulate the magnification of motion. We propose an alternative approach that allows us to directly train magnification on real data without requiring ground-truth images. In this way, our model produces more fine-grained and realistically looking results which is demonstrated in the experimental section.

As defined in Equation 5.3, we generate a magnified frame $\hat{x}^m$ by combining the magnified posture encoding m_π with the appearance encoding $E_\alpha(x^q)$. For computing m_π we first calculate the difference between x^q and x^r in the posture encoding. Then we amplify the posture deviation in the direction of $E_\pi(x^q)$. This procedure can be practically realized by extrapolating the posture differences,

$$m_\pi(E_\pi(x^q)|E_\pi(x^r), \lambda) = E_\pi(x^r) + \lambda\,(E_\pi(x^q) - E_\pi(x^r)) \tag{5.10}$$

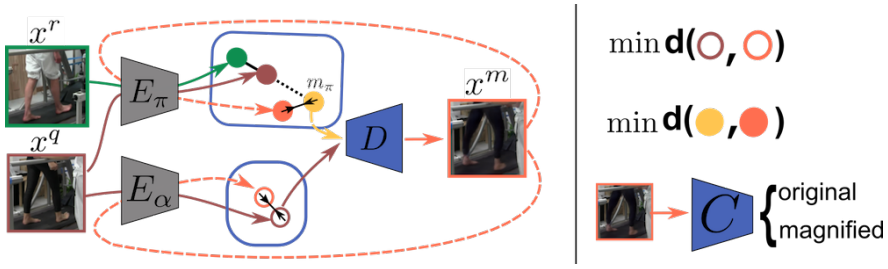


Figure 5.3: Learning to Magnify. Our magnification loss forces the decoder to precisely transfer the magnification m_π to the RGB space by re-encoding the magnified image $\hat{x}^m$ and minimizing the distance between the original magnified posture (yellow filled point) and its re-encoded posture (orange filled point). The same is applied for the appearance encodings (orange empty point and dark red empty point). Finally, an adversarial discriminator C enforces the generation of realistic-looking magnified images.

with $\lambda > 1$ being the magnification factor. Figure 5.2 visually depicts this procedure.

During training, we require a reference frame x^r containing a slightly different posture as x^q since we aim to amplify subtle posture differences. This can be chosen automatically by using the k-th nearest neighbor (NN) of x^q (excluding frames from the same video) with k randomly chosen from the range $[10, 20]$. By selecting different reference frames for x^q every epoch, the model learns to cover a wide range of posture deviations. We can now generate a magnified frame $\hat{x}^m$ for each x^q with respect to the sampled reference frame x^r. The magnification of posture deviations requires a decoder able to precisely transfer the magnified posture encoding m_π to the pixel space without distorting or losing any information about the new posture. In particular, our model should reach a fixpoint with respect to m_π, i.e. m_π should be equal to the re-encoded decoded m_π,

$$m_\pi \stackrel{!}{=} E_\pi\big(D(m_\pi, \bullet)\big). \tag{5.11}$$

To meet this requirement, we introduce a fixpoint loss that minimizes the distance between the re-encoded magnified frame $E_\pi(\hat{x}^m) = E_\pi\big(D(m_\pi, E_\alpha(x^q))\big)$ and the original magnification m_π. Figure 5.3 illustrates this procedure. We also minimize the distance between the respective appearance encodings $E_\alpha(\hat{x}^m)$ and $E_\alpha(x^q)$ to ensure a consistent appearance decoding. Our model is then updated with the following fixpoint loss

$$\begin{aligned}\mathcal{L}_{\text{fix}} =& \|E_\pi(\hat{x}^m) - m_\pi\big(E_\pi(x^q)|E_\pi(x^r), \lambda\big)\|_2^2 \\ &+ \|E_\alpha(\hat{x}^m) - E_\alpha(x^q)\|_2^2 .\end{aligned} \tag{5.12}$$

We only update the decoder with $\mathcal{L}_{\text{fix}}$ since the purpose of $\mathcal{L}_{\text{fix}}$ is to improve the generation of magnified images.

To encourage the decoder to produce realistically looking images, we introduce an adversarial loss. A discriminator C is trained to distinguish between real images x^q and magnified images $\hat{x}^m$ by maximizing the objective $\mathcal{L}_{\text{A}}(C, D)$

proposed by [62],

$$\begin{aligned}\mathcal{L}_{\mathrm{A}}(C, D) = &\mathbb{E}_{x\sim p_{\mathrm{data}}(x)}\left[\log C(x)\right] \\ &+ \mathbb{E}_{\hat{x}\sim p_{\mathrm{mag}}(x)}\left[\log\left(1 - C(\hat{x})\right)\right]\end{aligned} \tag{5.13}$$

with p_{data} the data distribution and p_{mag} the distribution of magnified images. The decoder is then trained by additionally minimizing $\mathcal{L}_A$. The adversarial loss allows us to visualize the differences in posture with higher magnification factors without generating artifacts or unrealistic images.

We summarize the losses described in this section as

$$\mathcal{L}_{\mathrm{mag}} = \mathcal{L}_{\mathrm{A}} + \beta\mathcal{L}_{\mathrm{fix}} \tag{5.14}$$

with $\beta = 2$.

Our model is then updated with the following final loss,

$$\mathcal{L} = \mathcal{L}_{\mathrm{rec}} + \mathcal{L}_{\mathrm{dis}} + \gamma\mathcal{L}_{\mathrm{mag}} \tag{5.15}$$

with $\gamma = 0.5$ and $\mathcal{L}_{\mathrm{mag}}$ is only being used to update the decoder.

5.3 Experiments

We evaluate our approach on three different scenarios and compare our results with previous work on motion magnification. First, we introduce the datasets and perform qualitative and quantitative evaluations. Then, we show that every component of our model is important for generating meaningful magnifications through ablation studies.

5.3.1 Datasets

Magnifying posture deviations is a challenging and relatively new task. We introduce three datasets showing three different actions for the task of magnifying posture deviations across subjects. It is particularly important that the dataset contains subjects with different appearances to analyze the abilities of our model to transfer posture from one subject to another. Our datasets cover the following actions: *(1)* walking on a treadmill, *(2)* swinging a golf club and *(3)* moving the pupil of a person's eye.

HG2DB. The first dataset has been already introduced in Chapter 4 for behavior analysis (for visual examples please see Figure 4.6 and 4.7). As a reminder, several patients, that are suffering from a neurological disease, are recorded while walking on a treadmill. The dataset also contains videos of healthy subjects which are used as reference. All videos were recorded indoors with a consistent camera setup. HG2DB contains in total 229 videos with around 700 frames each, leading to a total number of 172,288 frames.

Golf Swing. We collected several videos from YouTube[1] showing several golfers performing a golf swing on different tournaments. The videos are recorded

[1] `https://www.youtube.com/user/GolfswingHD/`, Accessed in August 2019

Figure 5.4: Overview of the Golf Swing dataset.

Figure 5.5: Depiction of the 10 subjects in CUEye with 3 different eye colors (brown, blue and green) and both genders.

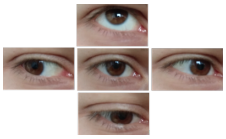


Figure 5.6: Possible postures in CUEye.

in slow-motion (120 fps) making them suitable for our scenario since many subtle differences in posture are represented. Our dataset has an overlap with the videos collected by Guha *et al.* [8] with the main difference that we use purely videos with a high frame rate. Overall, we collected 48 videos with a total number of 7000 frames. For our experiments, we cropped the frames so that the person-of-interest is located in the center which results in frames with an average size of 600x600. Figure 5.4 shows a subset of available subjects in Golf Swing and various types of postures. Golf Swing is more challenging than HG2DB since the videos were recorded from different tournaments (*i.e.* different backgrounds, lightning etc.) and they contain the full body of the person (*i.e.* more degrees of freedom regarding posture changes).

Close-Up Human Eye Dataset (CUEye). Even though eyes seem to be static if no direct movement is triggered by the person, the pupil still moves very subtle, often referred to as 'wobbling'. Magnifying posture deviations is an excellent tool to increase the visibility of this motion. We collected 10 videos showing close-up recordings of the eye from 10 different subjects (one video each) with three different eye-colors (brown, blue and green). The videos are recorded with a standard HD camera with 30 frames per second and an average length of 25 seconds. For the experiments, we cropped the videos so that only the eye is shown which results in frames with an average size of 500×350 (width×height). The subjects first move their eyes (left, right, up and down) to allow the generative model to differentiate between pose and appearance. This is followed by 5 seconds of starring used for evaluating if our approach is able to magnify the 'wobbling' effect of the pupil. We provide an overview of the subjects and occurring postures in Figure 5.5 and 5.6.

5.3.2 Qualitative Results

Figure 5.7, 5.8 and 5.9 show magnified images generated by our model for all three datasets.

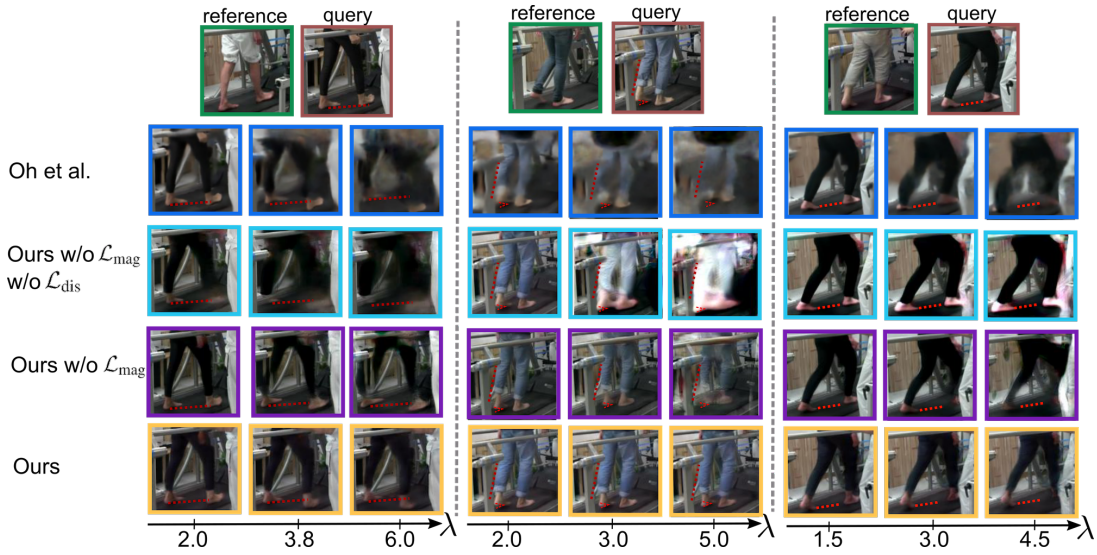


Figure 5.7: Qualitative Comparison on HG2DB. We show the magnification of posture deviations between a reference and query frame (first row) using the approach by Oh *et al.* [125] (2nd row), our model without $\mathcal{L}_{\text{dis}}$ and without $\mathcal{L}_{\text{mag}}$ (3rd row), our model without $\mathcal{L}_{\text{mag}}$ (4th row) and our final model (5th row). We manually superimposed red markers to facilitate the perception of the small differences and changes in the magnified images. The markers represent the posture of the query subject and are the same throughout a specific example. *Left*: The query subject keeps its legs more parallel than the reference subject. Our model exaggerates this behavior until the legs of the query subject are completely parallel. *Middle*: The query subject does not raise its left foot properly. Our magnifications visualize the differences until the entire left foot touches the treadmill. *Right*: The query subject performs bigger steps and our model further increases the distance.

Figure 5.7 demonstrates our results on magnifying posture deviations on HG2DB (5th row; yellow border) given a reference and query frame (first row). We show the output with three different magnification intensities λ. Our model is able to detect the posture differences and represent the magnifications on realistically looking images.

In Figure 5.8 we show our results (3rd row) on Golf Swing. Even though the dataset is very challenging due to the possible posture changes in arms and legs and the different recording locations, our robust model is able to magnify the differences between the reference and query frame. In particular, the example in the middle shows that our model can even magnify arms and legs at the same time.

Figure 5.9 displays the magnification of the subtle movements of a pupil while the eye is in idle state. Instead of comparing the posture deviations across different subjects, we first compute the pupil's movement in time of a query subject (top left) and transfer this motion to several target subjects with different appearances (right). Our model successfully detects the very subtle motion of the query subject and is able to transfer this motion to other subjects.

Comparison with Previous Work. Previous work addressed the task of

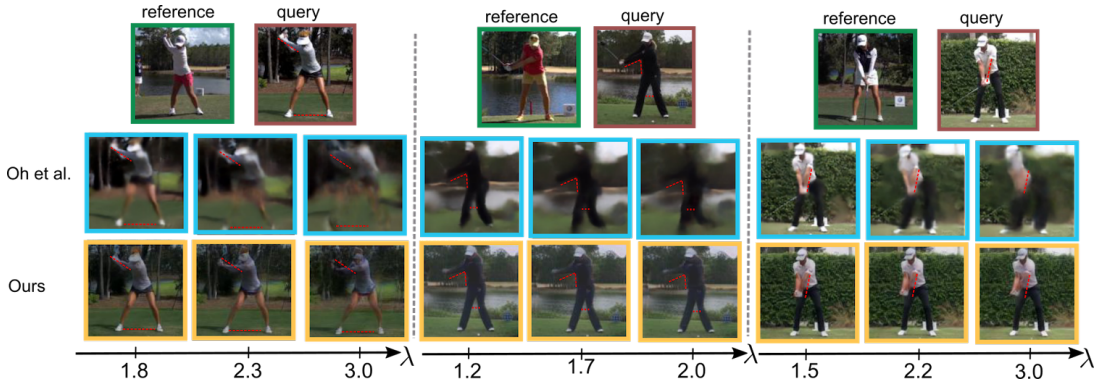


Figure 5.8: Qualitative Comparison on Golf Swing. We show the magnifications produced by our model and compare with previous work [125] (best viewed by zooming in on the digital version). The red markers indicate the posture of the query subject and are the same throughout a specific example. *Left*: The legs of the query subject are further apart, and the arm is kept lower. Our model further increases the distance of the legs and lowers the arms on the generated images. *Middle*: The right knee of the query is twisted inside, and the arms are kept higher. Our approach magnifies both by further twisting the knee and raising the arms. *Right*: The reference subject is holding its arms more centered than the query subject. Our model magnifies the deviation by slowly moving the arms of the query subject to the left.

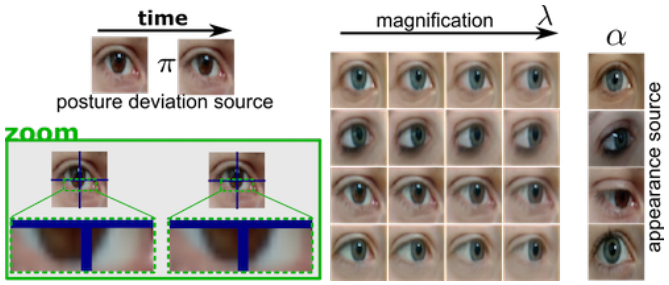


Figure 5.9: Magnification Results on CUEye. Detection of subtle posture differences in the pupil given a query movement (top left) and a target appearance (right). Bottom left shows a close up of the pupil with a blue grid as guide. The zoom shows a tiny motion from left to right. Given one of the target appearances shown on the right, our model can transfer and magnify the left-right movement from the query to the target appearance.

magnifying subtle motion within the same video [100, 183, 48, 173, 174, 166, 199, 125], but not across subjects with different appearances. Considering all motion magnification approaches, [125] has the highest potential to address the more complex scenario due to their usage of a generative model with a shape and texture representation. Therefore, we qualitatively compare our results in Figure 5.7 and Figure 5.8 (and quantitatively in Table 5.1) with [125] on the task of magnifying posture differences. We use the official implementation of [125] from their repository. Both figures show that Oh *et al.* [125] is not specialized on magnifying posture deviations. Their model is only invariant to intensity changes and also modifies the background and appearance of the subject. This leads to

Figure 5.10: Fake appearance images. The images are generated using the frames in the first row as input to the posture encoder and random Gaussian noise which replaces the appearance representation.

Figure 5.11: Fake posture images. The images are generated using the frames in the first row as input to the appearance encoder and random Gaussian noise as posture encoding.

very blurry and unrealistic images.

We additionally compare our final model without the newly introduced losses $\mathcal{L}_{dis}$ and $\mathcal{L}_{mag}$. Both losses are especially for larger λ essential to generate meaningful magnifications. In contrast, our final model is able to precisely magnify the posture differences and to generate realistic outputs without altering the appearance.

Fake Images. In order to better comprehend "fake" images, we provide examples with fake posture and fake appearance. In particular, in Figure 5.10 we show images generated with random Gaussian noise as appearance encoding ('fake appearance' $\hat{x}_{\pi}^{q}$) and in Figure 5.11 we display images generated with random Gaussian noise as posture encoding ('fake posture' $\hat{x}_{\alpha}^{q\prime}$). As intended, the images shown in Figure 5.10 only represent the posture and do not contain any appearance information; while the images in Figure 5.11 contain appearance information such as the background or color of clothes but display all subjects with the same average posture indicating the lack of posture information.

5.3.3 Quantitative Analysis

Classification of Impairment. Our model generates novel magnified postures not present in the given dataset. Hence, we cannot directly evaluate our magnified images due to missing ground-truth magnifications. As an alternative, we introduce a quantitative evaluation based on the health condition of patients in HG2DB.

We train two linear (binary) classifiers, both on healthy vs unhealthy samples. One classifier is trained with the original images (*Original*) and the second one

Classifier trained with	Postures										AVG
	1	2	3	4	5	6	7	8	9	10	
Original	58.9	61.3	63.2	53.3	55.9	51.7	58.3	50.3	50.7	61.2	56.5 ± 4.5
Oh *et al.* [125]	60.2	61.5	64.1	53.5	56.1	52.0	59.4	52.8	51.2	61.4	57.2 ± 4.4
Ours	**70.4**	**66.7**	**72.0**	**68.3**	**71.8**	**67.6**	**69.6**	**65.3**	**59.5**	**65.7**	**67.7 ± 3.5**

Table 5.1: Classification of Impairment. We report the test accuracies (%) per posture achieved by binary classifiers (healthy vs impaired) trained and tested on the key-points of (i) the original data, (ii) the magnified images generated by previous work on motion magnification and (iii) our magnified images. A visual example of postures 1 to 10 can be found in the supplementary material.

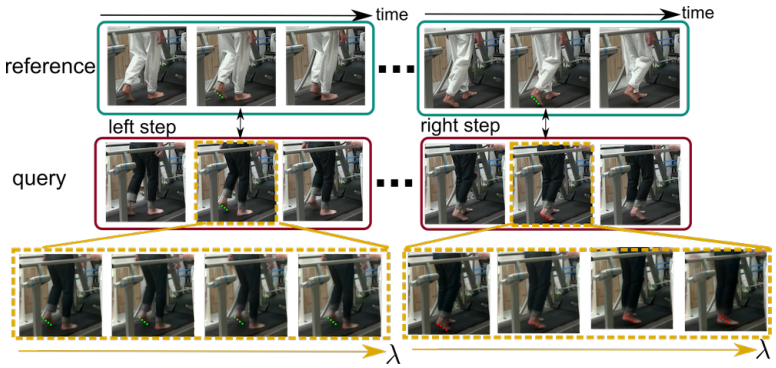


Figure 5.12: Magnifying Posture Deviations as a Medical Tool. Deviations between healthy (first row) and impaired (second row) is amplified in the generated images (third row) for a better analysis of the disease status. The patient only shows difficulties during the right step. Increasing λ emphasizes the deviation. We manually superimposed markers that indicate the angle of the right foot to facilitate the perception of the differences. The same marker is superimposed on the magnifications with different λs (3rd row). Clearly, larger magnifications λ (towards right) show a significantly changed foot angle in comparison to row 2 or 1. In contrast, the angle does not change for the healthy left leg's step.

with the magnified generations (*Ours*). The goal is to evaluate if the magnification improves the classification of impairment.

The classification should be subject independent and only based on posture information. For this specific experiment, we employ keypoints to represent postures since these correspond best to how humans perceive postures. In particular, we use DeepLabCut [112] for detecting the following 8 keypoints: left/right hip, left/right knee, left/right toe and left/right heel. The detector is trained with manually annotated frames of HG2DB. The keypoints are normalized using 'left hip' as origin to assure that they are comparable across different videos.

We sample 10 diverse linearly spaced postures from a complete walking cycle sequence and perform the quantitative analysis independently per posture. We provide a visual example of the postures in Figure 4.7. For every posture and subject (including all subjects in the dataset) we collect 10 Nearest Neighbors resulting in $10 \times$ *number of subjects* samples per posture for training and testing. Different postures require different magnification intensities to render visible posture discrepancies between healthy and impaired subjects. Therefore, we generate

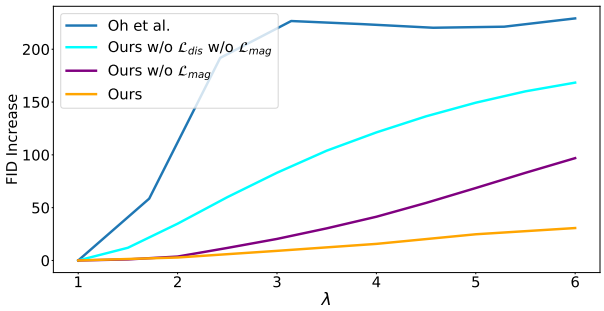


Figure 5.13: Quality of Visualizations - FID. We display the absolute FID increase relative to $\lambda = 1$ for different intensity values λ using the generation of Oh *et al.* [125] (dark blue), our model without $\mathcal{L}_{\text{dis}}$ and without $\mathcal{L}_{\text{mag}}$ (light blue), our model without $\mathcal{L}_{\text{mag}}$ (purple) and our final model (orange). In contrast to our final model, the generation quality of Oh *et al.* and our incomplete models decreases significantly (FID increase) with increasing λ.

the magnified images with in total 25 different magnification factors (λ), where $\lambda \in [1.2, 6]$ with a step size of 0.2, and train one classifier per λ and posture.

The optimal λ per posture has been found using cross-validation and Tab. 1 reports the accuracy on the test set. We do not expect the accuracy to be $\sim 100\%$ since not all impaired patients have issues with each posture, i.e., for specific posture-subject pairs no differences to healthy subjects should be detected. This behavior can be also observed in Figure 5.12. The patient (2nd row) only shows difficulties during the right step. Our model detects the deviation to a healthy subject (1st row) and only magnifies the inaccurate posture during the right step.

We also compare our quantitative results with previous work on motion magnification. We performed the same experiment explained above using the magnified generations of Oh *et al.* [125] and report the accuracies in Table 5.1.

Most of the classifiers trained on the original data stay close to random performance and are not able to distinguish healthy from impaired. Compared to the approach of [125], our magnified images can increase the accuracy significantly. We show especially for posture 4,5 and 8 a large boost and improve the classification accuracy in average by more than 11%. This experiment demonstrates that our model is a valuable tool for discovering impairment of motor behavior.

Quality of Visualizations - FID. The Fréchet Inception Distance (FID) was originally proposed by Heusel *et al.* [68] and aims to evaluate the quality of generated images. It measures the distance between two multivariate Gaussians (real and generated images):

$$\text{FID} = ||\mu_r - \mu_g||_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}), \tag{5.16}$$

where $X_r \sim \mathcal{N}(\mu_r, \Sigma_r)$ and $X_g \sim \mathcal{N}(\mu_g, \Sigma_g)$ are the activations of the Inception-v3 layer for real and generated images, respectively. We empirically found, that the features of the Inception network (pre-trained on ImageNet) are less susceptible to posture differences than to appearance perturbations. Different postures reach on average an FID of around 5. This shows that the posture change through magnification is negligible when measuring FID. Therefore, it allows us to evaluate the generation quality of our magnified images using different magnification

intensities without requiring ground-truth magnifications. In Figure 5.13 we show the absolute FID increase relative to $\lambda = 1$. This experiment demonstrates that with increasing λ our final model achieves the best results and only forfeits a small decrease in quality even with higher λ. Please note that this experiment evaluates the generation quality, not if the magnifications correspond to the actual amplification of the posture deviations.

5.3.4 Ablation Studies

In Figure 5.7 and 5.13 we evaluate the importance of our proposed losses. We compare the magnified images produced by our full model with the generations of our model without $\mathcal{L}_{\text{mag}}$ and/or without $\mathcal{L}_{\text{dis}}$. Figure 5.7 shows that our model without $\mathcal{L}_{\text{dis}}$ and $\mathcal{L}_{\text{mag}}$ generates, similar to [125], blurry and unrealistically looking images. Our model trained with the disentanglement loss $\mathcal{L}_{\text{dis}}$ improves the generations especially for smaller λ, but fails in producing valuable magnifications for larger λ. Instead, our final model is able to precisely display the magnification of posture deviations on the generated images even for large λ. Similar conclusions can be drawn from Figure 5.13. The quality of the magnified images decreases for our incomplete models for $\lambda > 3$ but stays almost constant for our final model. This shows that every component of our robust model is important.

5.4 Technical Details

We use a network architecture similar to the model proposed by Esser *et al.* [49] with 6 Resnet blocks [66] per encoder/decoder and skip connections between the appearance encoder and the decoder. For downsampling we use a convolutional layer (kernelsize=3, stride=2, padding=1) and for upsampling we employ NN upsampling followed by a 1x1 convolutional layer. Our discriminator C has the same structure as DCGAN[136]. The weights are Xavier initialized and we train our model using the Adam optimizer [86] with a learning rate of 1×10^{-4}. We use a batchsize of 10 and an image size of 128x128.

The standard deviation σ for the Gaussian distribution used in $\mathcal{L}_{\text{dis}}$ is set to 0.01. We did not see that as a critical parameter; slight changes did not affect the results.

To train our model with $\mathcal{L}_{\text{mag}}$ we require the nearest neighbors of every query sample x^q. Therefore, we first train our model without $\mathcal{L}_{\text{mag}}$ and then employ the trained posture encoding for obtaining the NNs. Afterwards, we train our model with all losses. We train in total for 300 Epochs for Golf Swing and 100 Epochs for HG2DB and CUEye.

5.5 Discussion

To enable a meaningful magnification of posture deviations in less restricted recording setups we have proposed a robust magnification strategy with two novel

losses. We obtain a stronger disentanglement of posture and appearance by forcing the generative model to reconstruct an image accurately only if both encoders contribute information to the generation process. In addition, our fixpoint loss enables us to integrate the magnification directly into the training and to learn on real data without supervision. Our approach is evaluated on three different scenarios including the very challenging golf swing dataset with varying camera setups and locations. We have demonstrated the importance of both novel objectives in the experimental section and compare favorably against the latest motion magnification approach.

Chapter 6

Semi-Supervised Representation Learning for Videos

Learning a feature representation without requiring any labels has shown to be an effective tool whenever annotations are difficult to obtain. In contrast, supervised learning requires fully labeled datasets, but shows superior performances on common visual understanding tasks. In this chapter, we aim at combining the best of both worlds: an approach that shows superior performances on visual understanding tasks while keeping the amount of human supervision at a minimum. In particular, we investigate the effect of semi-supervised learning on action recognition. Semi-supervised learning (SSL) efficiently combines labeled and unlabeled data to train powerful feature representations without being dependent on extensive annotated datasets. Previous works on SSL for action recognition have mostly established hand-crafted features or have been applied to only very restricted settings. Following the success of SSL for images using deep learning, we introduce a deep semi-supervised learning approach for action recognition that efficiently deduces pseudo-labels for the unlabeled samples. We further propose to combine the semi-supervised method with an unsupervised pre-training to create a better foundation before training on the desired downstream task with only few labels. Experimental evaluations for action recognition demonstrate the competitive performance on two benchmark datasets with up to 400 classes.

6.1 Introduction

Unsupervised learning enables to produce powerful feature representations for several image and video understanding tasks without requiring any manual annotations. The state-of-the-art in unsupervised learning is improving constantly, but the gap to supervised models is especially for videos still significant. In the semi-supervised setting, we assume that a small set of samples is labeled. The goal is to improve the performance on datasets with only few annotated samples by leveraging a large number of unlabeled samples. Thus, semi-supervised learning (SSL) offers a good trade-off between labeling time/expenses and performance. In particular recent SSL approaches for images [165, 24, 92, 97, 138, 87, 191, 117, 189] have shown that semi-supervised learning is a promising direction for solving

challenging visual understanding tasks. We observe that there has been unused potential in deep SSL for action recognition. Previous works have mostly developed hand-crafted features [195, 144, 135, 192] for very restricted settings. For instance, [195] adapts knowledge from the image domain to the video domain but evaluates their approach only on few action classes chosen by hand.

In this chapter, we present a deep SSL approach for action recognition that has been successfully applied to two benchmark datasets with up to 400 different activities. Given the recent success in unsupervised video representation learning, our method contains a self-supervised pre-training phase to create a better foundation before training with the few labeled samples. Moreover, we propose a novel semi-supervised approach that efficiently employs the labeled videos for inferring pseudo-labels of the remaining samples. Specifically, our model contains 3 major phases: *(1)* An unsupervised pre-training where we employ an unlabeled dataset that is larger than our target dataset and contains related content; *(2)* a fine-tuning phase where we use the few labeled samples to train our model on the desired downstream task; and *(3)* a post-training, that infers pseudo-labels for the unlabeled samples. Phase 2 and 3 are initialized with the models trained in phase 1 and 2, respectively.

Besides evaluating our model on two benchmark datasets, we additionally perform an extensive ablation study in order to separately analyze the performance achieved for every single phase.

The remaining of this chapter is structured as follows: At first, we will summarize in Section 6.2 previous work on semi-supervised and unsupervised learning. Then, we will describe our approach in section 6.3 and 6.4 and evaluate the performance gain in section 6.5. Finally, we will provide technical details, summarize the findings and discuss the limitations of our presented approach.

6.2 Related Works

In this section we introduce previous work on semi-supervised learning for videos and images and provide an overview of recent unsupervised learning techniques for videos.

SSL for Videos. Semi-supervised learning (SSL) provides a good trade-off between labeling time and accuracy. Surprisingly, there has been only little work on deep SSL for action recognition [3, 76, 64, 106] using visual information (videos or images) as input. Ahsan *et al.* [3] attempt to tackle SSL for videos through deep learning by first performing an unsupervised pre-training using generative adversarial networks followed by fine-tuning on action recognition using the full training set. However, this approach rather corresponds to the common unsupervised learning scenario; in SSL, only a fraction of the labeled samples is employed instead of the full dataset. Other works do not necessarily use RGB images as input, but alternative modalities such as RGB-D [76, 106] or accelerometers data [64, 105].

SSL for Images. In contrast to SSL for videos, SSL for images has recently gained increasing attraction. Recent approaches [165, 24, 92, 97, 138, 87, 191, 117, 189] significantly improve the performance upon baseline models. Our model is particularly inspired by deep SSL methods for image classification.

The approaches can be divided into two major categories: *(i)* inferring pseudo-labels for the unlabeled samples [97, 74, 117] and *(ii)* adding an unsupervised objective to the loss function [165, 92, 24, 191, 194]. Iscen *et al.* [74] employ a transductive label propagation method to infer pseudo-labels for the unlabeled samples. Their method iterates between updating the pseudo-labels with a nearest neighbor graph and training the network with the newly inferred labels. We also iterate in phase 3 between updating the network weights and re-computing the pseudo-labels of unlabeled videos. However, our model does not require the computation of nearest neighbors for every sample given all other samples in the dataset; we infer pseudo-labels from a more efficient clustering approach. Phi Vu Tran [165] has recently proposed an approach that falls under the second category (additional unsupervised objective). The model is simultaneously trained on a supervised and self-supervised task to enable an efficient use of the whole dataset. For the latter, [165] employs the surrogate task of predicting image rotations [60] using all images while the former only regards the few labeled samples. However, a multi-task network requires a careful balancing [180, 38] to exploit the full potential of the individual tasks. We also employ an unsupervised approach during the pre-training phase, but we do not train our model on two tasks at the same time. Our experiments have shown, that a sequential training achieves better results than training the tasks in parallel.

Unsupervised Video Representation Learning. Previous work in unsupervised learning of video representations capitalize on self-supervised methods [187, 98, 116, 15] or generative models [170, 34, 104, 3]. The latter aims to generate high-fidelity video clips [170, 34, 3] or predict long-term motion [104] while learning useful features for recognizing actions. In self-supervision, on the other hand, the representation is learned by solving a surrogate task that exploits the spatiotemporal signal in videos as source of information. Similar to our approach proposed in Chapter 3, Hsin-Ying *et al.* [98] have introduced an Order Prediction Network (OPN) architecture that learns a video representation by sorting sequences. A 2D convolutional neural network (CNN) receives temporally shuffled frames as input and is trained on the surrogate task of classifying the employed permutation. Xu *et al.* [187] have extended this work by proposing Clip Order Prediction (COP), which uses a 3D network architecture and shuffles sequence snippets instead of single frames. COP is the current state-of-the-art in unsupervised learning of video representations. Therefore, we have considered COP for phase 1 (unsupervised pre-training) of our proposed approach.

6.3 Unsupervised Pre-Training and Fine-tuning

As demonstrated in many approaches for unsupervised learning of video representations [187, 98, 116, 15, 170, 34, 104, 3], a pre-training, that does not require any labels can significantly boost the performance for action classification. Instead of initializing the network with random weights before training with labeled data, the model is initialized with the weights obtained during pre-training. Given the substantial success in unsupervised learning, we propose to adopt this procedure for SSL of videos.

Phase 1: Unsupervised Pre-Training. COP [187] significantly outper-

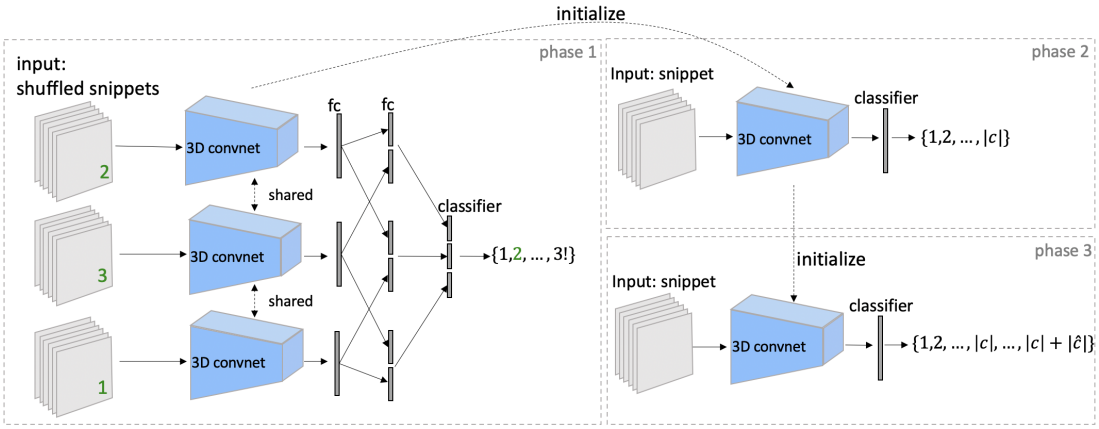


Figure 6.1: Overview of the network architectures for all phases. In phase 1 (left) we train the network with clip order prediction [187]: 3 snippets are extracted from a video and temporally permuted (from the original order $(1,2,3)$ to $(2,3,1)$). After a pairwise concatenation of the 3D convnet features, a classifier predicts the class of permutation with $3!$ possible permutations/classes. In phase 2 (top-right) we initialize the 3D ConvNet with the weights from phase 1 and fine-tune the network with the labeled samples on the downstream task with $|c|$ classes. In the last phase (bottom-right) we employ the ground-truth labels and inferred pseudo-labels to post-train the 3D ConvNet with $|c| + |\hat{c}|$ classes.

forms previous work on unsupervised video representation learning. A neural network is trained by first sampling m non-overlapping snippets $s_1, \ldots, s_m$ (the subscripts indicate the chronological order) from a video x where every snippet s_j $(j = 1, \ldots, m)$ consists of n frames. Then, after permuting the snippets in a random order, a 3D ConvNet $f(\bullet;\theta)$ with θ the network weights, is used to extract the features $\phi_{s_j} = f(s_j;\theta)$ for every snippet s_j. After a pairwise concatenation of the features, fully-connected layers placed on top of the 3D ConvNet predict the previously applied permutation, *i.e.* the order prediction is formulated as a classification problem with $m!$ classes (see Figure 6.1 (left) for a visual description of the network architecture). Thus, the network is updated based on the ability of predicting the applied permutation. Solving this task requires an understanding of the temporal structure of actions and is therefore a valuable network initialization for the downstream task of action classification. COP is currently the state-of-the-art in unsupervised video representation learning. Therefore, we have employed COP for phase 1 of our approach. We refer the interested reader to [187] for a more detailed description.

As commonly known, deep learning benefits from large volumes of training data. Fortunately, visual data is cheap to acquire if no labelling process is needed and unsupervised learning does not require any manual annotations. Therefore, instead of pre-training on the target dataset X, we propose to employ a larger and unlabeled dataset $\mathcal{X}$ for pre-training, where $\mathcal{X}$ and X are assumed to contain related content. We show in the experimental section that the assumption of achieving an increased accuracy when using a larger dataset for pre-training is especially true for small datasets.

Phase 2: Fine-tuning. After pre-training our model on the unsupervised

task, we fine-tune the network on the downstream task using the subset $X_L \subset X$ of labeled videos with $y_{x_l} \in \{1, 2, ..., |c|\}$ the label of $x_l \in X_L$ and $|c|$ the number of ground-truth classes. We will show in the experimental section that the pre-training, especially for small datasets, significantly boosts the performance on the downstream task in comparison to a baseline. However, so far, we have only employed the full dataset during the unsupervised learning phase. Previous work on SSL for images has shown a significant improvement in accuracy when inferring pseudo-labels [97, 74, 117] for the unlabeled subset X_U in order to train with the full dataset $X = X_L \cap X_U$ on the downstream task. Therefore, we propose in the next section a novel semi-supervised approach for videos that determines pseudo-labels for unlabeled videos given the network weights θ_L obtained during phase 2.

6.4 Post-Training via Pseudo-Labelling

The aim of SSL is to efficiently employ the few labeled samples for boosting the performance on the desired downstream task. Now, we present an iterative technique that post-trains the fine-tuned network with inferred pseudo-labels.

Inferring Pseudo-Labels by Nearest Neighbors. Our SSL approach for videos is based on the assumption, that similar samples (e.g. videos showing the same movement) should obtain the same prediction. Let Φ_{x^c} be the representation of an arbitrary video $x^c \in X$ with c the true class of x^c. Then we assume that the distance $d(\Phi_\bullet, \Phi_{x_u^c})$ between $x_u^c \in X_U$ and every $x_l^c \in X_L$ is on average smaller than to every $x_l^{\bar{c}} \in X_L$ where $\bar{c} \neq c$. In other words, we assume that an unlabeled sample x_u^c is on average closer to a labeled sample x_l^c from the same class c than to a labeled sample $x_l^{\bar{c}}$ from a different class $\bar{c}$ given the feature representation $\Phi_\bullet$. Given this assumption, we can infer the pseudo-label $y_{x_u^c}$ by first calculating the distance d between $\Phi_{x_u^c}$ and Φ_{x_l} for all $x_l \in X_L$ and then setting $y_{x_u^c} = y_{\hat{x}_l}$ where $\hat{x}_l \in X_L$ is the closest (labeled) sample to x_u^c (nearest neighbor) in the feature space, *i.e.*

$$\hat{x}_l = \text{argmin}_{x_l \in X_L} d(\Phi_{x_u^c}, \Phi_{x_l}). \tag{6.1}$$

With a precise video representation, the pseudo-label $y_{x_u^c}$ equals the true class c.

Comparing every unlabeled sample in X_U to every labeled sample in X_L is, however, even for small datasets computationally inefficient and unstable if high intra-class variances exist. Therefore, we assign the pseudo-label $y_{x_u^c}$ not based on the distance to individual samples but to a set of samples that are grouped with respect to their class affiliations. In particular, we first compute the feature representation Φ^{c_i} for every class c_i with $(i = 1, \ldots, |c|)$ by averaging the features of all samples belonging to class c_i

$$\Phi^{c_i} = \frac{1}{n_{c_i}} \sum_{x \in X_L^{c_i}} \Phi_x \tag{6.2}$$

with $X_L^{c_i}$ being the set of labeled samples from c_i and n_{c_i} its size. Then, we set $y_{x_u^c} = c_i$ if $d(\Phi_{x_u^c}, \Phi^{c_i}) < d(\Phi_{x_u^c}, \Phi^{c_k})\ \forall k \in \{1, \ldots, |c|\}/i$. In this way,

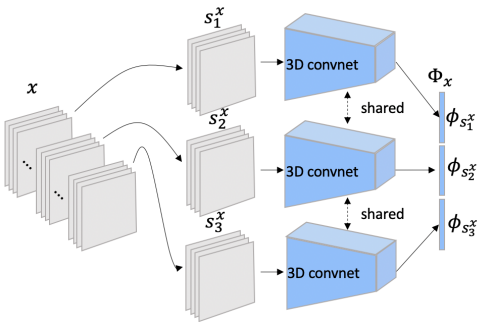


Figure 6.2: Depiction of the video representation for $\nu = 3$. Per video x, we sample $\nu = 3$ different snippets, extract their features and concatenate them to obtain the feature representation Φ_x.

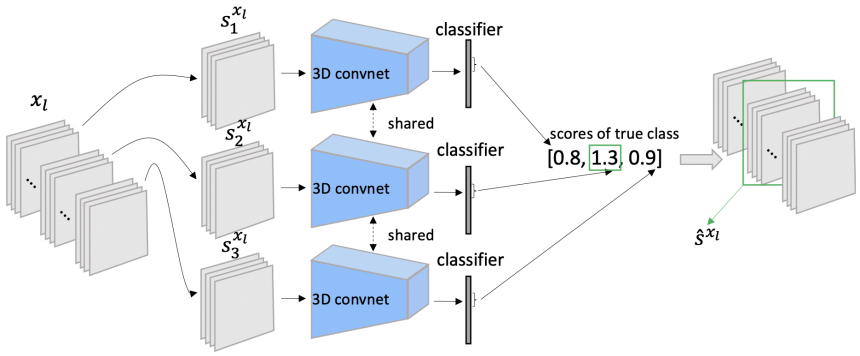


Figure 6.3: Alternative way for representing a labeled video x_l by selecting a snippet $\hat{s}^{x_l}$ from all extracted snippets s_j using the logits of the true class.

we significantly reduce the complexity for obtaining the pseudo-labels based on similarities since $|c| \ll |X_L|$.

Video Representation. For the assumption, that $\Phi_{x_u^c}$ is closest to the average feature representation of its true class c to hold, we require strong video features that accurately represent the full action. As commonly pursued in video understanding tasks [110, 187, 16], we employ video snippets to represent x, where every snippet consists of n frames. In particular, we employ in total ν non-overlapping and linearly-spaced snippets s_j per video with $j = 1, \dots \nu$ in order to capture the full action required for accurately measuring the similarity between videos. Then, we use the 3D ConvNet trained on the downstream task (in phase 2) to obtain the features $\phi_{s_j} = f(s_j; \theta_L)$ for every snippet s_j. Finally, we concatenate the features of the individual snippets to retrieve the video representation of an arbitrary video x,

$$\Phi_x = (\phi_{s_1^x}, \phi_{s_2^x}, \dots, \phi_{s_\nu^x}). \tag{6.3}$$

Figure 6.2 visually depicts this procedure.

Unfortunately, the input videos are often not perfectly cropped, *i.e.* there might be sequences in a video that contain a component or movement that does not belong to the labeled activity. Therefore, as an alternative, we have also experimented with selecting one of the snippets s_j^x as representative instead of concatenating the ν snippets. Given the network that has been trained in phase

2, we perform a forward pass for every snippet $s_j^{x_l}$ of a labeled video x_l and extract the score of the true class of x_l. We then select the snippet with the maximum score ($\hat{s}^{x_l}$) to represent x_l (see Figure 6.3). Since the true class of an unlabeled sample x_u is not known, we cannot employ the same procedure. Therefore, we propose an alternative approach for selecting a snippet for x_u that is based on the labeled videos. At first, we compute the average feature representation Φ^{c_i} per true class $c_i, (i = 1, \ldots, |c|)$ using the features of the labeled samples similar to Eq. 6.2. Though, instead of using the concatenated snippets, we now use the features of the previously selected snippets. Then, per unlabeled video x_u, we compute the distance between its snippets and all Φ^{c_i}. The snippet closest to one of the average feature representations Φ^{c_i} is then selected as representative of x_u.

Choosing only one snippet that supposedly best represents the presented action might remove parts of the video that are irrelevant for classifying the movement. However, this approach also entails the risk of eliminating important factors of the action, especially if the activity is protracted. Given the videos of the benchmark datasets we evaluate our approach on, we have empirically found that concatenating several snippets leads to slightly better results than choosing one of them. Therefore, our experiments in Section 6.5 are based on the video representation that employs all ν snippets.

Inferring Pseudo-Labels by Clustering. To obtain the highest possible gain from training the network with inferred pseudo-labels, we require strong features to find the true label of unlabeled samples through nearest neighbors. Therefore, we employ the network weights θ_L obtained during the fine-tuning phase. However, networks that are only trained with a small part of the dataset might not generalize well to unseen data, especially for classes with high intra-class variances. In this situation, the nearest neighbor of an unlabeled sample does not necessarily correspond to a video from the same class, but most likely to an outlier in the labeled set X_L. Thus, inspired by DeepCluster [17], we propose to cluster the unlabeled samples separately from the labeled videos and jointly train the network with $|c| + |\hat{c}|$ output classes, where $|\hat{c}|$ is the number of clusters for the unlabeled samples. In this way, we can explore new structures in the dataset using the unlabeled samples while maintaining the structure we have already established from the labeled samples. In fact, DeepCluster [17] has shown, that the highest performance gain in unsupervised representation learning for images is achieved when using 10 times more clusters than the actual number of ground-truth classes. This demonstrates, that it can be beneficial to over-cluster the data for learning the structure of the full dataset.

For clustering the unlabeled samples, we first extract the features Φ_x (Eq. 6.3) for all $x \in X_U$. Then, we apply k-means to cluster the unlabeled videos in $|\hat{c}|$ groups. The pseudo-label of an unlabeled samples x_u is based on the cluster label and the total number of ground-truth classes $|c|$, *i.e.*

$$y_{x_u} \in \{|c| + i\}_{i=1}^{|\hat{c}|}. \tag{6.4}$$

If an unlabeled video is closer to the centroid of a class that is represented by labeled samples, we employ the label of that class as pseudo-label instead of the cluster label.

	#Videos	#Per Class
100%	9537	71-221
20%	1818	18
10%	909	9
5%	404	4

Table 6.1: Number of available training videos for UCF101 in the semi-supervised setting (partly labeled).

	#Videos	#Per Class
100%	240,577	224-990
5%	11,600	29
2%	4,4000	11
1%	2,000	5

Table 6.2: Number of available training videos for Kinetics400 in the semi-supervised setting (partly labeled).

Training. Given the ground-truth labels of the labeled samples and the pseudo-labels determined in the previous step, we can now train our model on the whole training set. We alternate between updating the network weights and recomputing the pseudo-labels as proposed in [17]. Thus, the pseudo-labels always represent the current status of the network. We do not employ the full training set from the beginning, but slowly increase the number of unlabeled samples in the training set based on the distance to their centroids. Consequently, we also slowly increase the number of clusters relative to the growth of the training set.

6.5 Experiments

In this section we will first introduce the datasets and metrics employed for evaluating our approach. Then, we present the performance gain we achieve with our model in comparison to a baseline method. Finally, we perform ablation studies to evaluate every component of our approach.

6.5.1 Datasets

We evaluate our approach on two benchmark datasets for action recognition. We are not aware of any previous works, that evaluate an SSL approach for videos on the selected benchmark datasets. Therefore, we have employed our own sampled subsets for the experiments. We have sampled 3 subsets from a particular dataset with different numbers of videos to simulate the labeled set X_L. Per subset and class, we randomly select a pre-defined number of videos. Table 6.1 and Table 6.2 provide an overview of the selected subsets for our experiments.

UCF101. UCF101[150] has been already employed in previous chapters for evaluating the features of our unsupervised video representation learning approaches. It contains $\sim$ 13k YouTube videos divided in 101 different human action classes. $\sim$ 10k videos are provided for training and $\sim$ 3k for testing. Table 6.1 summarizes the number of labeled videos we employ for our experiments. We train our model with either 20%, 10% or 5% of the training data of split1 and evaluate on the test set. As mentioned in Section 6.3, we train the unsupervised task on a larger dataset $\mathcal{X}$ than the target dataset (UCF101). In particular, we employ Kinetics400 for UCF101 as the larger dataset since both contain related content (human activities).

	Accuracy (%)				Precision@1 (%)			
	5%	10%	20%	100%*	5%	10%	20%	100%*
BL	20.06	31.49	48.18	65.76	18.42	27.88	41.95	60.40
Ours	32.96	48.74	61.84	74.25	31.54	46.23	56.62	68.74

Table 6.3: Accuracy and Precision@1 on the test set of **UCF101** using our full model vs. Baseline (BL). *When using 100% of the labels, we omit the semi-supervised post-step and the results in *Ours* present the performances achieved after pre-training on Kinetics400 (phase 1) and fine-tuning on UCF101 (phase 2).

Kinetics400. As UCF101, Kinetics400 [82] is also a collection of YouTube videos showing various people performing in total 400 different actions. The dataset consists of $\sim 260k$ videos with $\sim 240k$ samples for training and $\sim 20k$ for testing. Table 6.2 summarizes the number of labels we employ per subset. We experiment with 1%, 2% and 5% of the available labels and report the results on the full testing set. We use Kinetics700 [19] for the unsupervised pre-training.

6.5.2 Evaluation Metrics

We evaluate our model using two different metrics: **accuracy** and **precision@1**. For the accuracy, we simply measure the prediction accuracy on the evaluation set of UCF101 or Kinetics400. Per video, we extract three snippets, with $n = 16$ frames per snippet, from the beginning, center and end of the video and average the logits over all snippets. The accuracy reports the relative amount of correct predictions for all videos in the test set. During the post-training phase, we might train with more classes than the actual amount of ground-truth classes $|c|$, *i.e.* the FC layer contains more than $|c|$ output nodes. Since our approach ensures that the first $|c|$ entries of the FC layer correspond to the actual ground-truth classes, we truncate the FC layer at $|c|$ to compute the accuracy.

The precision@1 is calculated using nearest neighbor retrieval. This evaluation measure does not require a trained FC classifier. For every sample in the evaluation set we calculate the nearest neighbor in the training subset using the features of the trained model (last layer before the FC layers). A sample is correctly predicted if the nearest neighbor has the same class. The average over all samples represents the precision@1. This evaluation procedure considers the actual feature representation and disregards the task-specific layers (e.g. the last FC layer for predicting the ground-truth class during fine-tuning). In this way, we guarantee a fair comparison of all models no matter with which task it has been trained.

6.5.3 Quantitative Evaluation

In the following, we will first describe the baseline model we compare with. Then, we will report the results on the test sets of UCF101 and Kinetics400.

Baseline. For the baseline model (BL), we only fine-tune the network on the labeled subset X_L without any unsupervised pre-training or pseudo-labelling, *i.e.* we randomly initialize the network before conducting phase 2.

	Accuracy (%)			Precision@1 (%)		
	1%	2%	5%	1%	2%	5%
BL	5.86	10.88	19.39	4.21	7.67	12.5
Ours	6.97	12.98	23.28	5.33	11.37	16.81

Table 6.4: Accuracy and precision@1 on the test set of **Kinetics400** using our full model vs. Baseline (BL).

Models	5%	10%	20%	100%
Phase 2 (BL)	18.42	27.88	41.95	60.40
Phase 1 & 2 (target)*	24.95	40.08	52.04	62.95
Phase 1 & 2	29.57	44.22	54.84	**68.38**
Phase 2 & 3	25.17	37.2	46.03	-
Full model (target)*	28.84	43.63	52.89	-
Full model	**31.54**	**46.23**	**56.62**	-

Table 6.5: Precision@1 on UCF101 obtained with different combinations of the 3 phases. *Instead of pre-training with the larger dataset in phase 1, we employ the target dataset (UCF101).

Results. Table 6.3 and 6.4 show the results achieved with our approach and the baseline model for UCF101 and Kinetics400, respectively. For UCF101, our full model significantly outperforms the baseline model. In fact, our model only requires half of the data to achieve the same or higher accuracy as the baseline model: with 5% of the data we obtain a higher performance than the baseline model that is trained with 10% of the samples; the same applies to 10%. Moreover, when using 20% of the samples our model almost reaches the precision@1 of the baseline model trained with 100%. Kinetics400 is with 400 classes significantly more difficult than UCF101. Nevertheless, our approach shows especially for 5% a significant boost in performance, proving that our method is applicable for middle- and large-scale datasets.

6.5.4 Ablation Studies

We have shown a significant performance gain when training the network with our model. Now we perform ablation studies to separately analyze the gain achieved with the unsupervised pre-training (phase 1) or pseudo-labelling based post-training (phase 3) on UCF101. In particular, we show in Table 6.5 the precision@1 obtained with *(i)* phase 1 and 2, *(ii)* phase 2 and 3 and *(iii)* phase 1, 2 and 3 (*Full model*). We also report the results for using the target dataset during phase 1 as opposed to employing the larger dataset. It shows, that especially the pre-training with the larger dataset (*phase 1 and 2*) and also the semi-supervised approach significantly boosts the performance for UCF101 in comparison to the baseline or the results obtained when pre-training with UCF101 (*phase 1 and 2 (target)*). It proves that all the components of our model are important to achieve the final performance.

k	SelfSL		SSL			SelfSL (K400)+SSL		
	UCF	K400	5%	10%	20%	5%	10%	20%
1	7.31	9.59	28.23	36.42	45.54	33.86	44.78	55.13
5	21.02	22.27	41.07	50.10	59.04	47.62	58.01	66.41
10	31.0	31.01	47.16	55.33	64.25	53.75	63.26	70.53
20	43.22	42.71	53.89	60.72	69.52	59.69	67.77	74.56
50	61.66	60.11	63.33	67.37	76.67	66.80	74.76	79.46

Table 6.6: Comparing self-supervised with semi-supervised learning using nearest neighbor retrieval on UCF-101. Test set videos are used as queries and videos from the training set as retrieval targets. If the true class of a test sample appears within the Top$k$ it is considered correctly predicted. As distance measure for retrieving nearest neighbors we employ cosine distance. We report mean accuracies (%) over all test samples using the representations trained on *(i)* only phase 1 (SelfSL) *(2)* phase 2 and 3 (SSL) and *(3)* phase 1, 2 and 3 (SelfSL (K400)+SSL).

6.6 Self-Supervised vs. Semi-Supervised

The previous section has shown that using only a small fraction of labeled samples and combining them with a large unlabeled dataset leads to significant improvements in comparison to training only with the labeled samples (baseline). In this chapter, we investigate if the effort for labelling few samples pays off in terms of performance in contrast to self-supervised learning. Thus, we directly compare the results of our semi-supervised method (which requires a small amount of human supervision) with the accuracy obtained by using solely unlabeled samples. In particular, we compare the representation achieved after phase 1 (only self-supervised task) with the features obtained after training phase 2 and 3 (fine-tuning and post-training) using nearest neighbor retrieval on UCF-101. For every sample in the evaluation set we calculate the k nearest neighbors in the training set using the features of the trained models (last layer before the FC layers) and cosine distance. A test sample is considered as correctly predicted if its class can be found within the Top$k$ nearest neighbors. The final accuracy is determined by computing the mean over all testing samples.

At first, we compare the performance achieved when using K400 instead of UCF-101 for training the network on the self-supervised approach (1st column in Table 6.6). Surprisingly, using the larger unlabeled dataset (Kinetics400) does not lead to a strong improvement in comparison to using UCF-101 during phase 1. The potential of applying a large dataset during pre-training seems to emerge only after fine-tuning on the downstream task (Phase 1 & 2 (target) vs Phase 1 & 2 in Table 6.5).

Next, we compare SelfSL with SSL (column 1 and 2 in Table 6.6). Using only 5% of the data (4 videos per class) leads already to a significant improvement of almost 300% for $k = 1$. Only for $k > 20$, SelfSL and SSL with 5% of labeled samples achieve similar results. It shows that spending a limited amount of time on annotating few labels (if possible) can lead to a superior improvement in performance. However, this assumption cannot be transferred blindly to arbitrary problems or datasets and should, therefore, be always reassessed in case of a new

task or dataset.

Finally, we additionally provide the nearest neighbor retrieval results for $k = 1, 5, 10, 20, 50$ for our final model which combines self-supervised pre-training with semi-supervised learning. The results in Table 6.6 (last column) confirm the conclusion, already drawn in the previous chapter, that a combination of unsupervised and semi-supervised learning methods yields the highest accuracies.

6.7 Technical Details

All deep networks in our experiments are implemented using the PyTorch[1] framework. We use C3D [162] as our backbone network and employ 2 FC layers as in [187] for phase 1 and 1 FC layer for phase 2 and 3 on top of the last conv layer of C3D (see Figure 6.1). During training, we sample 3 snippets per video with a length of 16 frames each and an interval of 8 frames between them. One mini-batch contains 12 videos, *i.e.* 3×12 snippets. For data augmentation we follow the same protocol as in [187]: during training, the snippets are first resized to 128×172, then randomly cropped to 112×112. For testing, the snippets are cropped in the center. We use mini-batch stochastic gradient descent to optimize the network with a starting learning rate of $1e-3$, a momentum of 0.9 and weight decay of 0.0005. The learning rate is reduced by 0.1 whenever a plateau is reached until a minimum learning rate of $1e-5$. For all phases, we train the network until the model has converged but at most 400 Epochs.

We perform clustering on the GPU using the faiss [79] library. For UCF101 we update the pseudo-labels every 5 Epochs and for Kinetics400 every 15 Epochs. We use 101 clusters for UCF-101, meaning that the highest accuracies for UCF-101 are reached when using nearest neighbors for inferring pseudo-labels instead of clustering. However, action recognition on Kinetics400 benefits more from clustering and the best results were achieved using 800 clusters.

6.8 Discussion

In this chapter we have proposed a novel semi-supervised representation learning approach for action recognition. Our model consists of 3 phases, namely *(1)* a self-supervised pre-training, *(2)* a fine-tuning and *(3)* a pseudo-labelling based post-training. We have evaluated our approach on two benchmark datasets including the large-scale data collection Kinetics400 [82]. Our semi-supervised method significantly improves upon the baseline for both datasets. We have shown that phase *(1)* and *(3)* are essential to achieve the final performances. Interestingly, our model achieves the best accuracy for the smaller dataset UCF-101 when using the nearest neighbors for inferring pseudo-labels instead of clustering. In contrast, for Kinetics400 we achieve the best results when using twice as much clusters as classes. This behavior indicates that over-clustering might only be beneficial for large-scale datasets such as Kinetics400 or ImageNet (used in DeepCluster [17]). For future work it would be interesting to further investigate this demeanor especially in combination with video understanding tasks.

[1] `http://pytorch.org/`

Chapter 7

Conclusion and Discussion

This thesis focused on developing deep learning approaches that require only minimal supervision to solve image and especially video understanding tasks. New methods were proposed to learn powerful representations of images and videos without requiring manual annotations. The learned representations enable not only an objective classification of actions and behavior but also allow to compare and quantify even small differences across individuals. To further increase the performance on visual understanding tasks without requiring extensive human supervision, this thesis additionally proposes a semi-supervised approach that efficiently combines large unlabeled datasets with a small set of labeled samples. The wide applicability of the presented methods was demonstrated on several visual understanding tasks including a biomedical application. The remainder of this chapter separately summarizes the different approaches and their applications and concludes with a discussion. The discussion at the end of this chapter contains personal opinions which are based on experiences and knowledge collected throughout this Ph.D.

Deep learning benefits from large volumes of training data. Fortunately, self-supervised learning allows to exploit a huge amount of data freely available on the internet without requiring tedious, time-intensive, manual annotations. Even though self-supervised learning has experienced a substantial amount of progress recently, the performance gap to supervised methods is still significant. Thus, a large part of this work was committed to developing novel self-supervised learning methods for advancing the research field. The approach introduced in Section 3.3.1 exploits temporal information from videos in order to construct an artificial supervisory signal. In particular, a neural network receives randomly permuted video frames as input and is tasked to recognize the applied permutation. This task can only be solved if the network understands the content of the video. The proposed method outperformed previous image and video-based self-supervised learning approaches on human pose estimation and action recognition. In order to exploit even more data which is freely available, Section 3.4.1 introduced a permutation framework that is able to process not only videos, but also images at the same time. A multi-task neural network is simultaneously trained on permuted video frames (temporal shuffling) and permuted image tiles (spatial shuffling). Addressing the two directly related ordering tasks jointly enables the extraction of information from images and videos simultaneously. The resulting

feature representation was evaluated on several image and video understanding tasks and showed superior performances compared to previous works.

Besides advancing the computer vision field by introducing new self-supervised learning approaches, this thesis additionally aimed at improving the workflow in biomedical research by automatizing tedious manual (and partly repetitive) processes. In particular, this dissertation includes a collaboration with neuroscientists which are evaluating new treatments for neurologically impaired individuals by analyzing the change in motor function before and after treatment. Previous machine-learning-based works were fully supervised and required experts from neuroscience to manually annotate virtual key-points in behavior videos. In order to facilitate the development of suitable treatments, Chapter 4 presented a diagnostic support system that solely requires the recordings as input without demanding any tedious annotations. The proposed system applies the temporal self-supervised approach, introduced in the previous Chapter, to learn a fine-grained posture and behavior representation. Moreover, the magnification tool described in Section 4.5 enables the diagnostic support system to discover and quantify even small deviations in posture between healthy and impaired individuals. A generative neural network is trained on disentangling posture and appearance for image synthesis to ensure that the posture deviations across individuals can be analyzed separately despite appearance differences. Then, during inference, the generative model can be used to magnify the subtle differences in posture between healthy and impaired individuals using only the recorded videos as input. The proposed diagnostic support system was evaluated on two different medical scenarios with rodents and humans and showed superior performances compared to previous works.

Magnifying subtle posture deviations across individuals is not only valuable for biomedical research projects. In sports, for instance, analyzing the execution of a specific action to discover subtle mistakes and to adjust the movement accordingly is crucial to increase the overall performance. A magnification tool can support such an analysis by facilitating the perception of those small mistakes using the movement of a professional athlete as reference. The sports scenario is, however, less restricted than a strongly controlled experiment with a static background in a medical lab. Videos from athletes might be recorded outside or at many different locations. Thus, Chapter 5 introduced a more robust magnification approach that can be applied to less restricted scenarios. A novel disentanglement loss guarantees a stronger separation of posture and appearance which is especially important if the videos are recorded at different locations. Furthermore, this thesis introduced a magnification loss which enables neural networks to be trained directly on inferred magnifications of real data. In this way, the generative model is able to produce more realistically looking magnifications during inference. The robust magnification approach was first evaluated and compared to previous work on a biomedical scenario in order to demonstrate its performance in a controlled setup. A new dataset which shows various people performing a golf swing at different tournaments was then used to evaluate the effectiveness of the robust magnification tool for more challenging setups.

The methods introduced in Chapter 3 to 5 demonstrated the immense potential unsupervised learning provides. Even though the performance is constantly

improving, and the field of unsupervised learning is gaining more and more attention, there is still no substitute for methods that use large labeled datasets. Thus, this thesis additionally investigated the improvement in performance when adding only a few labeled samples to the unlabeled dataset. In particular, Chapter 6 introduced a semi-supervised approach for video recognition that efficiently combines large unlabeled datasets with few labeled samples. The proposed method consists of three main steps, namely a self-supervised pre-training (only unlabeled samples), supervised fine-tuning (only labelled samples) and a pseudo-labelling based post-training (labeled and unlabeled samples). Experiments on two benchmark datasets including a large-scale dataset with up to 400 classes showed a large boost in performance in comparison to a baseline which is only trained on the labeled samples. Ablation studies demonstrated that the unsupervised pre-training is an essential part of the approach to create a good basis for the subsequent steps. Moreover, the experiments have shown that using only few labeled samples can significantly improve the performance if they are efficiently combined with the large set of unlabeled samples.

In conclusion, unsupervised deep learning is a very important research topic and its applicability goes beyond standard computer vision tasks. The interdisciplinary project introduced in this thesis demonstrated the potential of unsupervised learning and its high utility to other fields. A combination of unsupervised and semi-supervised learning is, in my opinion, the most promising direction for future work and should be investigated further. However, this strategy presupposes the possibility of obtaining at least a small amount of annotations. If this is not given, unsupervised learning is certainly a very good alternative to learn powerful visual representations for images and videos.

Publications

This dissertation has led to the following scientific publications:

- Brattoli, B.*, **Büchler, U.***, Wahl, AS, Schwab, ME, Ommer, B. LSTM Self-Supervision for Detailed Behavior Analysis. In IEEE Computer Vision and Pattern Recognition (CVPR), 2017.
- Wahl, A.S.*, **Büchler, U.***, Brändli, A., Brattoli, B., Musall, S., Kasper, H., Ineichen, B.V., Helmchen, F., Ommer, B. and Schwab, M.E. Optogenetically stimulating intact rat corticospinal tract post-stroke restores motor control through regionalized functional circuit formation. Nature communications, 8(1), 2017.
- **Büchler, U.***, Brattoli, B.*, Ommer, B. Improving Spatiotemporal Self-Supervision by Deep Reinforcement Learning. In European Conference on Computer Vision (ECCV), 2018.
- Dorkenwald, M.*, **Büchler, U.***, Ommer, B. Unsupervised Magnification of Posture Deviation Across Subjects. In IEEE Computer Vision and Pattern Recognition (CVPR), 2020.

Under submission is currently the following publication:

- Brattoli, B.*, **Büchler, U.***, Dorkenwald, M., Reiser, P., Filli, L., Helmchen, F., Wahl, A.S., Ommer, B. uBAM: Unsupervised Behavior Analysis and Magnification using Deep Learning.

*Indicates equal contribution

Bibliography

[1] M. F. AbdulHalim and Z. A. Mejbil. Automatic colorization without human intervention. In *2008 International Conference on Computer and Communication Engineering*, pages 62–65. IEEE, 2008.

[2] P. Agrawal, J. Carreira, and J. Malik. Learning to see by moving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 37–45, 2015.

[3] U. Ahsan, C. Sun, and I. Essa. Discrimnet: Semi-supervised action recognition from videos using generative adversarial networks. *arXiv preprint arXiv:1801.07230*, 2018.

[4] M. Alaverdashvili and I. Q. Whishaw. A behavioral method for identifying recovery and compensation: hand use in a preclinical stroke model using the single pellet reaching task. *Neuroscience & Biobehavioral Reviews*, 37(5):950–967, 2013.

[5] B. Antic, U. Büchler, A.-S. Wahl, M. E. Schwab, and B. Ommer. Spatiotemporal parsing of motor kinematics for assessing stroke recovery. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 467–475. Springer, 2015.

[6] A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. In *Advances in neural information processing systems*, pages 41–48, 2007.

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

[8] G. Balakrishnan, A. Zhao, A. V. Dalca, F. Durand, and J. V. Guttag. Synthesizing images of humans in unseen poses. *CoRR*, abs/1804.07739, 2018.

[9] H. B. Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

[10] M. A. Bautista, A. Sanakoyeu, and B. Ommer. Deep unsupervised similarity learning using partially ordered sets. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, 2017.

[11] M. A. Bautista, A. Sanakoyeu, E. Tikhoncheva, and B. Ommer. Cliquecnn: Deep unsupervised exemplar learning. In *Advances in Neural Information Processing Systems*, pages 3846–3854, 2016.

[12] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[13] G. J. Berman. Measuring behavior across scales. *BMC biology*, 16(1):23, 2018.

[14] P. Bojanowski and A. Joulin. Unsupervised learning by predicting noise. *arXiv preprint arXiv:1704.05310*, 2017.

[15] B. Brattoli, U. Büchler, A. S. Wahl, M. E. Schwab, and B. Ommer. Lstm self-supervision for detailed behavior analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[16] U. Büchler, B. Brattoli, and B. Ommer. Improving spatiotemporal self-supervision by deep reinforcement learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 770–786, 2018.

[17] 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)*, pages 132–149, 2018.

[18] M. Caron, P. Bojanowski, J. Mairal, and A. Joulin. Unsupervised pre-training of image features on non-curated data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2959–2968, 2019.

[19] J. Carreira, E. Noland, C. Hillier, and A. Zisserman. A short note on the kinetics-700 human action dataset. *arXiv preprint arXiv:1907.06987*, 2019.

[20] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[21] E. Cetinic, T. Lipic, and S. Grgic. Fine-tuning convolutional neural networks for fine art classification. *Expert Systems with Applications*, 114:107–118, 2018.

[22] H. Chen, X. J. Qi, J. Z. Cheng, and P. A. Heng. Deep contextual networks for neuronal structure segmentation. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[23] L. Chen, P. Bentley, K. Mori, K. Misawa, M. Fujiwara, and D. Rueckert. Self-supervised learning for medical image analysis using image context restoration. *Medical image analysis*, 58:101539, 2019.

[24] Y. Chen, X. Zhu, and S. Gong. Semi-supervised deep learning with memory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 268–283, 2018.

[25] Z. Chen and X. Huang. End-to-end learning for lane keeping of self-driving cars. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1856–1860. IEEE, 2017.

[26] Z. Cheng, Q. Yang, and B. Sheng. Deep colorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 415–423, 2015.

[27] F. Chiaroni, M.-C. Rahal, N. Hueber, and F. Dufaux. Self-supervised learning for autonomous vehicles perception: A conciliation between analytical and learning methods. *arXiv preprint arXiv:1910.01636*, 2019.

[28] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[29] R. S. Cruz, B. Fernando, A. Cherian, and S. Gould. Deeppermnet: Visual permutation learning. In *CVPR*, 2017.

[30] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[31] T. Darrell and A. Pentland. Space-time gestures. In *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR'93., 1993 IEEE Computer Society Conference on*, pages 335–340. IEEE, 1993.

[32] A. Datta, M. Shah, and N. D. V. Lobo. Person-on-person violence detection in video data. In *Object recognition supported by user interaction for service robots*, volume 1, pages 433–438. IEEE, 2002.

[33] T. Dekel, T. Michaeli, M. Irani, and W. T. Freeman. Revealing and modifying non-local variations in a single image. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia*, 2015.

[34] E. L. Denton et al. Unsupervised learning of disentangled representations from video. In *Advances in neural information processing systems*, pages 4414–4423, 2017.

[35] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[36] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

[37] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros. What makes paris look like paris? *Communications of the ACM*, 58(12):103–110, 2015.

[38] C. Doersch and A. Zisserman. Multi-task self-supervised visual learning. *arXiv preprint arXiv:1708.07860*, 2017.

[39] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015.

[40] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[41] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[42] J. Donahue and K. Simonyan. Large scale adversarial representation learning. In *Advances in Neural Information Processing Systems*, pages 10541–10551, 2019.

[43] M. Dorkenwald, U. Büchler, and B. Ommer. Unsupervised magnification of posture deviations across subjects. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[44] A. Dosovitskiy, P. Fischer, J. T. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(9):1734–1747, 2015.

[45] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with convolutional neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 766–774. Curran Associates, Inc., 2014.

[46] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[47] L. Duong, T. Cohn, S. Bird, and P. Cook. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 845–850, 2015.

[48] M. Elgharib, M. Hefeeda, F. Durand, and W. T. Freeman. Video magnification in presence of large motions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4119–4127, 2015.

[49] P. Esser, E. Sutter, and B. Ommer. A variational u-net for conditional appearance and shape generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8857–8866, 2018.

[50] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge 2007 (voc2007) results. 2007.

[51] M. Everingham and J. Winn. The pascal visual object classes challenge 2012 (voc2012) development kit. *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, 2011.

[52] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[53] S. Fernández, A. Graves, and J. Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. In *International Conference on Artificial Neural Networks*, pages 220–229. Springer, 2007.

[54] B. Fernando, H. Bilen, E. Gavves, and S. Gould. Self-supervised video representation learning with odd-one-out networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[55] L. Filli, J. Werner, G. Beyer, K. Reuter, J. Petersen, M. Weller, B. Zörner, and M. Linnebank. Predicting responsiveness to fampridine in gait-impaired patients with multiple sclerosis. *European journal of neurology*, 26(2):281–289, 2019.

[56] J. Friedman, T. Hastie, R. Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[57] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[58] P. Gebert, A. Roitberg, M. Haurilet, and R. Stiefelhagen. End-to-end prediction of driver intention using 3d convolutional neural networks. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 969–974. IEEE, 2019.

[59] G. Ghiasi, T.-Y. Lin, and Q. V. Le. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pages 10727–10737, 2018.

[60] S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018.

[61] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[62] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[63] B. Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[64] D. Guan, W. Yuan, Y.-K. Lee, A. Gavrilov, and S. Lee. Activity recognition based on semi-supervised learning. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*, pages 469–475. IEEE, 2007.

[65] B. Hariharan, J. Malik, and D. Ramanan. Discriminative decorrelation for clustering and classification. In *European Conference on Computer Vision*, pages 459–472. Springer, 2012.

[66] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[67] 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*, pages 770–778, 2016.

[68] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[69] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[70] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[71] S. Iizuka, E. Simo-Serra, and H. Ishikawa. Let there be color! joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Transactions on Graphics (ToG)*, 35(4):1–11, 2016.

[72] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. In *European Conference on Computer Vision*, pages 34–50. Springer, 2016.

[73] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[74] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum. Label propagation for deep semi-supervised learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5070–5079, 2019.

[75] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

[76] C. Jia, Z. Ding, Y. Kong, and Y. Fu. Semi-supervised cross-modality action recognition by latent tensor transfer learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[77] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast

feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[78] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[79] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[80] S. Johnson and M. Everingham. Clustered pose and nonlinear appearance models for human pose estimation. In *Proceedings of the British Machine Vision Conference*, 2010. doi:10.5244/C.24.12.

[81] M. Kabra1, A. A. Robie1, M. Rivera-Alba1, S. Branson, and K. Branson. Jaaba: interactive machine learning for automatic annotation of animal behavior. *Nature methods*, 10, 2012.

[82] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[83] A. Khoreva, R. Benenson, M. Omran, M. Hein, and B. Schiele. Weakly supervised object boundaries. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 183–192, 2016.

[84] D. Kim, D. Cho, D. Yoo, and I. S. Kweon. Learning image representations by completing damaged jigsaw puzzles. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 793–802. IEEE, 2018.

[85] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[86] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[87] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.

[88] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell. Data-dependent initializations of convolutional neural networks. *arXiv preprint arXiv:1511.06856*, 2015.

[89] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *University of Toronto*, 2009.

[90] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[91] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

[92] S. Laine and T. Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

[93] G. Larsson, M. Maire, and G. Shakhnarovich. Learning representations for automatic colorization. In *European Conference on Computer Vision*, pages 577–593. Springer, 2016.

[94] G. Larsson, M. Maire, and G. Shakhnarovich. Colorization as a proxy task for visual understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6874–6883, 2017.

[95] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[96] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[97] 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*, volume 3, page 2, 2013.

[98] H.-Y. Lee, J.-B. Huang, M. K. Singh, and M.-H. Yang. Unsupervised representation learning by sorting sequences. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[99] C. Liu. *Beyond pixels: exploring new representations and applications for motion analysis*. PhD thesis, Citeseer, 2009.

[100] C. Liu, A. Torralba, W. T. Freeman, F. Durand, and E. H. Adelson. Motion magnification. In *ACM transactions on graphics (TOG)*, volume 24, pages 519–526. ACM, 2005.

[101] 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*, pages 3431–3440, 2015.

[102] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[103] Z. Luo, B. Peng, D.-A. Huang, A. Alahi, and L. Fei-Fei. Unsupervised learning of long-term motion dynamics for videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[104] Z. Luo, B. Peng, D.-A. Huang, A. Alahi, and L. Fei-Fei. Unsupervised learning of long-term motion dynamics for videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2203–2212, 2017.

[105] Y. Ma and H. Ghasemzadeh. Labelforest: Non-parametric semi-supervised learning for activity recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4520–4527, 2019.

[106] M. F. Mabrouk, N. M. Ghanem, and M. A. Ismail. Semi supervised learning for human activity recognition using depth cameras. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 681–686. IEEE, 2015.

[107] P. Madhu, R. Kosti, L. Mührenberg, P. Bell, A. Maier, and V. Christlein. Recognizing characters in art history using deep learning. In *Proceedings of the 1st Workshop on Structuring and Understanding of Multimedia heritAge Contents*, pages 15–22, 2019.

[108] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *ICCV*, 2011.

[109] M. Martin, A. Roitberg, M. Haurilet, M. Horne, S. Reiß, M. Voit, and R. Stiefelhagen. Drive&act: A multi-modal dataset for fine-grained driver behavior recognition in autonomous vehicles. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2801–2810, 2019.

[110] B. Martinez, D. Modolo, Y. Xiong, and J. Tighe. Action recognition with spatial-temporal discriminative filter banks. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[111] S. Masood, A. Rai, A. Aggarwal, M. N. Doja, and M. Ahmad. Detecting distraction of drivers using convolutional neural network. *Pattern Recognition Letters*, 2018.

[112] A. Mathis, P. Mamidanna, K. M. Cury, T. Abe, V. N. Murthy, M. W. Mathis, and M. Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21(9):1281–1289, 9 2018.

[113] J. Mayr, C. Unger, and F. Tombari. Self-supervised learning of the drivable area for autonomous vehicles. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 362–369. IEEE, 2018.

[114] T. Milbich, M. Bautista, E. Sutter, and B. Ommer. Unsupervised video understanding by reconciliation of posture similarities. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[115] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3994–4003, 2016.

[116] I. Misra, C. L. Zitnick, and M. Hebert. Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision*, pages 527–544. Springer, 2016.

[117] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

[118] S. Mohammadi, A. Perina, H. Kiani, and V. Murino. Angry crowds: Detecting violent events in videos. In *European Conference on Computer Vision*, pages 3–18. Springer, 2016.

[119] T. Nathan Mundhenk, D. Ho, and B. Y. Chen. Improvements to context based self-supervised learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9339–9348, 2018.

[120] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[121] 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*, pages 392–405. Springer, 2010.

[122] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *IEEE European Conference on Computer Vision (ECCV)*, 2016.

[123] M. Noroozi, H. Pirsiavash, and P. Favaro. Representation learning by learning to count. *arXiv preprint arXiv:1708.06734*, 2017.

[124] M. Noroozi, A. Vinjimoor, P. Favaro, and H. Pirsiavash. Boosting self-supervised learning via knowledge transfer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[125] T.-H. Oh, R. Jaroensri, C. Kim, M. Elgharib, F. Durand, W. T. Freeman, and W. Matusik. Learning-based video motion magnification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 633–648, 2018.

[126] A. Owens, J. Wu, J. H. McDermott, W. T. Freeman, and A. Torralba. Ambient sound provides supervision for visual learning. In *European Conference on Computer Vision*, pages 801–816. Springer, 2016.

[127] Y. Patel, L. Gomez, M. Rusiñol, C. Jawahar, and D. Karatzas. Self-supervised learning of visual features through embedding images into text topic spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[128] D. Pathak, R. Girshick, P. Dollár, T. Darrell, and B. Hariharan. Learning features by watching objects move. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[129] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016.

[130] T. D. Pereira, D. E. Aldarondo, L. Willmore, M. Kislin, S. S.-H. Wang, M. Murthy, and J. W. Shaevitz. Fast animal pose estimation using deep neural networks. *Nature methods*, 16(1):117–125, 2019.

[131] S. M. Peters, I. J. Pinter, H. H. Pothuizen, R. C. de Heer, J. E. van der Harst, and B. M. Spruijt. Novel approach to automatically classify rat social behavior using a video tracking system. *Journal of neuroscience methods*, 268:163–170, 2016.

[132] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. V. Gehler, and B. Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4929–4937, 2016.

[133] R. Poplin, A. V. Varadarajan, K. Blumer, Y. Liu, M. V. McConnell, G. S. Corrado, L. Peng, and D. R. Webster. Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning. *Nature Biomedical Engineering*, 2(3):158, 2018.

[134] S. Purushwalkam and A. Gupta. Pose from action: Unsupervised learning of pose features based on motion. *arXiv preprint arXiv:1609.05420*, 2016.

[135] H. Qian, S. J. Pan, and C. Miao. Distribution-based semi-supervised learning for activity recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7699–7706, 2019.

[136] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[137] V. Ramakrishna, D. Munoz, M. Hebert, A. J. Bagnell, and Y. Sheikh. Pose machines: Articulated pose estimation via inference machines. In *ECCV*, 2014.

[138] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko. Semi-supervised learning with ladder networks. In *Advances in neural information processing systems*, pages 3546–3554, 2015.

[139] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[140] C. Ritter, T. Wollmann, P. Bernhard, M. Gunkel, D. M. Braun, J.-Y. Lee, J. Meiners, R. Simon, G. Sauter, H. Erfle, et al. Hyperparameter optimization for image analysis: application to prostate tissue images and live cell data of virus-infected cells. *International journal of computer assisted radiology and surgery*, 14(11):1847–1857, 2019.

[141] A. A. Robie, K. M. Seagraves, S. R. Egnor, and K. Branson. Machine vision methods for analyzing social interactions. *Journal of Experimental Biology*, 220(1):25–34, 2017.

[142] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[143] H. Ryait, E. Bermudez-Contreras, M. Harvey, J. Faraji, B. M. Agha, A. G.-P. Schjetnan, A. Gruber, J. Doan, M. Mohajerani, G. A. Metz, et al. Data-driven analyses of motor impairments in animal models of neurological disorders. *PLoS biology*, 17(11), 2019.

[144] S. P. Sahoo, U. Srinivasu, and S. Ari. 3d features for human action recognition with semi-supervised learning. *IET Image Processing*, 13(6):983–990, 2019.

[145] A. Sanakoyeu, M. A. Bautista, and B. Ommer. Deep unsupervised learning of visual similarities. *Pattern Recognition*, 78:331–343, 2018.

[146] N. Sayed, B. Brattoli, and B. Ommer. Cross and learn: Cross-modal self-supervision. In *German Conference on Pattern Recognition*, pages 228–243. Springer, 2018.

[147] H.-C. Shin, K. Roberts, L. Lu, D. Demner-Fushman, J. Yao, and R. M. Summers. Learning to read chest x-rays: Recurrent neural cascade model for automated image annotation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2497–2506, 2016.

[148] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[149] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[150] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[151] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[152] G. Strezoski and M. Worring. Omniart: multi-task deep learning for artistic data analysis. *arXiv preprint arXiv:1708.00684*, 2017.

[153] H.-I. Suk, D. Shen, A. D. N. Initiative, et al. Deep learning in diagnosis of brain disorders. In *Recent Progress in Brain and Cognitive Engineering*, pages 203–213. Springer, 2015.

[154] W. Sultani, C. Chen, and M. Shah. Real-world anomaly detection in surveillance videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6479–6488, 2018.

[155] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[156] 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*, pages 1–9, 2015.

[157] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[158] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[159] A. Taleb, C. Lippert, T. Klein, and M. Nabi. Multimodal self-supervised learning for medical image analysis. *arXiv preprint arXiv:1912.05396*, 2019.

[160] T. Tlusty, T. Michaeli, T. Dekel, and L. Zelnik-Manor. Modifying non-local variations across multiple views. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[161] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1653–1660, 2014.

[162] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[163] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

[164] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

[165] P. V. Tran. Exploring self-supervised regularization for supervised and semi-supervised learning. *arXiv preprint arXiv:1906.10343*, 2019.

[166] S. Tulyakov, X. Alameda-Pineda, E. Ricci, L. Yin, J. F. Cohn, and N. Sebe. Self-adaptive matrix completion for heart rate estimation from face videos under realistic conditions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2396–2404, 2016.

[167] C. E. Vargas-Irwin, G. Shakhnarovich, P. Yadollahpour, J. M. Mislow, M. J. Black, and J. P. Donoghue. Decoding complete reach and grasp actions from local primary motor cortex populations. *Journal of neuroscience*, 30(29):9659–9669, 2010.

[168] M. Veta, Y. J. Heng, N. Stathonikos, B. E. Bejnordi, F. Beca, T. Wollmann, K. Rohr, M. A. Shah, D. Wang, M. Rousson, et al. Predicting breast tumor proliferation from whole-slide images: the tupac16 challenge. *Medical image analysis*, 54:111–121, 2019.

[169] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[170] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *Conference on Neural Information Processing Systems (NIPS)*, 2016.

[171] C. Vondrick, A. Shrivastava, A. Fathi, S. Guadarrama, and K. Murphy. Tracking emerges by colorizing videos. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 391–408, 2018.

[172] N. Wadhwa, T. Dekel, D. Wei, F. Durand, and W. T. Freeman. Deviation magnification: Revealing departure from ideal geometries. *ACM Trans. Graph. (Proceedings SIGGRAPH Asia 2015)*, 34(6), 2015.

[173] N. Wadhwa, M. Rubinstein, F. Durand, and W. T. Freeman. Phase-based video motion processing. *ACM Transactions on Graphics (TOG)*, 32(4):80, 2013.

[174] N. Wadhwa, M. Rubinstein, F. Durand, and W. T. Freeman. Riesz pyramids for fast phase-based video magnification. In *2014 IEEE International Conference on Computational Photography (ICCP)*, pages 1–10. IEEE, 2014.

[175] A.-S. Wahl, U. Büchler, A. Brändli, B. Brattoli, S. Musall, H. Kasper, B. V. Ineichen, F. Helmchen, B. Ommer, and M. E. Schwab. Optogenetically stimulating intact rat corticospinal tract post-stroke restores motor control through regionalized functional circuit formation. *Nature communications*, 8(1):1187, 2017.

[176] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.

[177] J. Wang, J. Jiao, L. Bao, S. He, Y. Liu, and W. Liu. Self-supervised spatio-temporal representation learning for videos by predicting motion and appearance statistics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4006–4015, 2019.

[178] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Val Gool. Temporal segment networks: Towards good practices for deep action recognition. In *IEEE European Conference on Computer Vision (ECCV)*, 2016.

[179] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2015.

[180] X. Wang, K. He, and A. Gupta. Transitive invariance for self-supervised visual representation learning. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[181] X. Wang, A. Jabri, and A. A. Efros. Learning correspondence from the cycle-consistency of time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2566–2576, 2019.

[182] J. Wright, A. Ganesh, S. Rao, Y. Peng, and Y. Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2080–2088. Curran Associates, Inc., 2009.

[183] H.-Y. Wu, M. Rubinstein, E. Shih, J. Guttag, F. Durand, and W. T. Freeman. Eulerian video magnification for revealing subtle changes in the world. *ACM Transactions on Graphics (Proc. SIGGRAPH 2012)*, 31(4), 2012.

[184] Y. Wu and K. He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[185] J. Xie, R. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487, 2016.

[186] D. Xu, E. Ricci, Y. Yan, J. Song, and N. Sebe. Learning deep representations of appearance and motion for anomalous event detection. *arXiv preprint arXiv:1510.01553*, 2015.

[187] D. Xu, J. Xiao, Z. Zhao, J. Shao, D. Xie, and Y. Zhuang. Self-supervised spatiotemporal learning via video clip order prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10334–10343, 2019.

[188] C. Yan, F. Coenen, and B. Zhang. Driving posture recognition by joint application of motion history image and pyramid histogram of oriented gradients. *International journal of vehicular technology*, 2014, 2014.

[189] H. Yang, H. Wu, and H. Chen. Detecting 11k classes: Large scale object detection without fine-grained bounding boxes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9805–9813, 2019.

[190] J. Yang, D. Parikh, and D. Batra. Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5147–5156, 2016.

[191] A. S. Yoon, T. Lee, Y. Lim, D. Jung, P. Kang, D. Kim, K. Park, and Y. Choi. Semi-supervised learning with deep generative models for asset failure prediction. *arXiv preprint arXiv:1709.00845*, 2017.

[192] H. Yuan and C. Wang. A human action recognition algorithm based on semi-supervised kmeans clustering. In *Transactions on edutainment VI*, pages 227–236. Springer, 2011.

[193] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[194] X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE international conference on computer vision*, pages 1476–1485, 2019.

[195] J. Zhang, Y. Han, J. Tang, Q. Hu, and J. Jiang. Semi-supervised image-to-video adaptation for video action recognition. *IEEE transactions on cybernetics*, 47(4):960–973, 2016.

[196] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016.

[197] R. Zhang, P. Isola, and A. A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. *arXiv preprint arXiv:1611.09842*, 2016.

[198] W. Zhang, R. Li, H. Deng, L. Wang, W. Lin, S. Ji, and D. Shen. Deep convolutional neural networks for multi-modality isointense infant brain image segmentation. *NeuroImage*, 108:214–224, 2015.

[199] Y. Zhang, S. L. Pintea, and J. C. Van Gemert. Video acceleration magnification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 529–537, 2017.

[200] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Advances in neural information processing systems*, pages 487–495, 2014.