Inaugural dissertation
for
obtaining the doctoral degree
of the
Combined Faculty of Mathematics, Engineering
and Natural Sciences
of the
Ruprecht - Karls - University
Heidelberg

Presented by
M.Sc. Fynn Beuttenmüller
born in: Bremervörde, Germany
Oral examination: 27 November, 2024

Accessible Deep Learning for Bio-Image Analysis

Referees: Prof. Dr. Thomas Greb
Dr. Robert Prevedel

Accessible Deep Learning for Bio-Image Analysis

Bioimage analysis is undergoing groundbreaking transformations in the advent of deep learning. AI powered tools bring unrivaled capabilities to the analysis of biological images that allow us to extract novel insights at an unprecedented rate. In this thesis I present two major contributions towards making deep learning more accessible and impactful for the bioimaging community.

First, I introduce a novel hybrid microscopy technique that utilizes advanced reconstruction models to capture volumetric recordings of highly dynamic samples with exceptional image fidelity. I leverage the efficacy of deep learning to integrate complementing microscopy approaches which extends the pool of viable biological studies. With a focus on continuous validation the system allows experimentalists to reliably analyze their data in a quantitative manner and demonstrates how the power of deep learning can be used responsibly.

Furthermore I made key contributions to establish a FAIR community-driven model zoo for bioimage analysis. The platform facilitates knowledge sharing by increasing the discoverability and dissemination of pretrained deep learning models, biological image datasets and related computational tools. It accelerates the responsible adoption of AI techniques in the bioimaging field by fostering reproducibility and transparency. The central model zoo enables interoperability across a growing number of community partner software programs and empowers biologists to integrate state-of-the-art neural networks in their research without friction. This collaborative effort aims to democratize the use of deep learning in bioimaging, making cutting-edge technologies more accessible and user-friendly for a diverse scientific audience.

Leicht zugängliches Deep Learning für die biologische Bildanalyse

Die biologische Bildanalyse verändert sich wegweisend durch den Einsatz von Deep Learning. Neue KI-gestützte Möglichkeiten bei der Analyse biologischer Bilder lassen uns neuartige Erkenntnisse in beispielloser Geschwindigkeit gewinnen. In dieser Dissertation präsentiere ich zwei wesentliche Beiträge, um die Zugänglichkeit und Effektivität von Deep Learning Ansätzen im Bereich der biologischen Bildanalyse zu verbessern.

Zunächst stelle ich eine neuartige hybride Mikroskopietechnik vor, die fortschrittliche Rekonstruktionsmodelle nutzt, um volumetrische Aufnahmen sehr dynamischer Proben in außergewöhnlicher Bildqualität zu erfassen. Ich nutze die Wirksamkeit von Deep Learning, um komplementäre Mikroskopieansätze zu integrieren, wodurch die Bandbreite möglicher biologischer Studien erweitert wird. Mit einem Fokus auf kontinuierliche Validierung unterstützt das System Forscher darin, ihre Daten zuverlässig und quantitativ zu analysieren, und zeigt, wie das Potenzial von Deep Learning verantwortungsvoll genutzt werden kann.

Darüber hinaus präsentiere ich grundlegende Beiträge zum Etablieren einer FAIRen, Community-verwalteten Sammlung von Modellen für die biologische Bildanalyse. Die neue Plattform erleichtert den Wissensaustausch, indem sie die Verbreitung vortrainierter Deep-Learning-Modelle, biologischer Datensätze und relevanter Computerprogramme verbessert. Sie beschleunigt die verantwortungsvolle Einführung von KI-Techniken im Bereich der biologischen Bildverarbeitung durch die Förderung von Reproduzierbarkeit und Transparenz. Die zentrale Modellsammlung ermöglicht die Interoperabilität zwischen einer wachsenden Anzahl von Community-Partner-Softwareprogrammen und befähigt Biologen, hochmoderne neuronale Netze reibungslos in ihre Forschung zu integrieren. Dieses kollaborative Projekt zielt darauf ab, die Nutzung von Deep Learning für die biologische Bildanalyse zu demokratisieren, um einer diversen wissenschaftlichen Gemeinschaft Spitzentechnologien leichter zugänglich zu machen.

Acknowledgments

I want to thank my family, who stood by my side throughout my PhD time. Luciana, special thanks to you, once you can read. You are simply perfect! Thank you for giving me valuable perspective on the problems I encountered on the way to finishing this thesis.

I also want to thank the Kreshuk Lab at EMBL — past and present members — for providing such a friendly working environment. Thank you, Anna Kreshuk, for your amazing supervision. You have been an inspiration to me again and again. Thank you, Dominik Kutra, for your valuable feedback on my thesis drafts and thank you, as well as Tomaz Vieira, Maksim Novikov, Emil Melnikov and Benedikt Best for all those fruitful discussions on coding, software design and life. Thank you fellow Predocs — it has been a pleasure! My gratitude for a pleasant working environment extends to EMBL at large. Special shout-out to the Kinderhaus for taking such good care of Luciana and to our great cooks and baristas.

Thank you Nils Wagner et al. for a fantastic collaboration.
I am also grateful to Anna Kreshuk, Florian Jug and Wei Ouyang for envisioning the BioImage.IO Model Zoo. Thank you to all those involved in the BioImage.IO community, in particular Constantin Pape, Wei Ouyang and Estibaliz Gómez-de-Mariscal for those early hacking hours creating the zoo. 🦒

Contents

1 Introduction

Over the history of science, we reduced our dependence on our senses like smell, taste and vision for assessment of scientific experiments and discovery; measuring devices help to quantify the world around us and to conduct reproducible science. A similar trend can be observed for the analysis of collected data: From intuition and human interpretation to statistics, from prehistoric calendars to deep learning (DL).

Let us zoom in on the scientific landscape to biology. From research in cell biology informed by electron microscopy (EM) to advancing astrobiology with telescopes — imaging is an important source of information for biological research. Merely acquiring swaths of image data ceases to be expedient, if we were to rely on human analysis only. Computational image analysis comes to the rescue! And the complexity of computational analyses is rising in the age of DL.

Scientists are not all polymaths. Accessible computational tools should allow biologists to efficiently analyze their data without having to focus on software development themselves. Novel analysis pipelines, like novel protocols or instruments, steadily update and expand an experimentalist's toolbox. Ready-to-use tools for (semi-)automated image analysis are essential to increase throughput, ease reproducibility, and empower more researchers with access to modern image analysis methods.

Beyond scaling and efficiency, automated analysis can lessen human biases and provide super human capabilities: From numerical precision, for example when detecting subtle differences in intensity, to complex statistical analysis and AI. Humans also struggle notoriously with image analysis in three or even more dimensions.

In the field of AI, machine learning (ML), more specifically DL, has recently lead to impressive advances. DL utilizes (artificial) neural networks (NNs) with more than three layers (typically many more). Each layer performs a relatively simple mathematical operation.

Through non-linear connections of these layers NNs gain their interesting property of being a universal function approximator[1]. The promise: Given an appropriate training algorithm and sufficient training data an NN can learn anything!

1.1 Challenges of Deep Learning in Bioimage Analysis

DL has long become a crucial tool in the toolbox of modern data analysis. In particular in the area of biology and medicine significant impact and potential for further transformation has been ascertained[2, 3, 4, 5, 6]. Despite many success stories — AlphaFold[7, 8] (protein structure prediction), StarDist[9, 10] (image segmentation), CARE[11] (image reconstruction), Noise2Void[12] and Noise2Noise[13] (image denoising) to name a few — a question arises: If DL is such a powerful tool, why is it not used ubiquitously throughout the biological and medical research communities? In other words: What is holding DL back? The answer is multifaceted.

Current challenges for DL in the context of bioimage analysis are a lack of interpretability (the black box problem), the need for suitable, large training datasets, model architecture selection, and the high computational cost of training[3]. Additionally the limited prevalence of DL expertise among biologists limits a wider dissemination of DL techniques among research projects without suitable collaborators.

1.1.1 How to Trust Deep Learning?

DL models commonly used today are not as explainable, nor as interpretable in comparison to simpler statistical approaches, which limits their application in particular in the area of medicine[4], but also in biological research[14]. "[S]everal factors may cause model and data uncertainty and affect a [deep neural network's (DNN's)] prediction. This variety of sources of uncertainty makes the complete exclusion of uncertainties in a neural network impossible for almost all applications."[15] This difficult-to-assess prediction uncertainty can lead to a lack of trust in DL based methods. In addition researchers are often not only interested in accurate predictions for the task at hand, but also *why* a given prediction is even possible[14]. They seek to interpret DL model black boxes and their outputs. Furthermore, downstream statistical analysis of results (and to some extent benchmarking[14]) is hampered by this

black box problem that remains a challenge in biomedical applications and research[3, 4].

Explainable AI (XAI)[16] is an active research area that addresses this challenge in general[17, 18, 19]. Reliable estimates of predictive uncertainty might also alleviate the issue partially, but Ching et al. note that "unfortunately, uncertainty quantification techniques are underused in the computational biology communities and largely ignored in the current deep learning for biomedicine literature"[4]. This might change in the future as "uncertainty quantification for neural networks continues to be an active research area"[4].

If DL methods are not as amenable to explanation and understanding as statistical approaches, how can we trust results derived by DL nonetheless? The answer has proven to be validation. If we can reproduce measurable results consistently we can quantify our expectations and eventually trust novel findings. The most powerful and straight-forward avenue to establish trust through validation is to use independent data for which desired query results are known — so called ground truth in the context of supervised learning. Already knowing what a computational model is designed to predict is not reducing the problem to absurdity; obtaining ground truth might be destructive, costly, time or labor intensive. Given sufficiently large quantities of trustworthy training data though, a NN can be trained and validated once and then applied very efficiently henceforth. However, while this is certainly a workable solution initially, trust will (and should) diminish with changes to the application domain of a highly specialized NN. Even a small shift in the source domain, like subtle changes to an imaging setup over time may result in degraded performance. Model predictions may not always fail in obvious ways; a common issue in image restoration for example is hallucination[20, 21]. Cohen et al. describe a "fundamental tradeoff between uncertainty and perception"[21] for generative restoration models. Ideally DL models are validated continuously to not only initially gain our trust, but to also keep it.

1.1.2 Data Availability

For many biological problems data availability is a significant bottleneck — in particular the acquisition of ground truth (labeled data)[4]. For some problems clever data acquisition schemes can be designed to effectively generate ground truth. An example from image restoration: A model can be trained to denoise low Signal-to-noise ratio (SNR) images with pairs of low and high SNR samples[11].

Particularly models trained on small datasets are prone to overfitting. Fitting training data too closely decreases generalization to unseen data samples. Regularization methods like *weight decay*[22] or *dropout*[23] and pre-training — on natural images[24], or unsupervised (on unlabeled data)[25] — can reduce overfitting, but DL methods remain data-hungry. Datasets can be virtually enlarged to some extent by augmenting the model inputs (and ground truth) to mimic plausible alternative training samples. Examples of such augmentations are rotations, mirroring or the addition of noise.

Available data is often heterogeneous, which, even though NNs are better equipped to handle compared to statistical analysis[3], can drive up training complexity and cost. As in any field, severe discrepancies in the distributions of training and (real-world) test data also limit the application of DL in biology[14].

Another concern, particularly in medicine, is preserving (patient) data privacy when assembling datasets to train NNs[4, 3] or applying trained NNs to new samples[26]. Approaches to tackle these privacy concerns include methods based on homomorphic encryption[26] and secure multi-party computation (SMPC)[27, 28].

It is evident that high quality, publicly available datasets are a valuable resource to allow reusing data for similar tasks or across similar modalities. While reusing data is crucial, pretrained models exacerbates the need for validation.

1.1.3 Training Neural Networks

Angermueller et al. remark that "no single [data analysis] method is universally applicable, and the choice of whether and how to use deep learning approaches will be problem-specific."[2] For tasks amenable to DL based solutions the choice or design of a specific method typically includes selecting or creating a suitable NN and training algorithm, which often requires some insight into the field of DL. Despite growing efforts to equip biologist with computational skills, ML is typically not part of a formal biological education. Thus the capability to use and customize ML approaches constitutes a bottleneck. To some extent this knowledge gap can be mediated through collaborations, but with ML expertise sought after for many human activities, suitable collaborators might not always be available and are costly.

The NN architecture determines a network's inputs and outputs and how the outputs are computed from the inputs using the network's learnable parameters. Finding a set of parameter values — also known as model weights — which generate the most useful output, is the goal of network training. To adapt a given training algorithm to a specific problem parameters of the training algorithm itself — so called hyperparameters — need to be determined.

The wide range of model architectures and the intricacies of NN training make designing point-and-click solutions challenging. Allocating developer time to make software accessible in this manner to cater to users without coding experience is a delicate trade-off[29].

Neural Network Architecture

The number, type and size of the individual network layers constituting a NN — a network's architecture — influences performance potential, hardware demands and computation speed.

Common network architectures are based on, or are a combination of, various network prototypes. These range from multilayer perceptrons (MLPs) (the simplest feedforward NNs) over convolutional neural networks (CNNs) — NNs with convolutional layers, from the early LeNet-5[30] to the popular U-Net[31], ResNets[32] and DenseNets[33] to name a few — to the more recent (Vision[34]) Transformers[35] — NNs with special attention mechanisms.

The network architecture also includes design choices like adding special layers, for example a dropout[23] layer for regularization to improve training speed/stability. Other popular regularization methods integrated into the architecture are batch normalization[36] and group normalization[37]. Some of these techniques were introduced together with a novel architecture, like Label Smoothing — introducing noise to ground truth annotations — which was first utilized in the Inception-v3 CNN[38]. Best practices in designing network architectures are evolving and are task, or even problem specific.

Training Algorithms

Training NNs is an optimization problem. It requires an objective function, typically formulated as a loss function, for example the mean squared error (MSE). To minimize this loss,

an optimization algorithm like stochastic gradient descent (SGD) or Adam[39] is used to iteratively perform optimization steps computed from mini-batches of the available training data.

Supervised learning algorithms can be leveraged to tackle biological problems only if training data for which (pseudo) annotations or otherwise associated ground truth exists. In general supervised learning algorithms are more powerful than unsupervised learning methods which can detect patterns, cluster, and learn representations of the data. Again, choosing which approach is suitable for a problem requires expertise.

Data Normalization

Biological (image) data needs to be preprocessed — normalized to account for changing imaging conditions and improve the numerical stability of the training algorithm. Popular choices include 'min-max normalization' — scaling the data such that values lie in the interval $[0, 1]$ — its generalization 'percentile normalization' — replacing the minimum/maximum with a low/high percentile value — and 'zero mean unit variance normalization' — scaling the dataset mean to zero and its variance to one. Also more complex procedures like histogram matching are used[40].

Hyperparameters

Parameters to configure the network architecture, training optimizer, loss function, data normalization, and augmentations are fixed during training and therefore referred to as hyperparameters. Choosing the right hyperparameters is often guided by trial and error and generally requires some experience with DL. Learning hyperparameters is part of the ML subfield of Metalearning.

Computation

Even when a model architecture is chosen, suitable training datasets are available, and all hyperparameters set, training an NN is not trivial. Unlike most classical ML algorithms, DL approaches typically entail a particularly lengthy training step. The associated computational costs include actually running the training procedure on suitable hardware (especially GPUs).

1.1.4 Usability of Trained Neural Networks

Even with a readily trained NN, scaling up inference (making predictions) can be computationally challenging[4]. Approaches to reduce computational requirements algorithmically exist. One such technique is pruning (removing NN parameters), which typically entails fine-tuning (further training) to partially compensate the loss of accuracy[41]. More NN training is also needed for knowledge distillation approaches (training an NN from a bigger one or from a network ensemble)[42]. Reducing numerical precision in training and inference also helps to alleviate hardware constrains[43, 44]. Even before the initial model training, efforts can be made to use particularly efficient network architectures, for example by deploying neural architecture search (NAS) to that end[45].
Despite these efforts, specialized hardware, most commonly GPUs, remain paramount to DL applications at scale.

Evaluating and improving the transferability of trained models to new data domains, known as transfer learning, is an active research area[46] and another key challenge to deploy trained NNs.

1.1.5 Making DL more Accessible

Capabilities of bioimage analysis pipelines have increased dramatically with the advent of deep learning. Even though large DL frameworks like PyTorch[47] or TensorFlow[48] made working with neural networks much easier and more computationally efficient, training a neural network still requires expert knowledge and suitable hardware (in particular GPUs or Tensor Processing Units (TPUs)). Inference (obtaining predictions) is less demanding in skill and hardware and thus more accessible to a wider range of users. It follows that efforts to make DL more accessible to many areas of biological research have two stages: (i) capitalize on the value of pretrained models published by DL experts and (ii) democratize the training process to reduce dependence on DL experts. The value of pretrained networks also derives from the energy and time consumed by the training process; where pretrained models are insufficient, fine-tuning a pretrained model can minimize energy and time requirements and has the additional benefit of requiring less reference data compared to training from scratch.

1.2 In this thesis

In my first thesis project, detailed in chapter 2 *Deep Learning-Enhanced Light-Field Imaging with Continuous Validation* I introduce a fast and trustable volume reconstruction approach which builds on hybrid microscopy technology supporting integrated acquisition of validation data. I evaluate my DL based reconstruction method on artificial samples in addition to evaluating and demonstrating its capabilities on arrested and beating hatchling medaka hearts. Finally I showcase its potential for neural signal tracing in zebrafish larvae.

In chapter 3 *BioImage Model Zoo: A Community-Driven Resource for Accessible Deep Learning in BioImage Analysis* I present my substantial contributions to a large community project to advance accessibility to and dissemination of DL methods relevant to bioimage analysis in general. Together with collaborators I developed a new metadata standard for creating and sharing DL models. The standard is accompanied by an ecosystem of Python libraries that I developed, which allow for easy programmatic access to — and handling of — shared resources, as well as their fast integration into other Python projects within the community.

2 Deep Learning-Enhanced Light-Field Imaging with Continuous Validation

To study biological structures scientists have used microscopes as early as 1665[49]. With an early compact light microscope Robert Hooke observed the texture of thin cork slices (see figure 2.1) based on which he coined the term "cells"[50]. Almost 200 years later botanist Matthias Schleiden and zoologist Theodore Schwann described cells as the principal building-blocks of plants/animals in 1838/1839[51]. This eventually gave rise to cell theory and cells have been an important biological unit ever since. A lot of details about cells remain unknown, e.g. the biological role of "around 20% of the proteins even in well-studied model organisms"[52].

To image cells or tissue biologists use a range of sample preparations and different microscopy techniques as subcellular structures are generally not examinable by the naked eye as figure 2.2 illustrates. To study biological structures down to the atomic scale X-ray crystallography and electron microscopy play an important role to advance our understanding of biology. However these methods cannot be applied at scale and are inherently unsuitable for dynamic imaging.

Many light microscopy techniques require or greatly benefit from chemical or biological staining. Staining methods are used to enhance the contrast of — or uncover — specific cell compounds with suitable dyes. Some dyes are fluorescent. Alternatively fluorescent proteins can be synthesized by living cells themselves in transgenic organisms.

Dyes are often used in conjunction with fixation — techniques to preserve a sample and freeze it in time (quite literally through cryofixation for example). This fatal step is also designed to help permeate a dye used for staining more efficiently throughout the specimen and prevent any unintended reactions of the live organism. Additionally fixation reduces

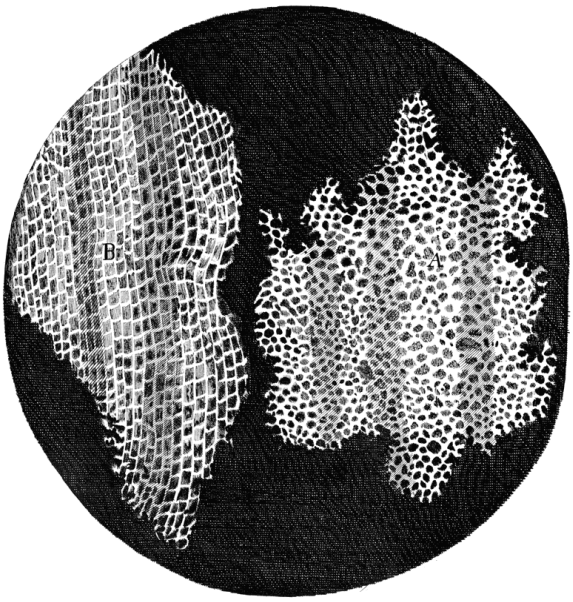


Figure 2.1: Thin slices of cork observed and illustrated by Robert Hooke in his 1665 book *Micrographia: or, Some physiological descriptions of minute bodies made by magnifying glasses*[49, Schem:XI.Fig:1.].

motion blur, which allows for longer exposure times. However, when imaging live specimen for longitudinal studies or to investigate dynamic processes, fixing the sample is not an option.

Imaging live specimen is crucial to answer many biological questions. Organisms do not live in static snapshots, nor are they confined to two dimensions. Any transmission based technique like bright-field microscopy is therefore intrinsically limited as they capture two-dimensional projections. A common volumetric imaging approach — imaging a sample sliced into thin sections — is not reconcilable with keeping the specimen alive. Thus imaging live specimen in three-dimensions turns out to be a challenging endeavor that has great potential to expand our understanding of life.

In recent years the limit of physical slicing or resorting to projection imaging has been addressed — even for live specimen. Currently the two most popular methods for volumetric live imaging are confocal microscopy and light sheet microscopy, also known as selective-plane illumination microscopy[53] (SPIM). These microscopes allow for optical sectioning. In confocal microscopy illumination is focused on a single point and a pinhole

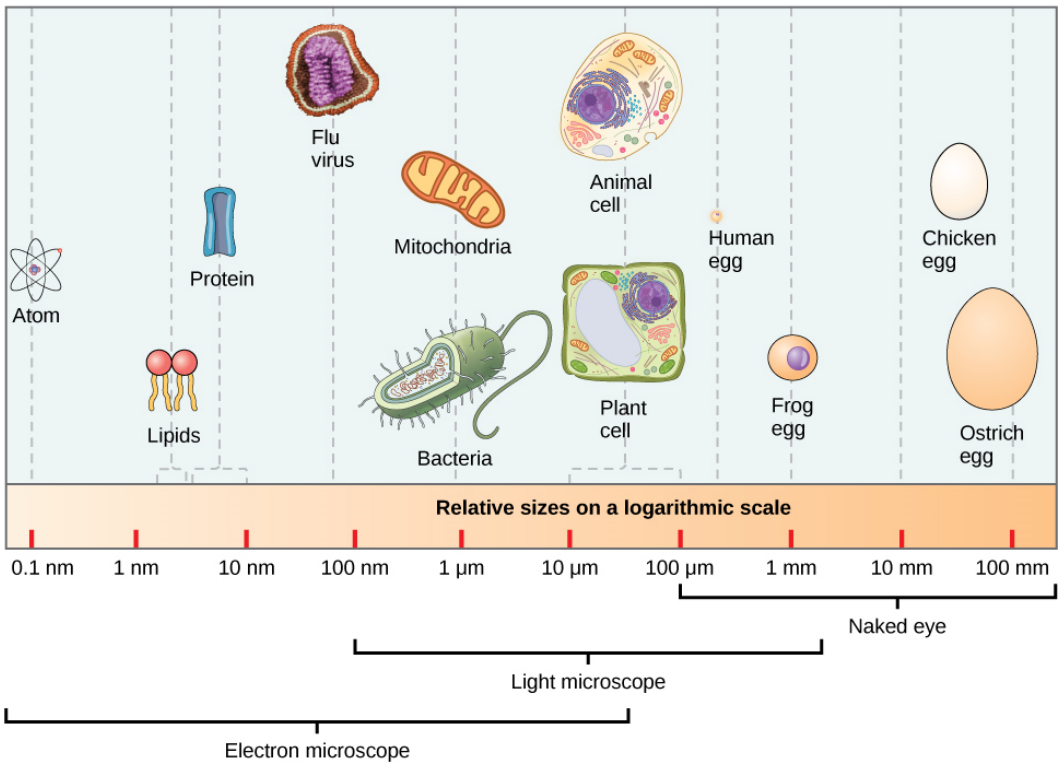


Figure 2.2: The naked eye is incapable of directly observing the inner workings of cells, but many biological structures can be resolved by a light microscope. Below the diffraction limit of light super resolution (light) microscopy and electron microscopy are used to resolve very small structures. Figure adapted from Fowler et al. [50, Figure 3.6].

in the detection path is used to filter out out-of-focus signal. Limiting the observed light to the focal plane — hence the name 'confocal' — improves the resolution at the cost of signal intensity. To image a plane or a volume the confocal microscope design necessitates to record a set of individual points. Scanning the specimen was first achieved by moving the objective lens[54], while modern laser SCM (LSCM) commonly use oscillating mirrors[55]. The scanning processes makes confocal microscopy inherently slow and the confocal design as discussed so far most suitable for fixed specimen.
Scanning an image can be sped up by parallelizing the point acquisition. Multi-point scanning has been implemented in spinning disk confocal microscopes; multiple pin holes on a spinning Nipkow/Petráň disk are captured in parallel with an image sensor — for example a charge-coupled device (CCD) camera — rather than sequentially with a photodetector such as a photomultiplier tube (PMT). A notable variant of the spinning disk confocal microscope is the microlens enhanced or dual spinning disk confocal microscope. In it a second disk with one microlens for each pin hole is added to focus the light into each pin hole which increases the signal-to-noise ratio[56]. This allows for imaging of some dynamic processes.

In SPIM instead of a point, a thin slice is illuminated by a light sheet and captured as a two-dimensional image without any lateral scanning which leads to a relatively high image acquisition speed. Scanning is only required axially for volumetric recordings. This revolutionary microscopy technique leads to dramatically lower photobleaching compared to confocal microscopy as demonstrated in figure 2.3. Phototoxicity, albeit a separate phenomenon, is also reduced as it, too, correlates with the energy load delivered to the specimen[57].

Other ingenious microscopy principles exist that unfortunately are unsuitable (in their current form) for volumetric imaging of highly dynamic processes in a large field of view (FOV) at significant imaging depth. These include Lattice light-sheet microscopy, total internal reflection microscopy (TIRFM), stimulated emission depletion (microscopy) (STED), two photon–excited fluorescence laser-scanning microscopy (2PLSM).

Lattice light-sheet microscopy is a further development of SPIM. The illumination is realized with two-dimensional optical lattices. "Optical lattices are periodic interference patterns in two or three dimensions created by the coherent superposition of a finite number of plane waves traveling in certain well-defined directions"[58]. Chen et al. discuss the limited imaging depth (20 µm to 100 µm) and propose the use of adaptive optics[59, 60] to

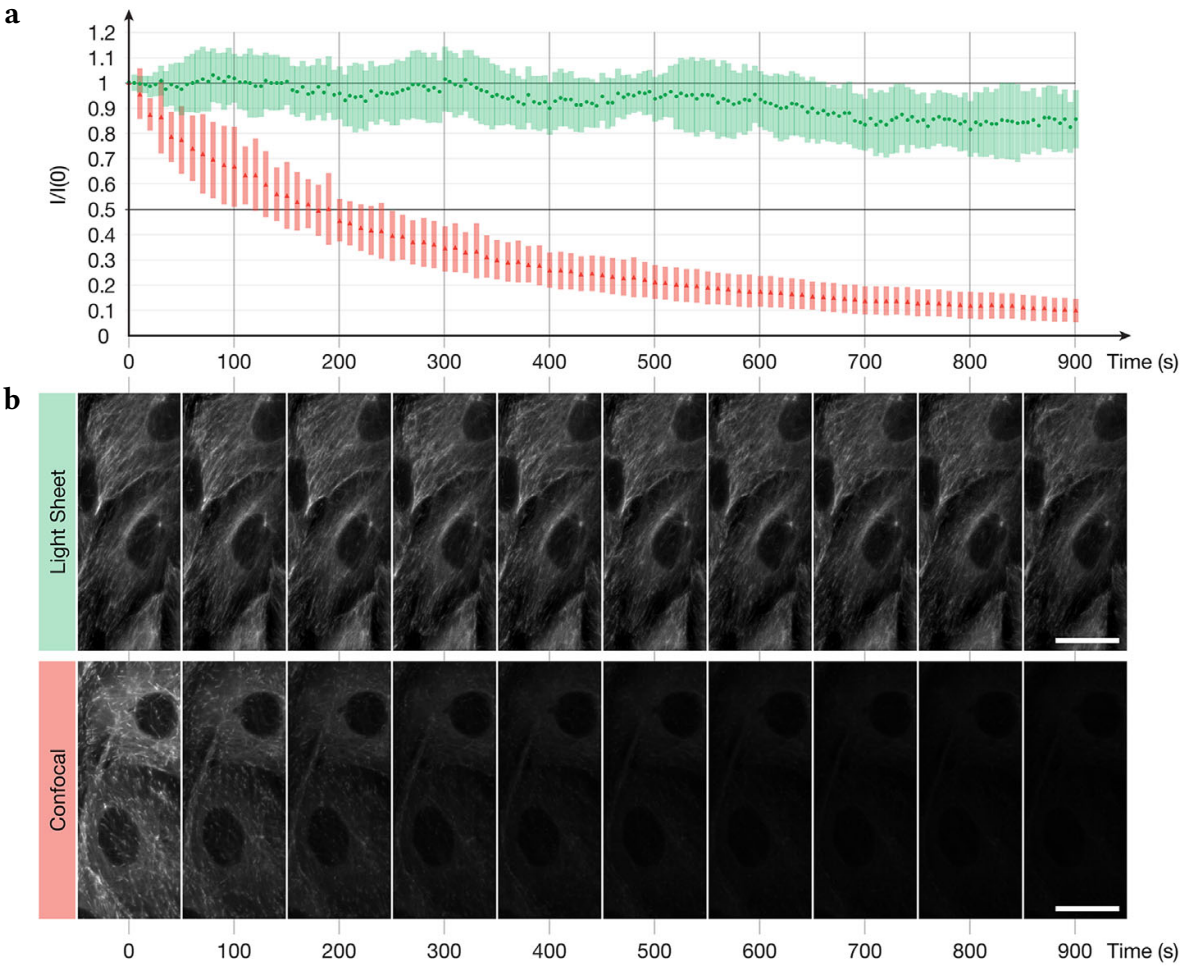


Figure 2.3: Photobleaching is much higher in (spinning disk) confocal microscopy compared to SPIM in otherwise comparable imaging setups. This manifests in the fading signal in the maximum projected volumes recorded by a spinning disk confocal microscope due to photobleaching. **a**: "Average intensity (I) normalized to intensity at time point 0 (I_0) of 10 small random regions of interest in different cells and planes. Intervals around mean indicate standard deviation."[57, Figure 1] **b**: "Maximum intensity projections of a 30×80mm region of interest. Scale bars: 20 mm."[57, Figure 1] Figure adapted from Icha et al. [57, Figure 1].

address this limitation in future work[58].

TIRFM, introduced by AMBROSE[61] in 1956, makes use of a peculiar physical phenomenon: the evanescent wave — an electromagnetic field at the surface of a medium in which a laser is totally internally reflected[62]. This electromagnetic field decays exponentially, which results in excitation of fluorophores in a a very thin optical slice at the interface, effectively eliminating out of focus signal[62].

STED is a super-resolution method for fluorescence microscopy that allows imaging beyond the diffraction limit by depleting fluorophores around the point of excitation prior to recording each image point[63]. This procedure slows down the point scanning image acquisition trading temporal for spatial resolution.

Rather than exciting fluorophores with a single laser as in scanning confocal microscopy (SCM), in 2PLSM two laser beams illuminate a sample point cooperatively[64]. As only the point in focus is effectively illuminated, out-of-focus signal is drastically reduced, which allows imaging deep into highly scattering tissue[64]. As point-scanning microscopes 2PLSM are relatively slow. However, Lavagnino et al. have successfully applied the principle of two-photon excitation to SPIM, which they call "Two-photon excitation selective plane illumination microscopy (2PE-SPIM)"[65].

Lemon et al. conclude that "there is no one-size-fits-all microscope."[66] The advantages and disadvantages of various microscopy methods are sometimes referred to as the 'pyramid of frustration in microscopy'[67], illustrated in figure 2.4. Compromises between FOV, sample health, SNR, spatial and temporal resolution have to be made when designing and operating a microscope. The FOV is not always included in this consideration[68, 66], but directly relates to the spatial resolution for microscopes capturing (a slice) of the FOV simultaneously or to the temporal resolution if scanning the FOV is required.

Despite the advances in image acquisition speed of the discussed microscopy methods so far, many biological processes occur on time scales so small that they present a persistent challenge for modern microscopy. Light-field microscopy[69] (LFM), introduced by Levoy et al. [69], is a new microscopy approach aiming at imaging large three-dimensional FOVs of sparse samples at unprecedented frame-rates, but has yet to unfold its potential. High-speed volumetric imaging with LFM is achieved by capturing volumetric information in a single camera frame as explained in section 2.2.

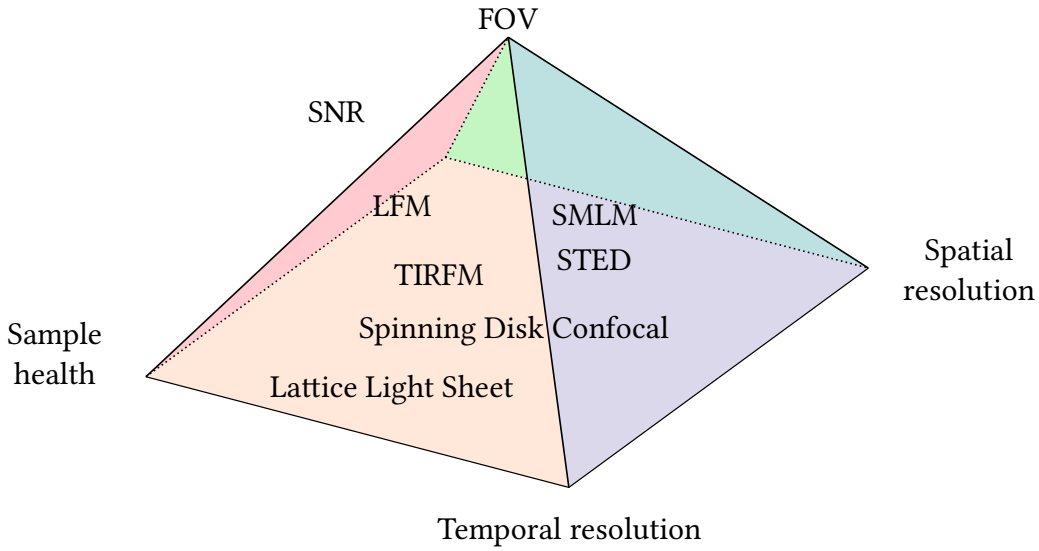


Figure 2.4: Different microscopy methods allow to realize different compromises between FOV, sample health, SNR, spatial and temporal resolution. Each (specific optical setup of a) microscope can be thought of as a point within this 'Pyramid of frustration'. Figure inspired by [67, Figure 1].

LFM can be used to image sparse, fluorescent or weakly scattering specimens[70]. Prevedel et al. [71] demonstrate LFM's capability to capture highly dynamic processes over a large FOV[71]. They deploy a classic deconvolution method — the Richardson-Lucy algorithm — to obtain a low resolution volumetric image from a captured light field with severe artifacts in the reconstructed focal plane. Stefanoiu et al. introduce an iterative aliasing-aware deconvolution scheme to remove these artifacts[72]. However, remaining drawbacks of Richardson-Lucy light field deconvolution (LFD) are the computational bottleneck and limited image fidelity. The latter can be mediated by fusing two light field views[73], which further increases the computational cost.

In N. Wagner*, F. Beuttenmueller* et al. [74] I deploy DL in conjunction with a new hybrid LFM/SPIM microscope design to mitigate these drawbacks. In this chapter I will introduce our method and detail how I designed, trained and evaluated a suitable NN to unite the advantages (and mitigate disadvantages) of LFM and SPIM.

My contributions to N. Wagner*, F. Beuttenmueller* et al. [74] were conducted in collaboration with Nils Wagner, Nils Norlin, Jakob Gierten, Juan Carlos Boffi, Joachim Wittbrodt, Martin Weigert, Lars Hufnagel, Robert Prevedel, and Anna Kreshuk[74]. We call our approach hybrid light-field light-sheet microscopy (HyLFM)[74]. The method was conceived by Anna Kreshuk, Lars Hufnagel and Robert Prevedel. The HyLFM optical setup is a combination of LFM and SPIM and was assembled and operated by Nils Wagner and Nils Norlin. They were assisted by Jakob Gierten, who produced transgenic animals under the guidance of Joachim Wittbrodt. Together with Anna Kreshuk and Martin Weigert, I conceived the NN architecture[74] — the HyLFM-Net that is the computational core of HyLFM bridging the capability gap between SPIM and LFM.
I performed all computational experiments in this project: I implemented and trained multiple HyLFM-Nets and evaluated their performance with input from Nils Wagner[74]. I collaborated with Nils Wagner on image preprocessing[74].
Juan Carlos Boffi performed the data analysis for the calcium imaging for the basis of which I trained and deployed a HyLFM-Net.

2.1 HyLFM Concept

In the following paragraphs I elaborate on the advantages and disadvantages of SPIM and LFM and why they form the basis of the novel HyLFM approach that I then introduce.

SPIM

While volumetric SPIM images benefit from a high resolution, they suffer from a low frame-rate as each plane of the volume is imaged sequentially. Imaging highly dynamic processes in this manner leads to distorted volumetric images, because the volume slices are captured at different points in time. This time distortion can be partially mitigated by slice-specific interpolation across time-frames — such an interpolation scheme reduces spatial and local temporal resolution — or by covering the depth of the specimen with fewer planes to speed up image acquisition — which reduces the spatial resolution axially and — due to wider planes — laterally, as well.

LFM

In LFM, on the other hand, information of a whole volume is recorded as a four-dimensional light field on a two-dimensional camera sensor allowing for a high frame rate. With state-of-the-art LFD a volumetric image of low spatial resolution can be reconstructed from a given light field image. One bottleneck for wider adoption of LFM is the high computation cost of LFD. Another is the insufficient reconstruction quality, which is aggravated by the fact that humans fail to qualitatively evaluate the fidelity of light field reconstructions. Note for example the lack of apparent depth information in the light field shown in figure 2.5 or in figure 2.15 how only the broad outline of the captured light field **a** can be visually mapped to the reconstructed volumes **b**–**d**.

SPIM + LFM = HyLFM

The goal of HyLFM: Profit from the benefits of SPIM and LFM by combining them in a single microscope and improve the imaging of a highly dynamic specimen through DL in a manner reminiscent of image restoration with DL such as CARE. The light-fields, acquired

at a high frame-rate, are processed by a HyLFM-Net — potentially in real-time — to yield whole volume video reconstructions at SPIM like high quality.

To train a HyLFM-Net capable of light-field deconvolution, a dataset with pairs of light-field and light-sheet images is used, such that the individual reconstructions resemble SPIM volumes as closely as possible. Due to the integrated optical paths of SPIM and LFM within a HyLFM microscope, suitable training data can be recorded directly without lengthy data curation, manual annotations or simulations.

Light fields are recorded through the LFM path while the SPIM path captures high quality images that serve as ground truth — the desired HyLFM output. For highly dynamic samples one light field is recorded for each two-dimensional SPIM plane. As the illuminated SPIM plane is moved through the specimen over time the HyLFM network can still learn to reconstruct the three-dimensional volume. If static samples are available, pairing a light field with a whole SPIM volume allows for more efficient training — updates for all network weights can be computed simultaneously as the training loss, and thus gradients, can be computed across the whole prediction volume.

HyLFM also serves as a setup for continuous validation and online refinement of a pretrained network by recording SPIM planes throughout the image acquisition of dynamic samples. For each time frame the volumetric HyLFM reconstruction, based on the light field, can be compared in a single slice to the acquired SPIM plane. By shifting the position of the recorded SPIM plane over time (as in regular SPIM imaging), the whole reconstructed FOV can be validated. This allows to establish and uphold trust in HyLFM-Net reconstructions.

2.2 Light-Field Deconvolution

To understand how we can reconstruct a volume of fluorescent emitters from an LFM image, let us take a look at what the captured light field encodes. While a conventional microscope captures a single orthographic view of the specimen, a light-field microscope records multiple perspectives through a microlens array (MLA) (also known as lenslets)[69]. Like stereopsis that we obtain from our binocular vision, these angled views provide depth information.

The direct approach of capturing an array of images with each image focused by

one lenslet — placing the MLA in the Fourier plane of the objective — implies that the angular resolution depends on the number of views, thus number of lenslets, while the spatial resolution of each of these angular views relates to the overall spatial resolution. Recently Guo et al. have introduced this as "Fourier LFM"[75]. However, due to challenges in manufacturing an array of high quality lenslets, it is often preferred to place the MLA in an image plane[69].

With the MLA in an image plane the dependencies of spatial and angular resolution are switched: the spatial resolution depends on the number of lenslets and the angular resolution corresponds to the resolution of each lenslet image[69], in other words, each pixel in a lenslet image corresponds to a different view/angle, not each whole lenslet image. This optical setup, capturing transposed light field images, is typically preferred for its better spatial resolution (laterally), compromising angular resolution[69], which is anyway limited by the half-angle θ as I discuss in section 2.4.6.

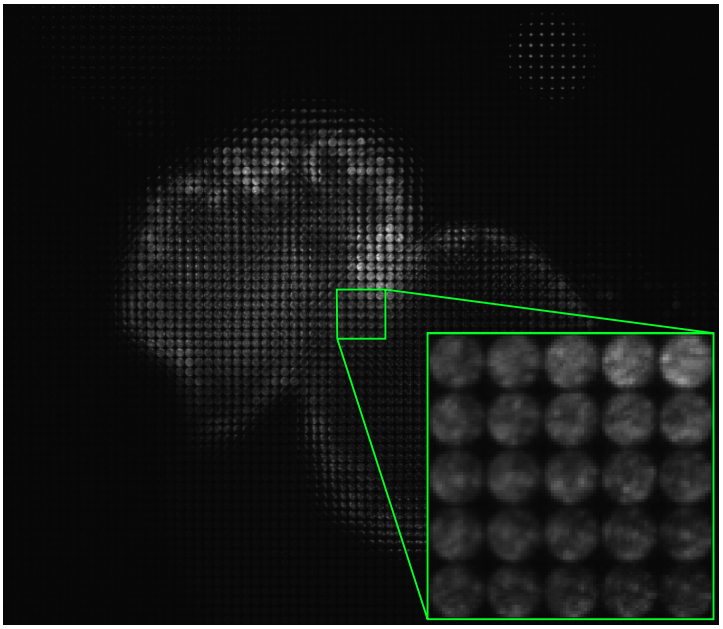
The relation of a captured transposed light field and its non-transposed equivalent is illustrated in figure 2.5. Note that in this thesis the *recorded transposed light field images* are usually referred to simply as *light fields* as the difference is not important conceptually.

A captured light field image can be parameterized as shown in figure 2.6.

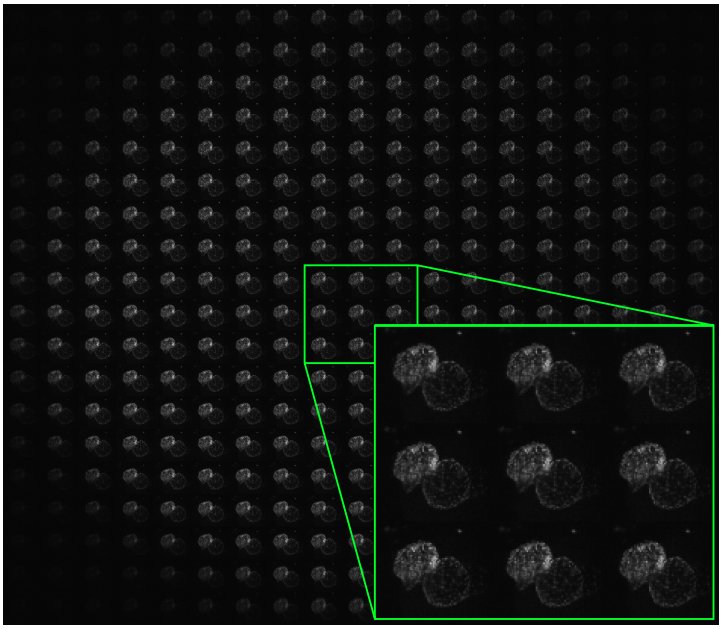
Transposed or not — a light-field microscope captures angled views, but ultimately what we are interested in is a volumetric image. To obtain it, we need to invert the noisy measurement that projects the full five-dimensional light field — the plenoptic function $L(x, y, z, \theta, \phi)$ — to the recorded four-dimensional, scalar light field $L^*(u, v, s, t)$ as illustrated in figure 2.6.

Due to this dimensionality reduction the inverse is ill defined, meaning there are more than one possible solution. To overcome this predicament a deconvolution algorithm has to be applied to find a plausible solution. The iterative Richardson-Lucy algorithm can be used to reconstruct a convolved image if the point-spread function is known[76, 77]. Prevedel et al. [71] demonstrate applying this algorithm to LFM (in this thesis referred to as LFD) to analyze neural activity in zebrafish[71]. Three-dimensional deconvolution of LFM images can also be seen as limited angle cone-beam tomography [78, 69].

Conventional deconvolution algorithms in the domain of image restoration have recently been outperformed by DL approaches that exploit learned, data dependent biases and



(a) Transposed light-field image as recorded: Each lenslet subimage represents a position in object space and consists of $N_{\text{num}}^2 = 19^2$ pixels, each corresponding to a different direction of incident rays at that location.



(b) Rearrangement of the recorded transposed light-field in (a) to resemble a non-transposed light-field image: Each of the $N_{\text{num}}^2 = 19^2$ lenslet images represents an angular view.

Figure 2.5: Recorded transposed light-field (a) and its corresponding non-transposed light-field (b). *Note: best viewed on screen without interpolation by the PDF viewer.*

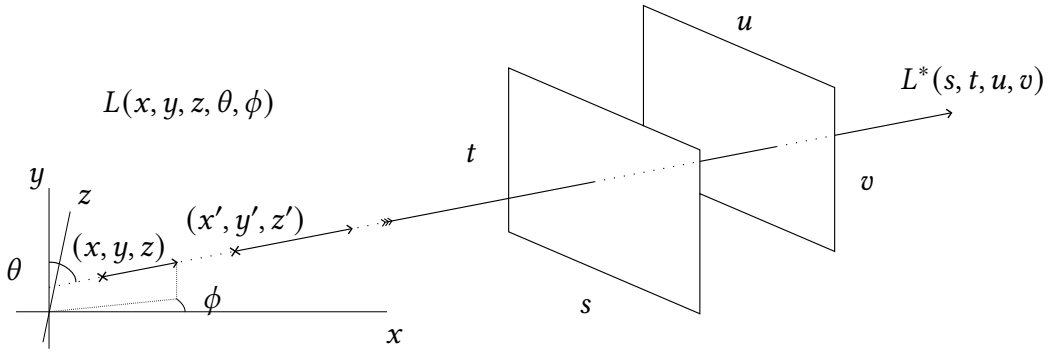


Figure 2.6: The five-dimensional light field or plenoptic function $L(x, y, z, \theta, \phi)$ is projected to the four-dimensional scalar light field $L^*(s, t, u, v)$ parameterized by a ray through two planes. This parameterization conceptually maps to the integration over angled rays going through a microlens in the discretized (t, s)-MLA-plane and hitting a camera pixel of a microlens subimage in the discretized (u, v)-sensor-plane.

local image context[11]. This hints at the viability of a DL approach to the light field deconvolution problem. However, a key challenge for AI based image enhancing/reconstruction methods — especially in a scientific context — is to proof their validity, as these methods have the potential to hallucinate features that are not present in the sample. Generated outputs might look plausible, but without quantitative evaluation these looks might be deceiving. To address this not only high-quality, trustworthy ground truth is needed for training faithful and robust models, but also validation schemes are needed to verify results in later applications. By designing a NN for real-time light field deconvolutions I wanted to ensure that the proposed HyLFM approach can utilize its hybrid nature throughout imaging operations to maximum effect. With the ongoing SPIM volume scanning immediately

2.3 Approach

An overview of the functional principle of HyLFM is given in figure 2.7 and a rendering of a final result is shown in figure 2.8. The following sections 2.3.1 to 2.3.4 elaborate on different aspects of the HyLFM approach, guiding the reader from concepts to experiments, from results (section 2.4) to discussion (section 2.5).

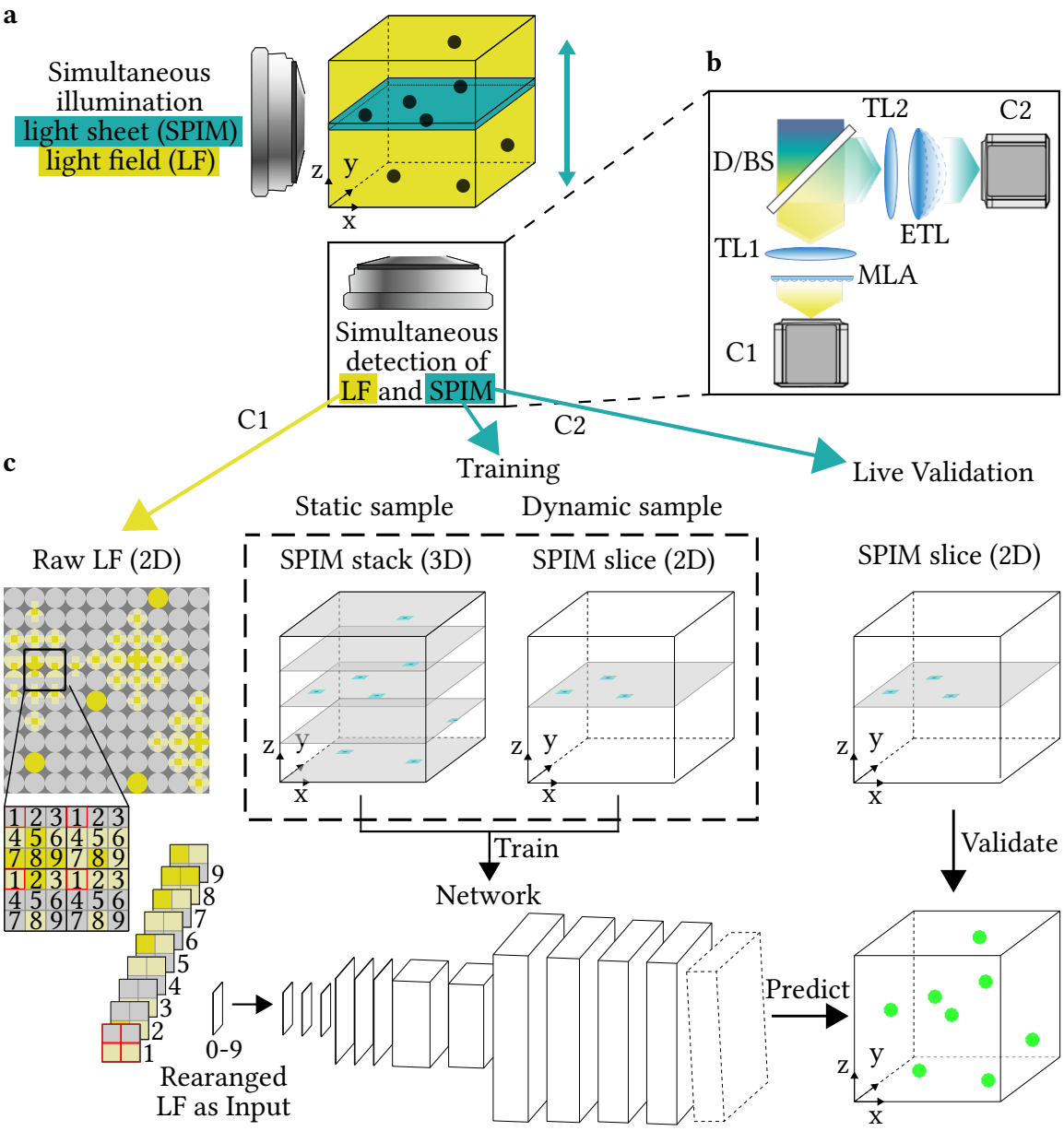


Figure 2.7: "**a**, Microscope geometry showing simultaneous imaging via and light-field modalities. **b**, Schematic view of the and light-field detection paths. (D-)BS, (dichroic) beamsplitter used for (dual-) or single-color imaging; TL, tube lens; MLA, microlens array; C, sCMOS camera. **c**, HyLFM-Net image reconstruction pipeline. The raw light-field image serves as input to a reconstruction , where the lenslet pixels are rearranged channelwise. The last layer encodes the affine transform between the and spaces. The can be trained either on high-resolution light-sheet volumes for static samples, or on high-resolution light-sheet planes for dynamic samples. The network output can additionally be validated by sweeping light-sheet planes."[74] Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Fig. 1]

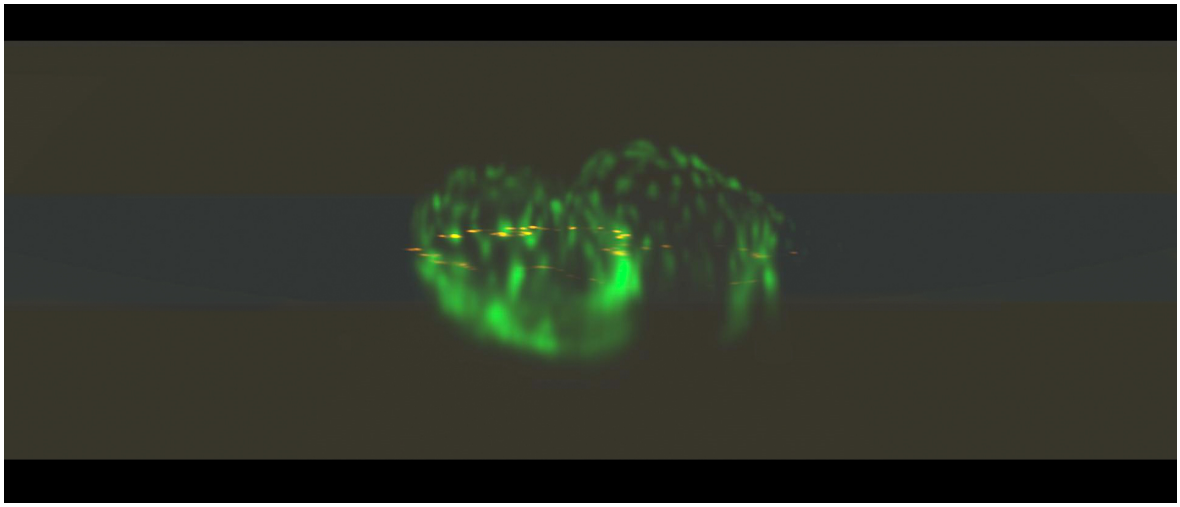


Figure 2.8: Rendered frame of a beating medaka heart imaged with HyLFM. Single frame of a rendering of a beating medaka heart (*Oryzias latipes* 8 dpf) imaged at 40 Hz with HyLFM in dual color mode. In this single frame the whole heart (in green) was captured as a light field at the same time as the single SPIM light sheet image (illumination in blue, detection in orange) was taken. The yellow background represents the light field illumination. The heart was reconstructed by a HyLFM-Net-stat refined on a preceding five minute time laps. The network refinement process is discussed in section 2.4.4 and illustrated in figures 2.17 and 2.18.

2.3.1 Optical Setup

To image the same specimen the two types of microscope — LFM and SPIM — share the illumination and detection objectives, while all other optical components are separated[74]. HyLFM especially benefits from two-color labeling of the same structure to simultaneously acquire light fields and light sheets to avoid out-of-focus signal in the SPIM path by using independent illuminations. However, alternating between the imaging modalities can be used to operate the HyLFM with a single color. N. Wagner*, F. Beuttenmueller* et al. use two identical lasers to keep the same dual beam path setup for imaging in dual and single color mode[74]. These separate beam paths are joint/split via dichroic mirrors in dual color mode or beamsplitters when imaging with a single color[74]. In the single color setup two excitation lasers of the same wavelength are deployed to keep the setup analog to dual color mode[74]. The detailed optical setup is laid out in figure 2.9.

2.3.2 HyLFM-Net architecture

The architecture of the proposed, novel HyLFM network, that I designed with input from Anna Kreshuk and Martin Weigert, specifically caters to the reconstruction of three-dimensional predictions from the four-dimensional LFM images.

The four-dimensional scalar light field images are captured in two dimensions. An example is shown in figure 2.5a. These images are resized if necessary such that each lenselet subimage takes up exactly n_{num}^2 pixels (on the uv-plane in figure 2.6). As n_{num} is given by the optical setup, and thus constant for one HyLFM microscope, each acquired light field image is rearranged to form a multichannel image with one channel for each lenselet subimage position corresponding to a viewing angle. This rearranging ensures that pixels in this multichannel image and locations in and directions from the object share the same neighborhood relation. A step in this rearranged pixel space directly corresponds to a lateral step in the object space. Depending on the axial object position more a less rearranged image pixels will be affected by signal from that position. The affected circle of image pixels is symmetrically centered around the lateral position. These neighborhood relations can be represented with convolutional kernels in CNNs. Kernel size and number of layers (network depth) yield the network's FOV, which needs to be compatible with the extent of the axial dimension expected to be recovered by the network. As the "angle/perspective

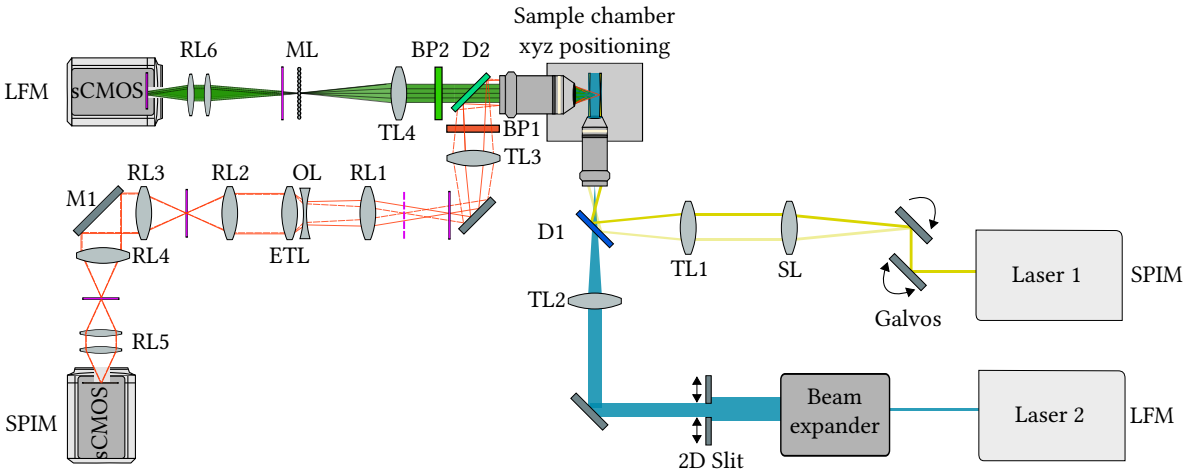


Figure 2.9: HyLFM Optical Setup. Combined LFM and SPIM setup with the main microscope components illustrated. "The sample is illuminated through a single illumination objective with two excitation beam paths (ocra, light-sheet illumination[color added] and blue, light-field selective volume illumination[color added])"[74, Fig. 2]. These illumination beams are combined by the dichroic mirror D1[74, Fig. 2]. An orthogonally oriented detection objective detects the fluorescence, which is optically separated into two detection paths by the dichroic mirror D2[74, Fig. 2]. "For single color imaging, the dichroic mirrors D1 and D2 are replaced by beamsplitters."[74, Fig. 2] "Bandpass filters BP1 and BP2 are placed in front of tube lenses TL3 and TL4 for the respective detection path. [In] the light field detection path (green[color added]), the tube lens TL4 focuses on the microlens array ML[. T]he image plane ([...]magenta[color added])[,] displaced by one microlens['s] focal length[,] is relayed by the 1-1 relay lens system RL6 to an image plane coinciding with the camera sensor ([...]magenta[color added]). [... To image the two axially displaced objective focal planes in] the light-sheet detection path [in a common image plane at the sensor (magenta, dotted and solid[color added]), the] combination of several relay lenses RL1 to RL4 [and] the 1:1 macro lens RL5 [are used] together with a lens pair consisting of the offset lens OL, and the ETL. [...] The refocusing is achieved by [applying] different currents [on] the ETL. The mirror M1 is placed at a Fourier plane, such that the FOV of the light sheet path can be laterally aligned to fit the light field detection FOV."[74, Fig. 2] Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Fig. 2].

view dimensions" are treated as channels highly optimized two-dimensional and three-dimensional convolutional layers provided by the PyTorch library[47] can be utilized to implement the HyLFM,

2.3.3 Volume Registration

Aligning LFM and SPIM paths is challenging. It should be expected that (reconstructed) light field volumes and light sheet stacks do not align perfectly, but that an affine transformation exists which — if known — can be applied to remedy any misalignment to a satisfactory degree.

To solve this image registration problem N. Wagner*, F. Beuttenmueller* et al. use the conventional Richardson-Lucy light field deconvolution (LFD) to reconstruct a light field of a sample of sparse fluorescent fiducials ('beads'), which is also recorded with the SPIM path[74]. Registering the obtained volumes with tools such as the Multiview Reconstruction Plugin for Fiji[79][80] allows to determine a compensating affine transformation. Repeated imaging of beads has shown that the alignment of the optical setup is stable enough to reuse a measured affine transform over multiple days of microscope operation.

To address the registration problem in the HyLFM-Net training pipeline I transformed the SPIM stacks to the LFD reconstruction space for static samples with a predetermined affine transformation. For (highly) dynamic samples only a single light sheet can be captured in each time frame. Transforming it can lead to interpolation artifacts, which is why I decided to apply the affine transformation to the (relevant section of the) predicted volume instead. The affine transformation, that can be viewed as part of the model architecture as described in figure 2.10, has to be differentiable to allow for training of the HyLFM-Net through backpropagation. This is achieved by generating a grid of coordinates from which the transformed output is then sampled from (with bilinear interpolation). This approach has been popularized by Jaderberg et al. [81]. For a fixed affine transformation used in dynamic HyLFM-Net training the grid needs to be generated only once when initializing the training procedure.

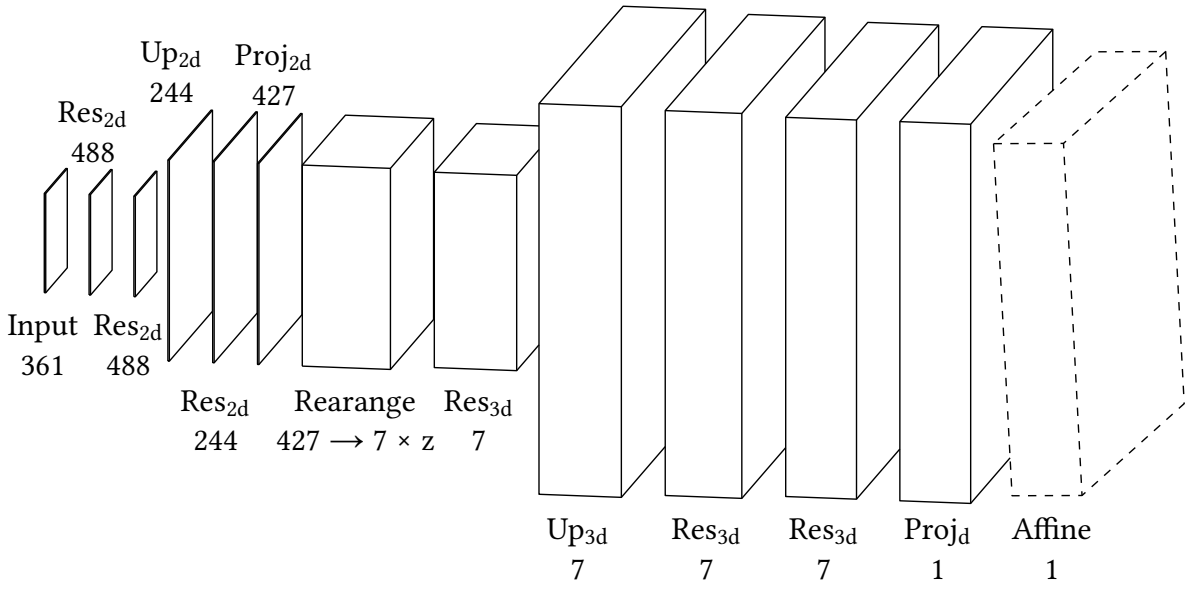


Figure 2.10: Example of HyLFM-Net architecture used in "Deep learning-enhanced light-field imaging with continuous validation"[74] to reconstruct fluorescent beads, medaka larvae heart nuclei, and zebrafish larvae neural activity. $Res_{2d/3d}$: residual blocks with $2d$ or $3d$ convolutions with kernel size $(3\times)3 \times 3$. Residual blocks contain an additional projection layer with a $(1\times)1 \times 1$ convolution if the number of input channels is different from the number of output channels. $Up_{2d/3d}$: transposed convolution layers with kernel size $(3\times)2\times2$ and stride $(1\times)2 \times 2$. $Proj_{2d/3d}$: projection layers with $(1\times)1 \times 1$ convolutions. The numbers always correspond to the number of channels. With 19×19 pixel lenslets ($n_{\text{num}} = 19$) the rearranged light-field input image has $n_{\text{num}}^2 = 19^2 = 361$ channels. The affine transformation layer at the end is only part of the network when training on single SPIM planes acquired from dynamic samples; otherwise, the inverse affine transformation is applied to the SPIM volume to align static samples once to the LFM reconstruction space and avoid redundant computations during training. In inference mode, an affine transformation might be used in post-processing to yield a prediction aligned to the SPIM FOV. Volume registration is elaborated on in section 2.3.3. In analogy to the popular U-Net architecture[31] I refer to all layers before the Rearrange layer as the encoder and all layers after the Rearrange layer as the decoder. The line of thought being that the light field is being encoded into a high dimensional embedding space (with a high number of feature channels) from which the volumetric reconstruction is decoded. Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Extended Data Fig. 1]

2.3.4 HyLFM-Net Training Prerequisites

I trained all HyLFM-Nets in a fully supervised manner using the registered (see section 2.3.3) SPIM volumes for training a HyLFM-Net-stat and frames with single SPIM slices for training a HyLFM-Net-dyn. For HyLFM-Net-dyn the registration is done with a (differentiable) affine transformation of the predictions as motivated in section 2.3.3.

A HyLFM-Net is trained to predict the relative fluorescence intensity of each voxel in the FOV. Using SPIM as ground truth this assumes correlating irradiance measurements for the SPIM and LFM paths of the HyLFM microscope.

In single color mode this can only be the case if the fluorophore's dynamics are not undersampled by the temporal resolution. In dual color mode the stoichiometric relation of fluorophores can be fixed biologically in transgenic organisms by dual-labeling the same protein or labeling proteins whose expressions have a constant relation throughout the sample or more specifically the FOV. If dual-labeling a HyLFM sample by staining, experimentalists should take precautions to permeate the sample evenly with both dyes, potentially using a single compound with dual-fluorescent properties[82, 83] instead.

2.4 Results

This section presents the output of my contributions to the HyLFM project. All code that I wrote to perform the described computational experiments and evaluations of this project are available at github.com/kreshuklab/hylfm-net. To support further development of HyLFM and to ensure the reproducibility of my results, I added instructions on how to execute the code to perform HyLFM-Net training and evaluation. With permission of my collaborators, I shared the data used in this project in the BioStudies database (part of the European Life-Science Infrastructure (ELIXIR)[84, Glossary] infrastructure hosted by EMBL-EBI at `https://www.ebi.ac.uk/biostudies/`), see table A.1.

2.4.1 Experimental Setup

The optical setup described in section 2.3.1 was used to record all HyLFM data used in this project. A water dipping objective (Olympus UMPLFLN 20XW) with 0.5NA is used for

detection[74]. A Nikon MXA20696 tube lens is used to yield an effective magnification of $M = 22.22$, see table A.2. Together with the MLA "RPC photonics microlens array s125 f25"[74] with a pitch $d_{\text{MLA-pitch}} = 125\,\mu\text{m}$ and the sCMOS camera Andor Zyla the setup is designed to encompass the required FOV of $350\,\mu\text{m} \times 300\,\mu\text{m} \times 150\,\mu\text{m}$ for the medaka heart (*Oryzias latipes* 8 dpf/1 to 3 days after hatching) and $350\,\mu\text{m} \times 280\,\mu\text{m} \times 120\,\mu\text{m}$ for the larval zebrafish (*Danio rerio* 5 dpf) brain [74].

To record static medaka hearts "*myl7::H2B-eGFP*, *myl7::H2A-mCherry* transgenic medaka hatchlings were sedated with $150\,\text{mg}\,\text{L}^{-1}$ tricaine and the heart was pharmacologically arrested with 40 mmol 2,3-butanedione 2-monoxime."[74]

Medaka hearts were imaged in dual-color mode, using mCherry in the SPIM optical path and enhanced GFP (eGFP) to capture the light field[74]. To track neural activity in larval zebrafish brains, calcium imaging with GCaMP6s was performed in single-color mode[74].

2.4.2 Characteristics of the Experimental Setup

I approximate the wavelength λ of light captured in the light field considering eGFP and GCaMP6s were used:

$$\lambda \approx 510\,\text{nm}. \tag{2.1}$$

According to Hale et al.[85] the refractive index of water for this wavelength is

$$n \approx 1.33\,. \tag{2.2}$$

An objective's ability to capture angled light rays is characterized by the numerical aperture (NA).

$$\text{NA} = n \sin\theta \tag{2.3}$$

It depends on the refractiveindex n of the medium in which the objective is placed in and the half-angle θ, the maximum angle of the captured light cone. Rearranging equation (2.3) and using $n_{water} \approx 1.33\lambda$[85] and NA = 0.5 for the relevant wavelengths of eGFP and GCaMP6s

the half-angle θ for this experimental setup can be calculated to be

$$\theta = \sin^{-1} \frac{\mathrm{NA}}{n} \approx 22.08^{\circ}. \tag{2.4}$$

2.4.3 HyLFM-Net Training

Due to less efficient training on single ground truth slices compared to ground truth volumes, I chose HyLFM-Net-dyn to be slightly smaller — having fewer model weights. The number of channels for each layer of the HyLFM-Net architecture is specified in table 2.1.

The MSE (also known as squared L2 norm) — a standard choice for many regression problems — proved to be a suitable loss function for datasets with samples that were not too sparse; learning to predict sparse, sub-diffraction beads typically resulted in predicting empty volumes when deploying the MSE as a loss function. I found that a smooth mean absolute error (MAE) (also known as smooth L1 loss) adjusted by an additional downweighing of low intensity voxels with a decaying weight lead to stable training and high multi-scale-structural similarity index measure (MS-SSIM) validation values. The smoothness (substituting MAE with a quadratic function below a fixed threshold) makes the loss function 'more forgiving' of minor discrepancies that otherwise may add up over the whole prediction volume, leading to instability during training. Downweighing non-peak SPIM voxels accounts for the imbalance of background to peak-signal voxels in the sparse samples. I also found that additionally minimizing the MS-SSIM directly can further aid to converge to a high quality solution. During training I used the Adam optimizer with "the learning rate set between 1.0×10^{-5} and 3.0×10^{-4}"[74].

To find performant sets of hyperparameters I employed Bayesian search by running a Weights & Biases (W&B)[86] "Sweep"[86]. The optimized hyperparameters included the initial learning rate, learning rate scheduling, network depth (number of model architecture layers), number of feature channels of specific layers (see figure 2.10), as well as the weighting threshold of the weighted, smooth L1 loss described above. I logged and compared training runs using W&B[86].

HyLFM-Net-beads layer	network architecture c	z [px]	image dimensions y [px]	x [px]
input	361	1	49	74
Res_{2d}	976	1	49	74
Res_{2d}	976	1	49	74
Up_{2d}	488	1	98	148
Res_{2d}	488	1	98	148
Up_{2d}	244	1	196	296
Res_{2d}	244	1	196	296
$Proj_{2d}$	441	1	196	296
Rearrange	7	63	196	296
Res_{3d}	7	59	192	292
Up_{3d}	7	59	384	584
Res_{3d}	7	55	380	580
Res_{3d}	7	51	376	576
$Proj_{3d}$	1	51	376	576

(a) Dimensions of HyLFM-Net-beads trained on fluorescent beads.

HyLFM-Net-stat layer	network architecture c	z [px]	image dimensions y [px]	x [px]
input	361	1	65	75
Res_{2d}	768	1	65	75
Res_{2d}	768	1	65	75
Up_{2d}	512	1	130	150
Res_{2d}	256	1	130	150
Res_{2d}	256	1	130	150
$Proj_{2d}$	128	1	130	150
Rearrange	64	61	130	150
Res_{3d}	32	57	126	146
Up_{3d}	8	57	252	292
Res_{3d}	8	53	248	288
Res_{3d}	8	49	244	284
$Proj_{3d}$	1	49	244	284

(b) Dimensions of HyLFM-Net-stat trained on arrested hatchling medaka hearts.

HyLFM-Net-dyn layer	network architecture c	z [px]	image dimensions y [px]	x [px]
input	361	1	65	75
Res_{2d}	488	1	65	75
Res_{2d}	488	1	65	75
Up_{2d}	244	1	130	150
Res_{2d}	244	1	130	150
$Proj_{2d}$	427	1	130	150
Rearrange	7	61	130	150
Res_{3d}	7	57	126	146
Up_{3d}	7	57	252	292
Res_{3d}	7	53	248	288
Res_{3d}	7	49	244	284
$Proj_{3d}$	1	49	244	284
Affine Trf.	1	1	244	284

(c) Dimensions of HyLFM-Net-dyn trained on beating hatchling medaka hearts.

HyLFM-Net-brain layer	network architecture c	z [px]	image dimensions y [px]	x [px]
input	361	1	70	83
Res_{2d}	488	1	70	83
Res_{2d}	488	1	70	83
Up_{2d}	244	1	140	166
Res_{2d}	244	1	140	166
$Proj_{2d}$	427	1	140	166
Rearrange	7	61	140	166
Res_{3d}	7	57	136	162
Up_{3d}	7	57	272	324
Res_{3d}	7	53	268	320
Res_{3d}	7	49	264	316
$Proj_{3d}$	1	49	264	316
Affine Trf.	1	1	244	284

(d) Dimensions of HyLFM-Net-brain trained on calcium imaging of larval zebrafish brains.

Table 2.1: Architecture dimensions for different HyLFM-Nets instances that I trained following the NN layout described in figure 2.10. Table adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Supplementary Table 1].

2.4.4 HyLFM Reconstructions

To evaluate reconstruction fidelity quantitatively I trained HyLFM-Net-beadss (see figure 2.10 and table 2.1) on 137 261.2 $\mu m^3 \times 400.2\,\mu m^3 \times 100\,\mu m^3$ SPIM volumes of fluorescent sub-diffraction-sized beads for 26.5 h. I found that HyLFM-Net-beads reconstructed volumetric images of fluorescent beads with a uniform and high lateral (7.1(13) µm) and axial resolution (7.1(13) µm). It performs substantially better than LFD — not suffering from artifacts as shown in more detail in figure 2.12 — and on par with LFD + CARE. This evaluation is based on 4966 beads and detailed in figure 2.11. To complete the evaluation of image fidelity for beads I evaluated the precision and recall of reconstructed beads to SPIM (figure 2.13). This shows a surprising discrepancy between the performance of HyLFM-Net and LFD + CARE: Even though the apparent resolution is on par, HyLFM-Net locates the fiducials more reliably. This hints at an information loss during LFD and shows that reconstructing end-to-end with HyLFM-Net is superior to enhancing LFD reconstructions with CARE.

To evaluate the applicability and image fidelity of HyLFM on biological samples my collaborators imaged arrested and beating medaka hearts (*Oryzias latipes* 8 dpf/1 to 3 days after hatching). I trained and evaluated HyLFM-Net-stat/HyLFM-Net-dyn on these static/-dynamic heart recordings, which is summarized in figure 2.14 and figure 2.15 respectively. As with the bead reconstructions, HyLFM-Net yields visually impressive results on par with LFD + CARE, outperforming LFD. The high image fidelity is confirmed and quantified by MS-SSIM and peak signal-to-noise ration (PSNR) measures plotted in figure 2.14 **e**.

Training a HyLFM-Net with complete ground truth volumes is more efficient than training on axial slices only, as computing a loss only for a thin section requires processing the full input light field nonetheless, while gradients (updates to the NN) only apply to a section of the network. Due to the benefits of LFM and HyLFM highly dynamic samples are the desired application, though. Therefore it is important to be able to train a HyLFM-Net merely on single SPIM as well for applications in which volumetric ground truth is difficult to obtain. To demonstrate the feasibility of training on individual slices acquired from a highly dynamic sample I trained a HyLFM-Net-dyn (see figure 2.10 and table 2.1) on 13 747 339.0 $\mu m^2 \times 394.6\,\mu m^2$ and 5925 277.9 $\mu m^2 \times 283.5\,\mu m^2$ SPIM planes with corresponding light field images for a total of 68 h on a single NVIDIA GeForce RTX 2080Ti GPU. In figures 2.14 and 2.15 I compare the reconstruction performance to a HyLFM-Net-stat, the (Richardson-

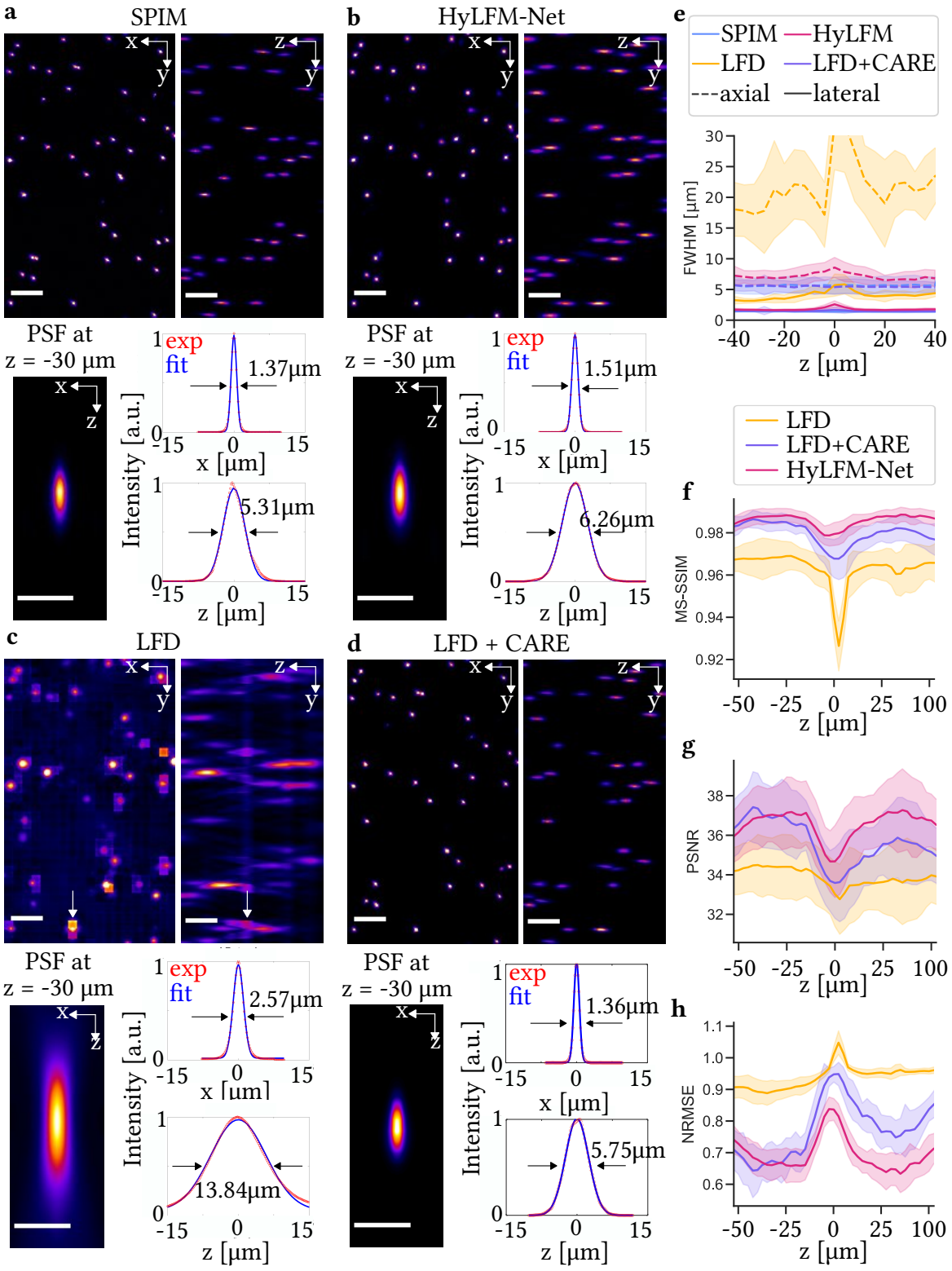


Figure 2.11: Analysis of HyLFM resolution using sub-diffraction-sized, fluorescent beads. Caption on next page. Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Fig. 2]

Figure 2.11: "Evaluation of HyLFM-Net's imaging performance on sub-diffraction-sized, fluorescent beads. **a**, A 3D SPIM stack (max intensity projections shown) and the corresponding raw light-field image. **b**, Same volume as in **a** after reconstruction by a HyLFM-Net trained on other SPIM/light-field pairs. **c**, Conventional reconstruction based on Richardson-Lucy type LFD. Exemplary artifacts are highlighted by white arrows, which are zoomed in on in figure 2.12. **d**, LFD reconstruction improved by deep learning-based image restoration (LFD + CARE). Images are representative of n = 28 samples with 51 z slices each. AU, arbitrary units. **e**, Lateral and axial resolution as a function of imaging depth for SPIM, HyLFM-Net, LFD and LFD + CARE, respectively, averaging measurements over 17,495 beads. **f**–**h**, MS-SSIM (**f**), PSNR (**g**) and NMRSE (**h**) image quality metrics across imaging volume comparing HyLFM-Net, LFD and LFD + CARE with the SPIM ground truth. Shadows in **f**–**h** denote standard deviation, inferred from all beads (n = 6,716) in the dataset at the same z position. Scale bars in **a**–**d** are 20 µm in whole FOV (top row) and 10 µm in PSF close-up (bottom row). PSFPSF close-ups are computed by averaging over ten beads. Experimental data in **a**–**d** are shown in red and fitting curves in blue. **i**, Precision and recall measurements averaged over all volumes, such that each point represents a threshold, inferred from 28 samples with 6,716 target beads in total."[74]

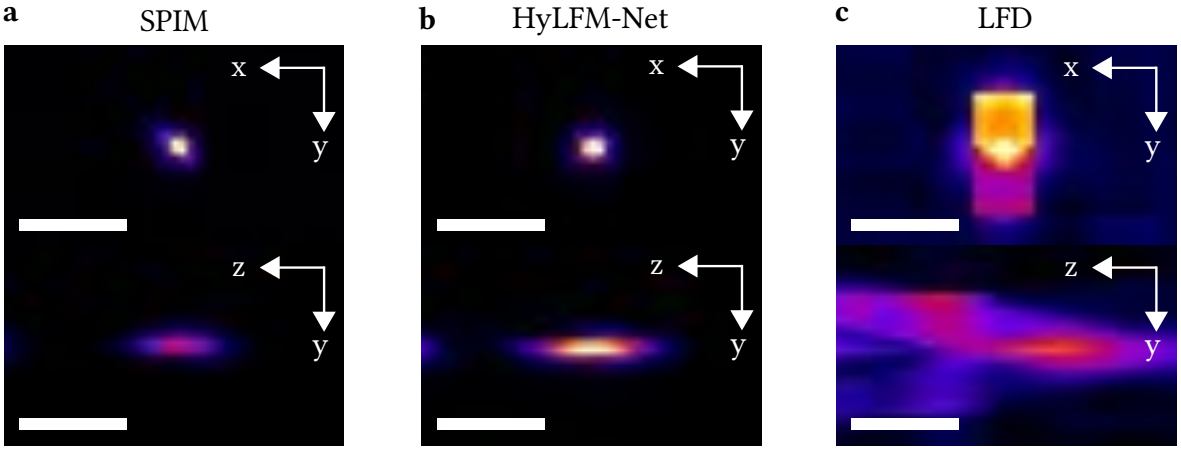
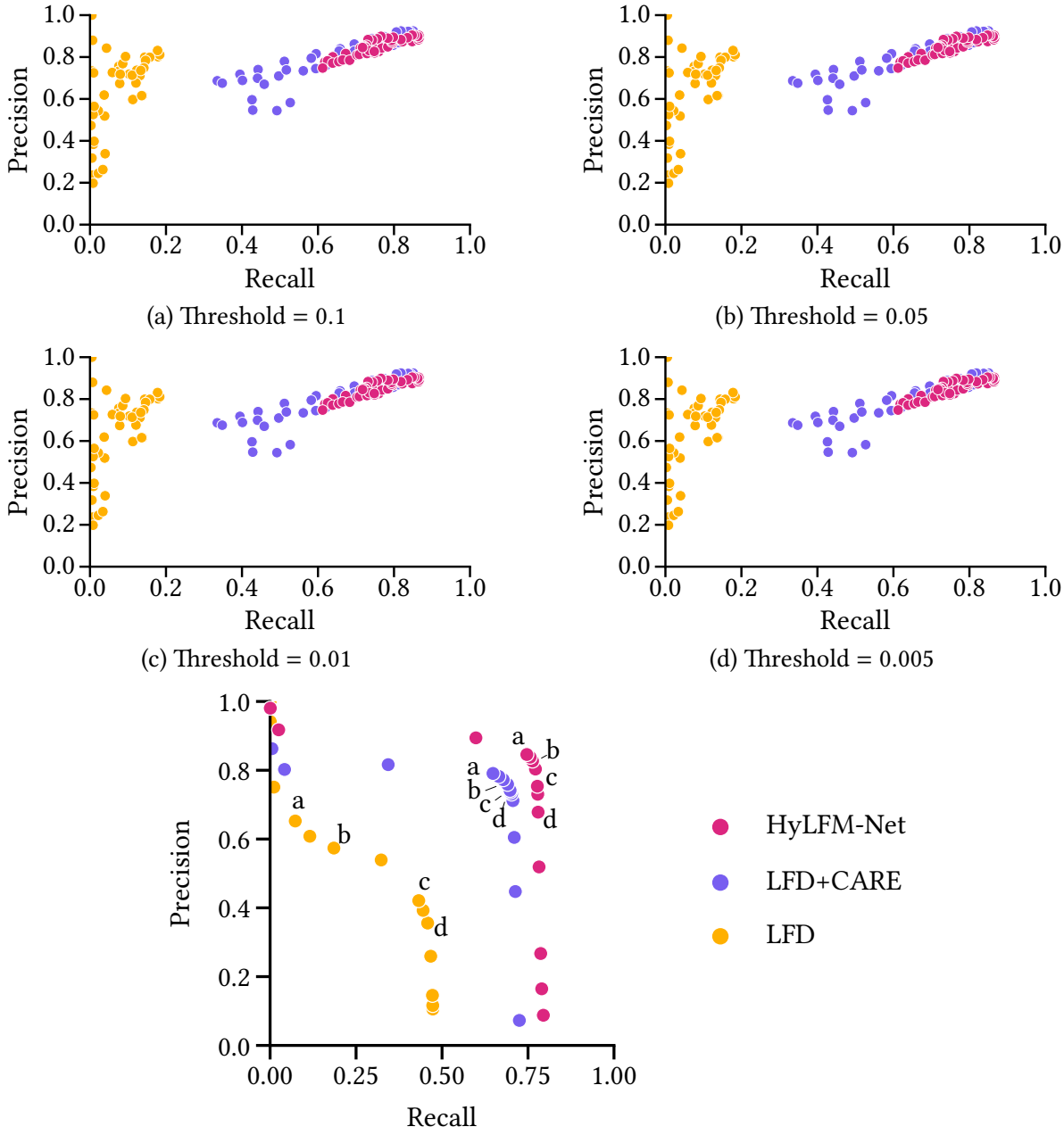


Figure 2.12: An exemplary fiducial from figure 2.11 in the focal plane of the LFM optical path, imaged by SPIM **a**. HyLFM-Net reconstructions **b** do not suffer from artifacts like LFD **c**. Scale bars are 10 µm. Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Extended Data Figure 3]



(a) Threshold = 0.1

(b) Threshold = 0.05

(c) Threshold = 0.01

(d) Threshold = 0.005

(e) Thresholds in $\left[1 \times 10^{-4}, 0.75\right]$. Data points marked a–d correspond to the respective average of precision/recall per plane in the above subplots (a)–(d).

Figure 2.13: Analysis of precision and recall of sub-diffraction-sized beads reconstructed by HyLFM-Net-beads, LFD, and LFD improved by CARE (LFD + CARE). True fiducial positions were determined by SPIM imaging. In plots (a)–(d) the precision/recall for each individual SPIM plane at a fixed threshold is marked. The averaged precision/recall of all axial planes for a discrete set of thresholds is shown in (e). Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Extended Data Figure 4]

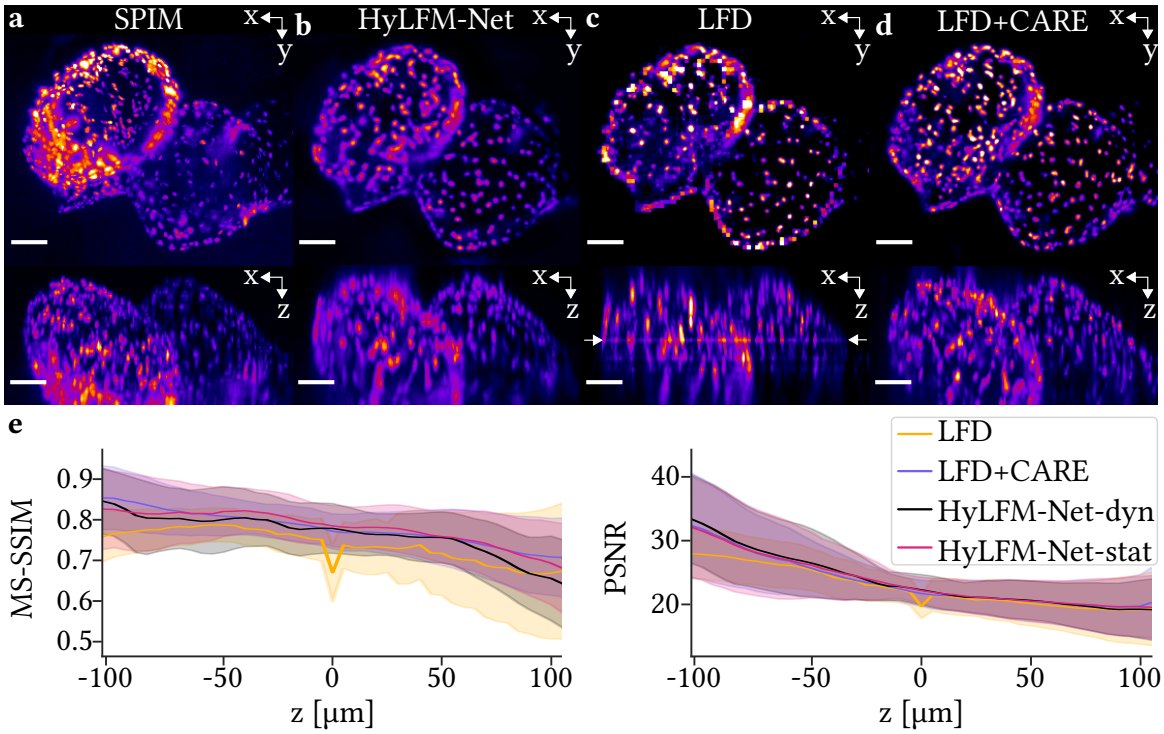


Figure 2.14: HyLFM-Net reconstructions of a static medaka heart. "**a**, A pharmacologically arrested hatchling medaka heart acquired by light sheet (SPIM, maximum intensity projections). **b**–**d**, The corresponding light-field volume reconstructions by HyLFM-Net (**b**), LFD (**c**) and LFD + CARE (**d**). Here, HyLFM-Net (later HyLFM-Net-stat) was trained on different static volumes. Note reconstruction artifact (white arrows) and signal dimming in off-center regions in LFD. Images are representative of n = 197 volumetric heart images with 49 z slices each. **e**, MS-SSIM and PSNR image quality metrics across the imaging volume of HyLFM, LFD and LFD + CARE restorations compared to SPIM ground truth (197 volumetric samples). Scale bars are 50 µm. [...] Shadows in **e**[...] denote standard deviation."[74] Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Fig. 3]

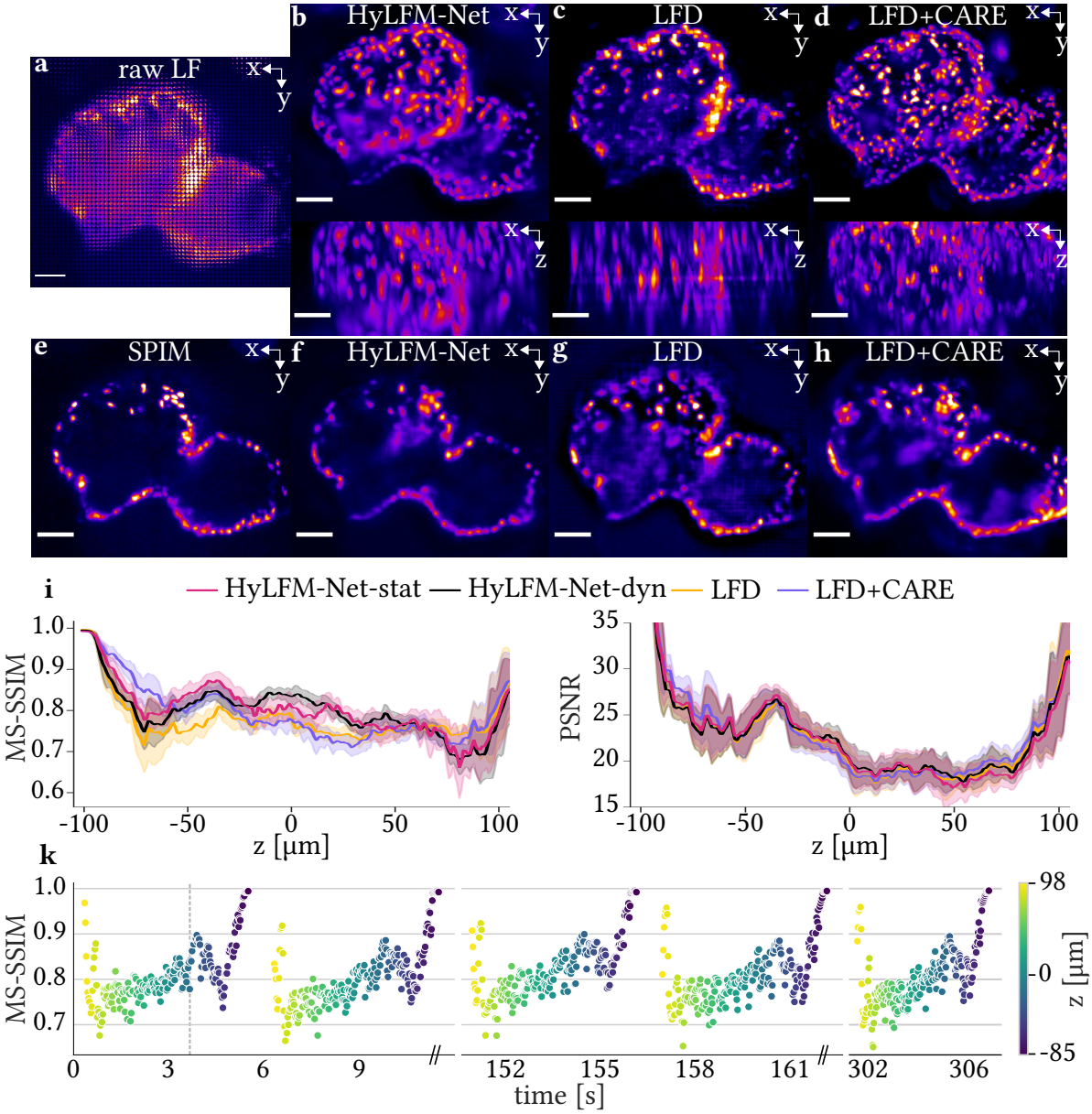


Figure 2.15: Reconstructions of a beating hatchling medaka heart 8 dpf, recorded at 40 Hz. The raw light field image **a** is reconstructed by HyLFM-Net-stat **b**, LFD **b** and LFD + CARE **c** (maximum intensity projections). A SPIM plane **e** at −27 µm off the light field focal plane is taken simultaneously with the light field in **a**. Such individual planes allow to compute validation metrics MS-SSIM and PSNR **i** based on 10 659 SPIM-plane-prediction-slice pairs averaged over time. Shadows denote standard deviation. Continuous MS-SSIM validation of a pretrained HyLFM-Net-stat (same model used to generate prediction **b**) is illustrated in **k**: As the SPIM plane (**e**) is moved through the volume over time, a corresponding slice of the HyLFM-Net prediction (**f**) is evaluated. The axial position is color-coded over the depth of the heart relative to the LFM focal plane. SPIM plane **e** and the matching HyLFM-Net-stat reconstruction slice **f** correspond to the vertical, dashed line in **k**. Scale bars are 50 µm. Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Fig. 3]

Lucy) LFD and LFD enhanced by a DL image restoration method (LFD + CARE). Like HyLFM-Net-stat, HyLFM-Net-dyn outperforms LFD and performs comparable to LFD + CARE at much higher reconstruction speeds.

To showcase another application of a HyLFM-Net trained only on SPIM slices of a dynamic sample, I trained a HyLFM-Net-brain (see table 2.1) on HyLFM acquired calcium imaging, capturing neural activity of zebrafish larvae. This is an application for which it is not feasible to obtain volumetric ground truth with SPIM due to the activity dependent fluorescence — we cannot arrest the brain. My collaborator Juan Carlos Boffi was able to extract neural traces from LFD and HyLFM-Net-brain reconstructions that correlated equally well to SPIM as presented in figure 2.16.

Due to the continuous SPIM validation in HyLFM evaluations of reconstruction fidelity are feasible during regular microscopy operation — no need for compromises to acquire ground truth (slices). As explained in the context of HyLFM-Net-dyn these slices are sufficient for training a HyLFM-Net. This motivates fine-tuning of pretrained HyLFM-Nets on dynamic data to further improve reconstruction fidelity. Note that to keep validating the reconstruction model, not all available SPIM slices should be used to continue HyLFM-Net training. However, the majority of acquired SPIM planes may be used for fine-tuning. Figure 2.17 evaluates fine-tuning of HyLFM-Net-stat on a beating medaka heart not included in its static medaka heart training data. Mere minutes of further training on the unseen dynamic sample improve HyLFM-Net reconstruction fidelity.

I evaluate the practicability of fine-tuning for different domain gaps in figure 2.18. As is to be expected, the larger the domain gap, the longer it takes to converge to peak performance. Even large domain gaps (HyLFM-Net-beads applied to medaka heart) can be overcome, although this likely involves catastrophic forgetting — no benefit is gained from starting the training process with a pretrained HyLFM-Net compared to a randomly initialized one.

2.4.5 Evaluation of Inference Speed

NN inference speed is hardware and software dependent and thus subject to change. Fortunately code execution times tend are generally shortening with hardware and software updates. At the time of submitting N. Wagner*, F. Beuttenmueller* et al. [74] I evaluated the

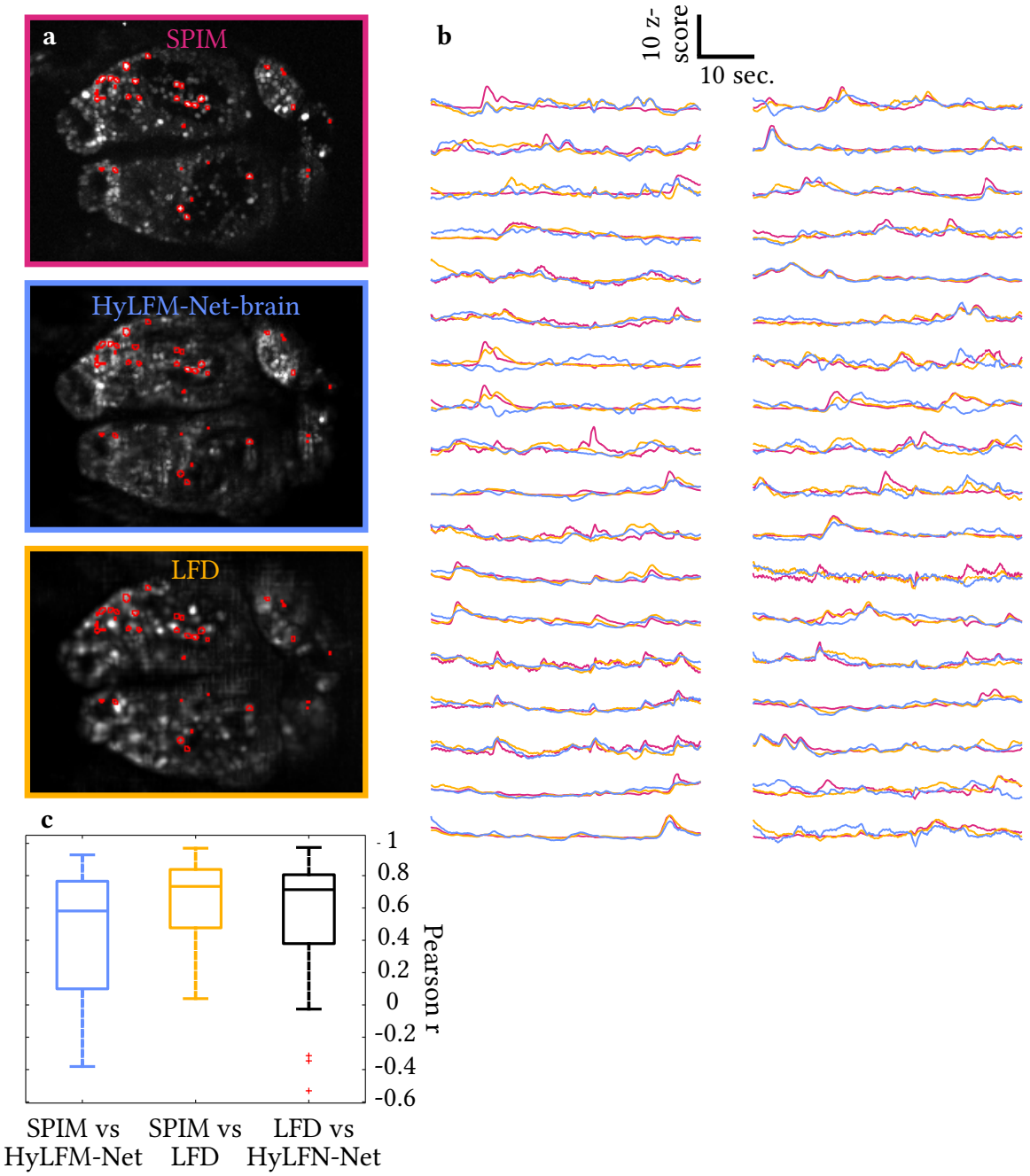


Figure 2.16: Neural activity in a larval zebrafish brain. "**a**–**c**, Representative image plane acquired with SPIM, HyLFM-Net and LFD, respectively (standard deviation projection over time). **d**, Selected Ca^{2+}-traces extracted from regions indicated in (**a**–**c**), 36 traces with 150 time points analyzed. **e**,Comparison of Pearson correlation coefficients (R) of Ca^{2+}-traces extracted by SPIM, HyLFM-Net and LFD. Note that the difference between LFD and HyLFM-Net[-brain] performance is not statistically significant (p = 0.053, Dunn-Sidak). Scale bar in (**a**) is 50 µm. Results representative of n = 5 individual image planes in the volume."[74] I trained the HyLFM-Net-brain; neural activity analysis was done by Juan Carlos Boffi[74]. Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Extended Data Fig. 7]

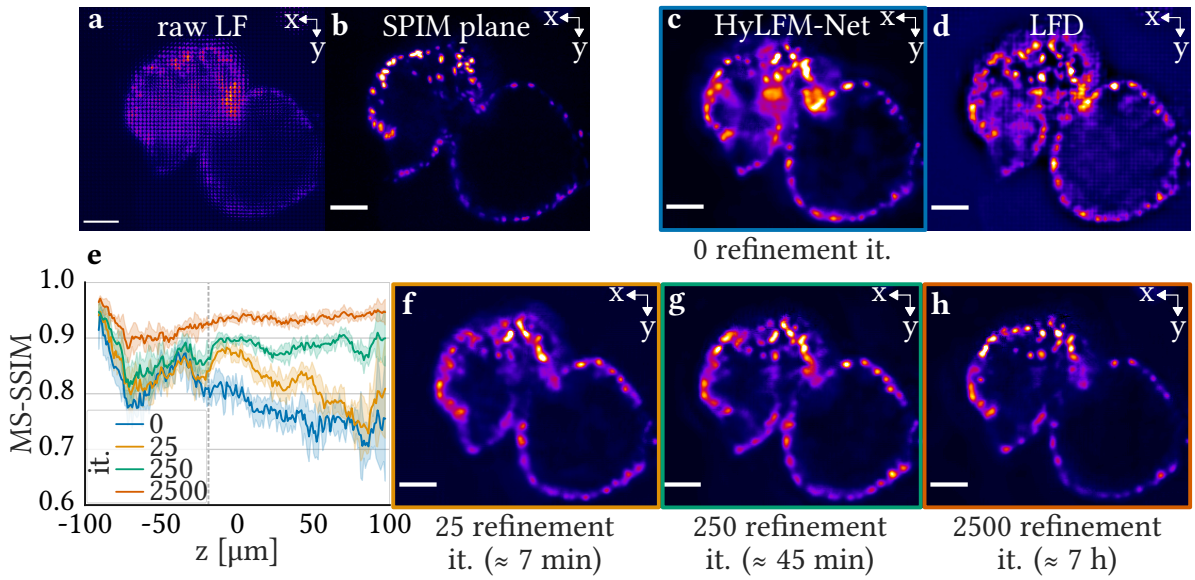


Figure 2.17: "Online network validation and refinement in HyLFM. 'a,b, Example raw light-field (LF) image of a dynamic (beating) medaka heart, acquired at 40 Hz (**a**), and high-resolution light-sheet image plane at -19 µm depth and at the same time point (b). **c**,**d**, Corresponding HyLFM-Net (**c**) and LFD (**d**) slices of the 3D reconstructions. **e**, MS-SSIM image quality metrics for increasing refinement iterations visualized in **f**–**h**. Gray dashed line in e indicates the time point and axial position shown in **c**. Shadows in **e** denote standard deviation, inferred from all test time points and single planes (n = 756) of a dynamic dataset (same as in [figure 2.15 **i**]). Scale bar is 50 µm.'[74] Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Fig. 4]"

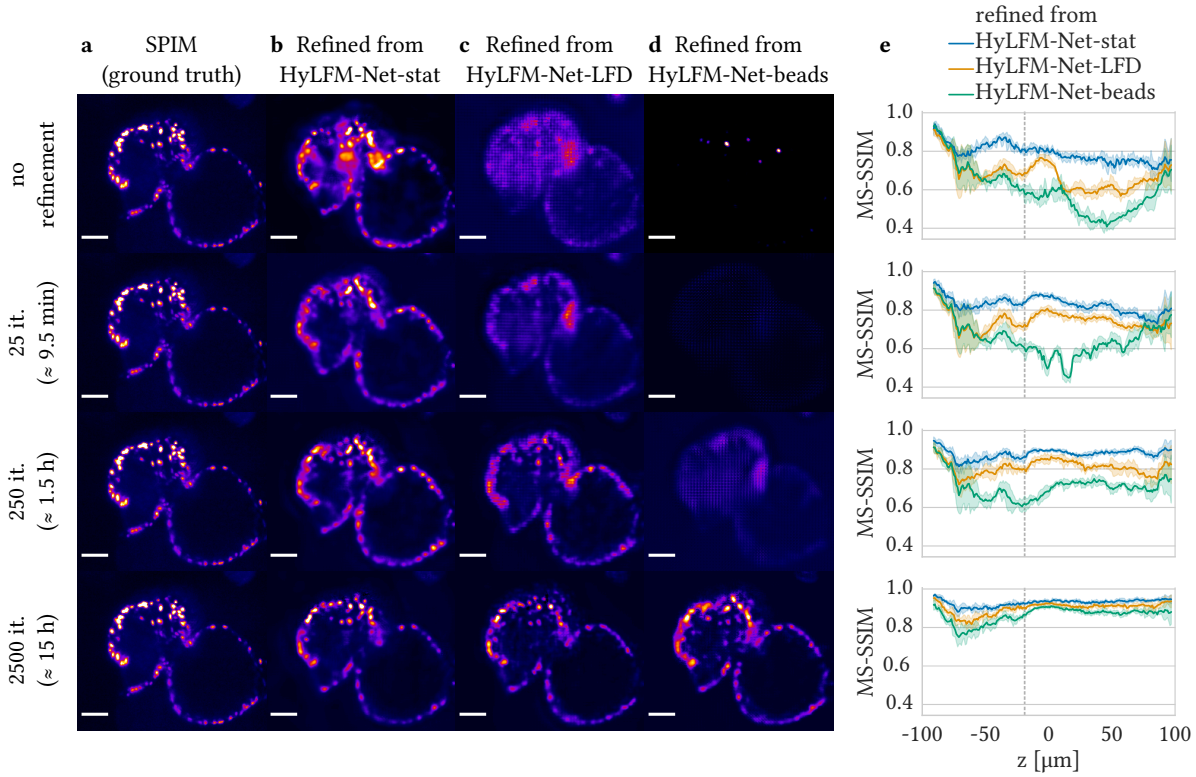


Figure 2.18: "Refinement of three differently pre-trained HyLFM networks on dynamically acquired medaka heart images. Column (**a**) shows SPIM ground truth plane at axial position z = -19µm. Columns (**b**–**d**) depict corresponding slices after increasingly many refinement iterations of different pre-trained HyLFM networks. **b**, A statically trained HyLFM-Net (such as the one in [figure 2.14]), **c**, HyLFM-Net trained on LFD reconstructions of light-field images acquired on another microscope setup[73][...] and (**d**) HyLFM-Net trained on medium-sized beads (see [figure 2.11]). **e**, Respective MS-SSIM image quality metrics for each network and stage of refinement. Standard deviation shown as shadows inferred from N = 756 individual time-points. Note that depending on the domain gap, the refinement converges at different speeds, but high fidelity results can be obtained for all pre-trained networks."[74] Figure adapted from N. Wagner*, F. Beuttenmueller* et al. [74, Extended Data Fig. 6]

specimen	medaka heart				zebrafish brain		beads	
LF size [px^2]	1235×1425				1292×1577		931×1406	
output lateral [px^2]	244×284				256×316		376×576	
output lateral [µm^2]	339.0×394.6				355.7×439.1		261.2×400.2	
output axial [px]	49				49		51	
output axial [µm]	245.0				140.0		100.0	
HyLFM-Net	-stat		-dyn		-brain		-beads	
NVIDIA GPU	2080Ti	A100	2080Ti	A100	2080Ti	A100	2080Ti	A100
sample rate [Hz]	11.1	16.9	**26.7**	**57.3**	20.4	45.9	5.8	12.9
input rate [Mpx s^{-1}]	19.5	29.7	47.0	100.8	41.6	93.5	7.6	16.9
output rate [Mvx s^{-1}]	66.3	101.0	159.6	342.4	80.9	181.9	64.1	142.5
mini-batch size	5	27	**13**	**32**	15	31	3	23

Table 2.2: HyLFM-Net inference speeds without I/O from/to the hard drive. Average of three runs of 100 consecutively processed mini-batches. Highlighted values are referenced in section 2.4.5. Table content previously published in N. Wagner*, F. Beuttenmueller* et al. [74, Supplementary Table 2].

reconstruction speed of the trained HyLFM-Nets to support our claim that a HyLFM-Net can reconstruct fluorescent volumes at video-rate speed.

On a consumer-grade GPU of the time (NVIDIA GeForce RTX 2080Ti) HyLFM-Net-dyn achieved an average volume reconstruction rate of 26.7 Hz when processing 13 frames in parallel. With a high-end NVIDIA A100 GPU this is more than doubled (processing 32 samples in parallel). Details are presented in table 2.2

2.4.6 Depth of field

This section provides further analysis of the optical setup described in sections 2.3.1 and 2.4.1 and will lead to conclusions for future improvements proposed in section 2.6.1 *Tweaking the Optical Setup*.

The theoretical depth of field (DOF) for the optical setup described in section 2.3.1 as

derived by[69, Equation 5.3] is

$$D_{\text{tot}_3} \approx \frac{(2 + N_u^2)\lambda n}{2\text{NA}^2} \tag{2.5}$$

$$\approx 189\,\mu\text{m} \tag{2.6}$$

with

$$N_u = \frac{d_{\text{MLA-pitch}}}{R_{\text{obj}}} \qquad \text{[69, adapted from Equation 3]} \tag{2.7}$$

$$\approx 11.7 \tag{2.8}$$

and

$$R_{\text{obj}} = \frac{0.47\lambda}{\text{NA}} M \qquad \text{[69, Equation 2]} \tag{2.9}$$

$$\approx 10.7\,\mu\text{m}\,. \tag{2.10}$$

As demonstrated in figures 2.14 and 2.15 the HyLFM-Net can reconstruct biological samples with a depth of around 200 µm successfully, which is in agreement with this estimate of the theoretical DOF. Limiting the quantitative evaluation (using fiducials, see figure 2.11) to −50 µm to 50 µm axially was motivated by the fact that LFD did not yield usable results beyond these limits to compare to. The quantitative evaluation beyond these limits is left for future work.

2.5 Discussion

I demonstrated that HyLFM in conjunction with reconstructions by HyLFM-Net allows to image — artifact free — highly dynamic samples with near SPIM fidelity at LFM frame rates. Precautions need to be taken to avoid to avoid biased reconstructions, but the continuous validation capabilities of HyLFM avoid uninformed applications of this powerful method. Additionally, fine-tuning improves image fidelity further and reduced training times through the viability of reusing trained HyLFM-Nets.

2.5.1 Training Bias

The ability to refine HyLFM-Nets with relative ease to obtain high quality reconstructions is a great benefit of the HyLFM method (see section 2.4.4 in particular figures 2.17 and 2.18). However, like any NN training, HyLFM-Net training induces biases that need to be considered at inference time. To demonstrate and quantify such a learned bias, I trained one HyLFM-Net-beads on small fluorescent beads (0.1 µm in diameter) and another HyLFM-Net-beads on large beads (0.4 µm) and used them both to reconstruct test data of small and large beads. The results, presented in **??**, show clear biases towards the bead size 'seen' by the HyLFM-Net during training. This emphasize the importance of continuous validation to detect and quantify deviations from the more accurate SPIM measurements.

The role of DL

HyLFM as a method would not have been possible without DL. With the design of a novel architecture and descriptions of the HyLFM training and fine-tuning procedures its contributions lie partially in the DL domain. As computation speed of specialized hardware further increases larger NNs are capable of processing ever larger data inputs. This opens possibilities to extend HyLFM's capabilities as

Dissemination of HyLFM

Given how few laboratories can build their own HyLFM microscopes and that trained HyLFM networks cannot be expected to perform across different optical setups, training and deployment of HyLFM networks will likely involve DL experts of some degree in the near future.

Efforts to make HyLFM networks and their successors more accessible would greatly benefit from more standardized descriptions of training algorithms and datasets. Such standards have the potential to boost computational methods like HyLFM by reducing the perceived software complexity and lowering the obstacles to 'just give it a try'. Accessibility to HyLFM and other LFM variants might also increase with increasing popularity of LFM in general. LFM paths might find their way in various hybrid microscopy designs whenever high speed image acquisition is a concern.

2.6 Outlook

In this section I discuss potential optical and computational improvements and variations to HyLFM. In this section I discuss potential improvements to and variants of the optical setup, as well as HyLFM-Net and training algorithms and variations to HyLFM.

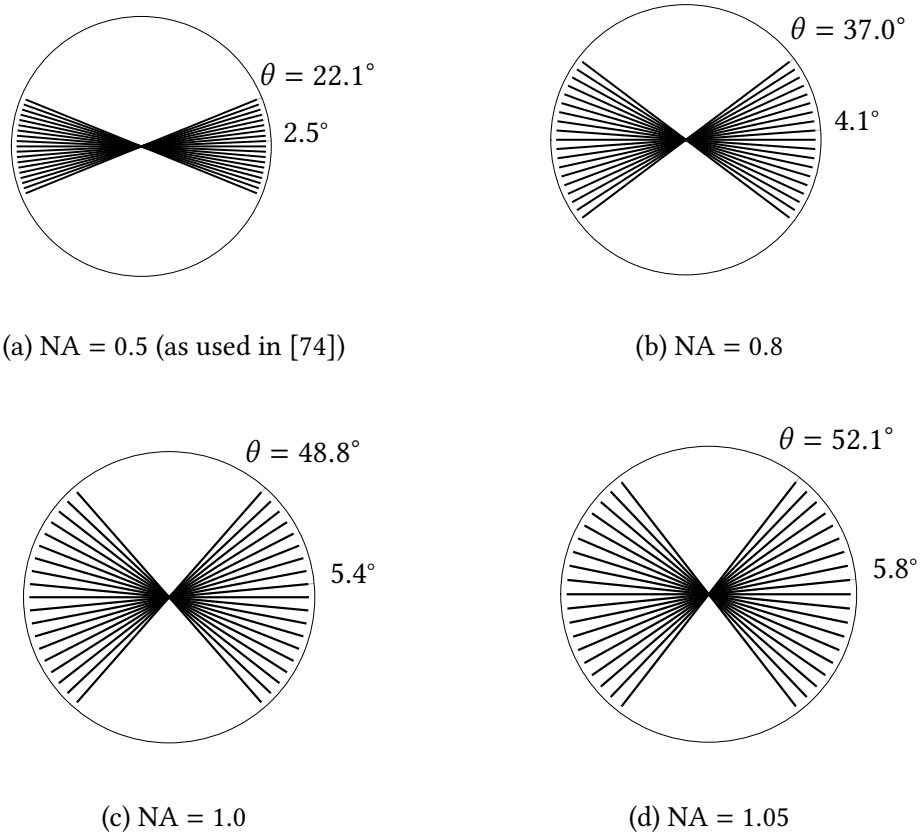
As computation speed and memory capacity of specialized hardware like GPUs and TPUs further increases, larger NNs are capable of processing ever larger data inputs. This opens possibilities to extend HyLFM's capabilities with new model architectures, extended training algorithms and additional model inputs.

2.6.1 Tweaking the Optical Setup

When designing the optical setup there are many trade-offs concerning lateral and axial resolution and the FOV. Key parameter to adjust for an alternative optical setup are the objective's magnification and NA, as well as the MLA's pitch (the microlenses' diameter/offset of neighboring microlenses) and the camera sensor. To limit the search space for tweaking the optical setup, the camera sensor and MLA are viewed as given in this section. Also the emission wavelength λ is kept at $\lambda \approx 510\,\text{nm}$ as described in section 2.4.1 *Experimental Setup*. The following considerations map directly to alternative values for these parameters. Note that the size of the MLA should cover the camera sensor; otherwise the image size may be adjusted optically in-between.

In an LFM setup according to [69] the axial dimension of the FOV can be estimated by the DOF as confirmed in section 2.4.6 *Depth of field*. It remains to be seen if a HyLFM-Net can be trained to recover signal in even greater depths if high quality ground truth is available, possibly exceeding the estimate given by equation (2.5).

For designing future HyLFM-like optical setups, one should keep in mind that the HyLFM-Net increases the effective resolution axially, as well as laterally; HyLFM-Net is capable of improving the resolution through the learned bias as shown in figure 2.11. With respect to the implemented optical setup described in section 2.3.1 this circumstance motivates to trade lateral resolution for axial resolution in future setups, which should allow the HyLFM-Net to improve the overall resolution, because some of the degraded lateral resolution due to changes to the optical setup can be recovered, while optical improvements



(a) NA = 0.5 (as used in [74])

(b) NA = 0.8

(c) NA = 1.0

(d) NA = 1.05

Figure 2.19: Perspective views illustrated as a tilt-series for different optical LFM setups.

to the axial resolution may lead to the detection of objects previously 'hidden' in the limited viewing angles. A more isotropic resolution should maximize the positive effect of the learned bias and minimize erroneously merging object features. An isotropic resolution is also desirable for many downstream analysis tasks like object tracking.

The most straight-forward path to improving axial resolution is to use a detection objective with a higher NA as equation (2.4) indicates. The resulting differences in half-angle can be quite substantial as illustrated in figure 2.19. Exemplary objectives realizing the NA values in figure 2.19 are listed in table A.2. Various LFM setups have successfully used objectives with a high NA (NA >= 0.8)[87, 88, 75, 89].

Objectives with a slightly increased magnification should still be able to capture a sufficiently large FOV in an otherwise unchanged optical setup. Of course maximizing the

camera sensor area used for a given FOV by increasing the effective magnification is another route to improve image resolution slightly.

2.6.2 Iso-HyLFM

If lateral resolution may not be compromised in order to achieve isotropic resolution, an alternative path to obtaining isotropic resolution has been demonstrated by Wagner et al. with their "Iso-LFM"[73]. They archive isotropic resolution by adding a second detection objective orthogonal to the first (and the illumination) and fusing the independently deconvolved light field reconstructions[73]. An adapted NN architecture based on the HyLFM-Net —'Iso-HyLFM-Net'— could implement a second encoder (all layers before the Rearrange layer in figure 2.10) or simply a second input channel. Either change would allow to deconvolve both captured light fields jointly, which has the potential to outperform independently reconstructed volumes fused in postprocessing. This might be true especially for transformer architectures (see section 2.6.7) as their attention mechanism could efficiently facilitate the information exchange between the two light field recordings.

If spatial constrains allow it, an additional detection objectives may provide an additional view to further improve image resolution and SNR[90].

2.6.3 Local Hints for HyLFM-Net

Even without additional views as described in section 2.6.2 HyLFM-Net may greatly benefit from additional inputs (recorded by the optical setup presented in section 2.3.1). The simultaneously recorded SPIM slice may be given as an additional HyLFM-Net input to locally inform the reconstruction. Sweeping through the volume over time these SPIM slices have the potential to significantly improve the average volumetric reconstructions. To train such a network static ground truth would be required to avoid the trivial solution of simply returning the SPIM input slice. For an efficient positional encoding of the provided SPIM slice, the aligned slice (in an empty volume) could be concatenated with the features in different pooling levels of the decoder (all layers after the Rearrange layer in figure 2.10) akin to skip-connections.

Reverse one Optical Path

When training a HyLFM-Net on two-dimensional ground truth slices of a dynamic sample using SPIM slices *perpendicular* to the LFM's optical axis makes a single training step more computationally efficient, because predictions of axially limited subvolumes can be computed directly. As object depth is presumably more difficult to infer from the captured light field though, training iterations with SPIM slices *parallel* to the optical LFM axis may prove more informative, potentially reducing the number of training iterations needed for convergence. *Parallel* SPIM slices added as an additional input (see section 2.6.3) may prove to be a more valuable local hint during reconstruction in comparison to *perpendicular* ones for the same reason. The orientation of SPIM and LFM may be changed by reversing one of their optical paths.

2.6.4 De-scattering with HyLFM

Recently DL has been utilized successfully to de-scatter microscopy images based on ground truth recorded using point-scanning two- (or three-) photon microscopy (PSTPM)[91]. This indicates that combining the LFM deconvolution with de-scattering in one next-level HyLFM-Net has great potential, despite the intuitively assumed need for ballistic photons for depth reconstruction. The question remains to what extend trained, biologically informed biases are able to aid a NN to perform volumetric reconstruction of heavily scattered light fields. To increase captured information (without increasing phototoxicity) in order to improve the reconstruction in this proposed, even more challenging experimental setup, a multi-view approach as described in section 2.6.2 *Iso-HyLFM* should be considered for this endeavor.

If high imaging speed (for dynamic refinement and/or validation) should not be sacrificed, employing a 2PE-SPIM instead of the SPIM would allow to maintain a high two-dimensional ground truth image acquisition speed, while operating in slightly deeper/more scattering tissue with lower SNR[65].

2.6.5 Smart and Event Based HyLFM

Due to HyLFM-NetHyLFM-Net's real-time inference speed (see table 2.2 'local hints' (see section 2.6.3) may be guided by real-time performance evaluation. Detecting events of biological interest fast within the whole LFM FOV and capturing a small subvolume with SPIM subsequently has the potential to advance high speed, high quality imaging in specific use-cases. This is distinct from events on a lower (hardware) level. Event cameras have been successfully deployed to increase LFM imaging speed "at kHz frame rates"[92].

2.6.6 Pseudo-time for HyLFM-Net

Providing a few (e.g. 3) light field time frames as light field input channels adds context for the reconstruction of a single time point and thus should improve the effective resolution not only in areas of slower sample movement. It may prove especially useful in single color mode as the light fields captured before and after any given SPIM slice are both equally distant in time, allowing the NN to learn to interpolate between these time points and finally reconstruct based on more captured photons.

2.6.7 Alternative Network Architectures

In addition to expanding the inputs of a HyLFM-Net architecture as described in sections 2.6.3 and 2.6.6 different layers and base architectures should be explored further.

It has been demonstrated that (separable [93, 94]) four-dimensional convolutions can be used to process discrete light field views[95, 96] instead of encoding them as channels like HyLFM-Net does. A direct speed and performance comparison between these options — high channel two-dimensional convolutions, separable four-dimensional convolutions as alternating two-dimensional convolutions or fully four-dimensional convolutions — is outstanding. Especially inference speed may depend on the DL framework (implementation details of the convolution operations) and the hardware used, as well as n_num, the pixel size of the lenselet subimages.

An alternative reconstruction approach is to apply an image restoration technique such as CARE[11] not to LFD predictions, but to a focal stack[69] that can be computed much

faster. This has recently been done by Guo et al.[92] based on a network proposed in Xue et al. [97].

With more LFM datasets becoming available, benchmarks for LFM reconstruction methods can be established to compare the proposed model architectures and formulate recommendations based on a given optical setup and experimental requirements.

Meanwhile Lu et al. have shown that adding attention layers can improve the efficiency of the HyLFM-Net architecture[98]. When considering a more drastic change of network architecture the (hybrid) Vision Transformer (ViT)[34] in particular is a promising candidate to draw more inspiration from. Rehman et al. demonstrate the potential of this model architecture familiy with their version of a hybrid transformer, the Convolutional Neural Network Transformer (CNNT)[99]. Their model outperforms other state-of-the-art denoising methods like CARE on fluorescence microscopy images, while reduing training time due to their network's amenebility to fine-tuning[99]. However, they report an inference time of approximately 1 minute for a single, large (100 px × 1200 px × 1200 px) image[99], implying that a similar architecture for LFM reconstructions would require 21 days to reconstruct a five minute experiment recorded at 100 Hz (note that this image size is roughly 6 times greater than the output of HyLFM-Net, see table 2.1). Nevertheless, hardware capabilities are steadility increasing and parallelization, reducing the NNs size or processing smaller images might still improve on current state-of-the-art LFM reconstruction models.

The introduction of attention mechanisms to the HyLFM-Net model architecture is particularly interesting when combining it with the idea of 'local hints' discussed in section 2.6.3. Depending on the position of the SPIM plane, different network areas could learn to pay attention to it, which might be an avenue to elegantly incorporate single SPIM planes to locally inform the reconstruction. This has some similarity with Masked Autoencoders[100], that reconstruct images from partial inputs.

2.6.8 Sample Modelling

As HyLFM-Nets can be refined/trained on dynamic samples, training a HyLFM-Net instance for a specific sample has shown great success, see section 2.4.4. Following this train of thoughts — training a NN for a single sample — is related to the concept of neural rendering[101]: "predict[ing] the appearance of [a] scene from previously unobserved

viewpoints."[101]

In neural radiance field (NeRF)[102] "a scene [is represented] using a fully connected (nonconvolutional) deep [neural] network, whose input is a single continuous 5D coordinate (spatial location (x, y, z) and viewing direction (θ, φ)) and whose output is the volume density and view-dependent emitted radiance at that spatial location."[102] This NN is trained through minimizing the difference of ground truth views to views rendered from its predictions. This is possible, because the rendering process is differentiable[102]. A key difference of the proposed NeRF to prior novel-view-synthesis work[103] (or a HyLFM-Net) is that it represents the scene not as a discretized grid of voxels, but as a continuous function.

Training a NeRF bears some resemblance to the iterative Richardson-Lucy[76, 77] algorithm — the volume density is estimated/learned by comparing the output of the forward path/rendering process to recorded views. Introducing the comparison of the convolved prediction to the recorded view as a cyclic constraint to the HyLFM-Net training has the potential to improve prediction accuracy and can be extended to refine predictions at test time as well.

Adding a time input to NeRF models to represent dynamic scenes as been explored[104, 105]. Transferring this approach to LFM deconvolution could yield volumetric video reconstructions of very high quality. HyLFM presents an ideal experimental setup to train and evaluate such approaches to detect and avoid hallucinations, thus making the reconstructions trustable and amenable to further quantitative analysis.

3 BioImage Model Zoo: A Community-Driven Resource for Accessible Deep Learning in BioImage Analysis

The BioImage Model Zoo (BMZ) is a collaborative effort to bring DL to a growing bioimage analysis community. The design of the BMZ is guided by the FAIR principles and a joint effort of the AI4Life[106] consortium.

The founding collaborators were Wei Ouyang*, me, Estibaliz Gómez-de-Mariscal*, Constantin Pape*, Tom Burke, Carlos Garcia-López-de-Haro, Craig Russell, Lucía Moya-Sans, Cristina de-la-Torre-Gutiérrez, Deborah Schmidt, Dominik Kutra, Maksim Novikov, Martin Weigert, Uwe Schmidt, Peter Bankhead, Guillaume Jacquemet, Daniel Sage, Ricardo Henriques, Arrate Muñoz-Barrutia, Emma Lundberg, Florian Jug, and Anna Kreshuk — (first*) authors of "BioImage Model Zoo: A Community-Driven Resource for Accessible Deep Learning in BioImage Analysis"[107]. All collaborators are associated with one ore more BioImage.IO community partners (ilastik, the DeepImageJ plugin for Fiji, ImageJ, and ImageJ2[108], ZeroCostDL4Mic, ImJoy, and Icy)[107] — software projects in the area of bioimage analysis.

As outlined in chapter 1 applying DL to biological problems can be difficult and expensive. A straight forward approach to alleviate costs and provide modern AI empowered workflows to a wide range of researchers and other users is to not only share acquired datasets, but also trained models. Sharing such resources is the central idea of the BMZ. Through combined efforts symbiotic partnerships can form that have greater reach and

ultimately a greater impact.

This notion of a model collection, or 'model zoo', has been growing in popularity over recent years with efforts focused mainly on natural image processing and natural language processing (NLP). Examples of such "model zoos" include huggingface.co with a dedicated library for NLP with transformers[109], DLHub[110], pytorch.org/hub[111], tensorflow.org/hub[112], modelzoo.co[113] and MONAI[114]. The usability of a model collection is balanced with its scope of model applications: The narrower the collection's focus, the more specific shared models can be described and the easier it is to find a model within the collection's scope; the wider the range of research fields and applications a model collection covers, the more generic common model descriptors become.

Large collections profit from shared infrastructure and provide better model exposure compared to niche collections. Dedicated software programs on the other hand can cater to specific use-cases of pre-trained models and may provide excellent usability — even for users unfamiliar with DL and computer programming.

The BMZ addresses this gap between vast code based model collections — like — primarily catered to DL developers and dedicated user-friendly tools or plugins in the bioimage community[107]. It is not practical to develop a standalone tool or plugin for every new DL method to ensure accessibility and generate impact among life scientists regardless of their computational skills. Furthermore, using a multitude of custom applications takes effort to navigate and use effectively. To make new DL methods more accessible, the interoperability of DL models across a variety of familiar, user-friendly tools and plugins is required. A common, well-defined model format can ease integration into respective applications. In the BioImage.IO ecosystem deployment and dissemination of new DL methods are further accelerated by the use of common libraries (see sections 3.2, 3.4 and 3.5) to bring well-maintained features to a variety of tools, an therefor to end-users, fast. This approach combines the best of two worlds: a large, but community specific model zoo providing well documented models with tight integration to user-friendly and widespread software tools.

Creating a smooth user and developer experience across open source software constitutes a challenge due to independent governance and varying developer preferences.

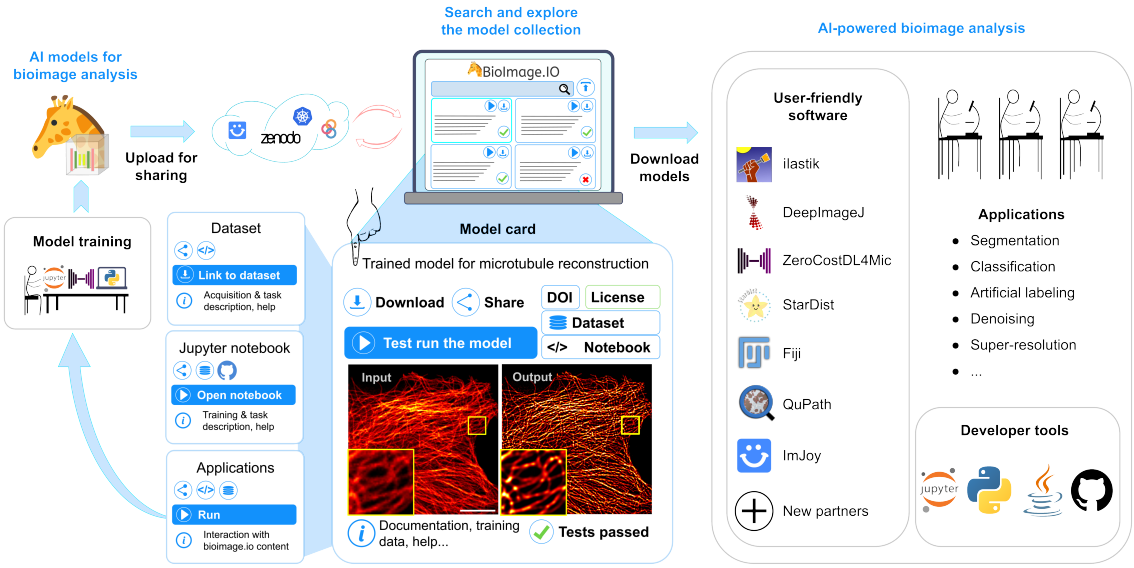


Figure 3.1: With applications, notebooks and datasets discovered on bioimage.io users train AI models for bioimage analysis. Trained models are described following the format specifications of the BioImage.IO and uploaded for sharing through the BMZ. The shared models can be downloaded and deployed with user-friendly software by the wider bioimage analysis community in a variety of applications. Development of community partners' software and custom workflows is supported by dedicated Python and Java libraries. Figure from W. Ouyang*, F. Beuttenmueller*, E. Gómez-de-Mariscal*, C. Pape* et al. [107, Fig. 1].

3.1 Overview

In this section I give an overview of the BioImage Model Zoo (BMZ) and highlight my contributions to it. The BMZ is a community platform to share AI models, datasets and analysis tools. It relies on user contributions illustrated in figure 3.1. The contributed resources constitute the BioImage.IO Collection — a database of all shared model, dataset, notebook and application descriptions. The bioimage.io website[115], serves as a central entry point to discover this BioImage.IO Collection. Its development is led by Wei Ouyang at github.com/bioimage-io/bioimage.io[115]. To grow and maintain the BioImage.IO Collection, I developed the github.com/bioimage-io/collection-bioimage-io[116] repository, as well as the *bioimageio_collection_backoffice* Python package at github.com/bioimage-io/collection as its successor[117]. These semi-automated processing tools help maintaining and updating

the BioImage.IO collection as explained in section 3.3 *Content of the BMZ: The BioImage.IO Collection.*

Each BioImage.IO resource is defined by an *rdf.yaml* file, a metadata file holding description according to BioImage.IO specifications. I designed and developed the `bioimageio.spec` Python library[118] that defines these specifications with contributions from Constantin Pape, Wei Ouyang, Dominik Kutra, Tomaz Vieira, Estibaliz Gómez-de-Mariscal and others. Among other utility functionality, the `bioimageio.spec` library also allows to validate resource metadata for correctness and completeness. The `bioimageio.spec` Python library, its motivation and details on the specifications it implements are described in section 3.2 *BioImage.IO Resource Descriptions*, in particular section 3.2.4.

On bioimage.io, collection entries are represented by cards as shown in figure 3.2. The displayed resource cards may be filtered by type, tags, and free text. To date the BMZ differentiates between four resource types: models, datasets, notebooks and applications. These four types may be chosen from with tab-like links located above the search input box (at the time of writing notebooks are included in the applications tab). Users browsing the BMZ may test a model on their own data directly on the bioimage.io website. This is one of the features enabled by the BioEngine[107], a BioImage.IO specific application framework powered by ImJoy[119], "an open-source computational platform"[119]. Models can be downloaded as ZIP archives which is facilitated by another dedicated BioEngine[107] application: the BioImage.IO Packager. In a graphical user interface (GUI) users may select which software they intend to use the model with, which will select a model weights format most suitable to the respective software in order to reduce the download size to one set of model weights. The framework dependent model weights format can also be chosen manually. The downloaded model package can be used to run inference on suitable data in various software tools such as ilastik's *Neural Network Classification* workflow[120], the DeepImageJ[121] plugin for ImageJ[108] or the `bioimageio.core` Python library[122].

I developed the `bioimageio.core` Python library[122] providing tools for BioImage.IO model use in Python with contributions by Constantin Pape, Maksim Novikov, Dominik Kutra and others. Details are elaborated on in section 3.4 *Working with BioImage.IO Resources*. Meanwhile, a Java library, the Java deep learning library (JDLL), was created to bring similar capabilities to Java based software[123].

The BioImage.IO Collection can also be accessed programmatically through an auto-

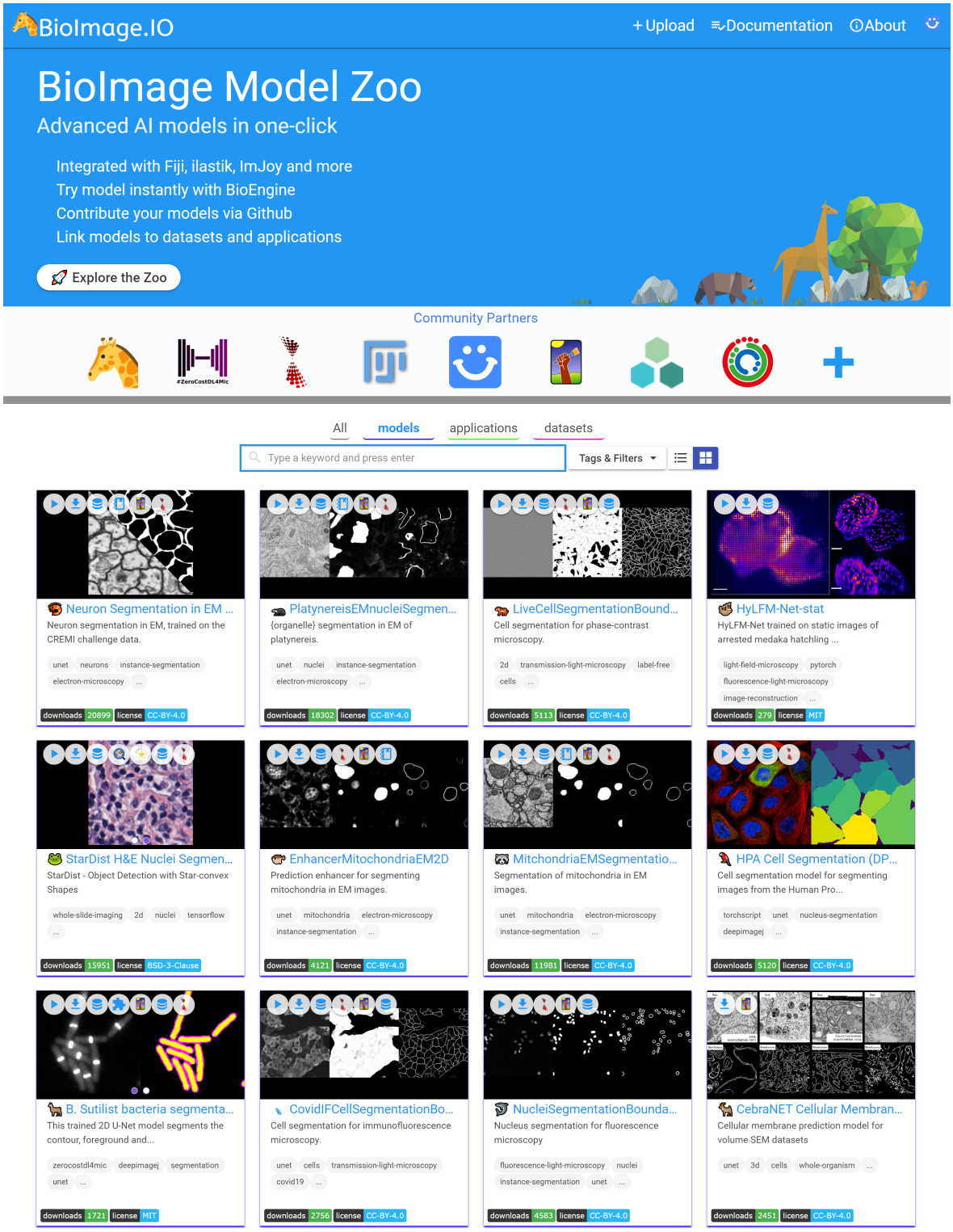


Figure 3.2: The bioimage.io home page shows current community partners and model cards[115, screenshot, model order and white space edited].

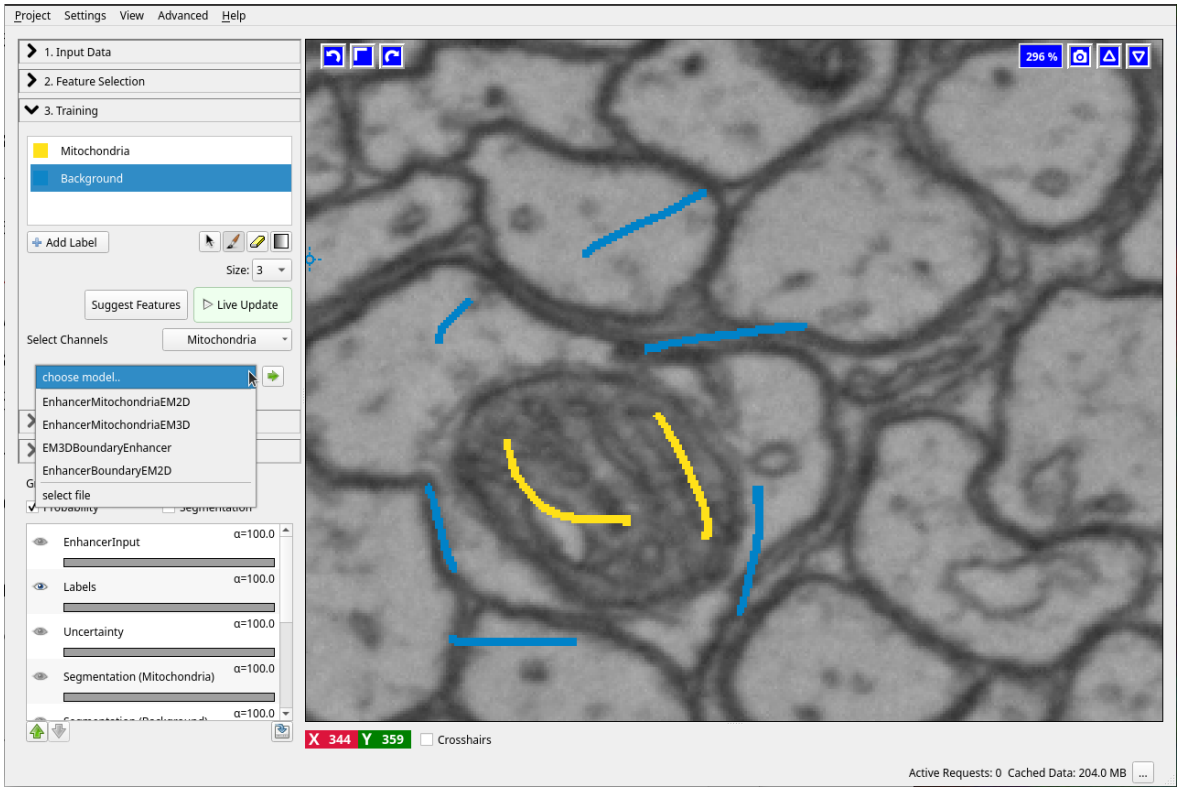


Figure 3.3: In ilastik's *Trainable Domain Adaptation Workflow* compatible "Enhancer" models shared in the BMZ can be selected in a drop-down menu[124, screenshot].

matically updated JavaScript Object Notation (JSON) file: *The collection.json*. This allows software applications to present (a select subset of) the BioImage.IO Collection, for example as a drop-down menu of compatible models as part of the DeepImageJ GUI or ilastik's *Trainable Domain Adaptation Workflow* shown in figure 3.3. The bioimage.io website uses this *collection.json* file to render the BMZ content as well. Users with scripting or coding experience can access the collection's resources via dedicated Python packages that I introduce in sections 3.2.4 and 3.4.

Resources were initially added by creating a Zenodo[125] record, tagged with 'bioimage.io', that contains an *rdf.yaml* file. An upload interface at bioimage.io was initially capable of creating such a Zenodo record with a user's authorization and uploaded data. Meanwhile, the interface uploads new contributions to an intermediate cloud storage and triggers further processing by the `bioimageio_collection_backoffice` at github.com/bioimage.io/col-

lection directly.

3.2 BioImage.IO Resource Descriptions

> If you think of ”standardization” as the best that you know today, but which is to be improved tomorrow, you get somewhere.
>
> (Henry Ford, Today and Tomorrow, 1926)

To share models across the bioimage community standards are required. In this section I describe conclusions of many rounds of discussion among the BioImage.IO development team to converge on a solution that would suit our different requirements. Creating a flexible, permissive standard to cover a wide range of needs within the diverse bioimage community had to be balanced with a more restrictive approach crucial to integrating collected metadata into software applications in a meaningful way. All while keeping the development of and interaction with the BMZ accessible to all community members. Beside me, leading roles in these discussion were taken by Wei Ouyang, Constantin Pape, and Estibaliz Gómez-de-Mariscal. These efforts were supported by Anna Kreshuk, Florian Jug, Arrate Muñoz Barrutia, Tomaz Vieira, Carlos Garcia-López-de-Haro, Guillaume Jacquemet, Daniel Sage, Ricardo Henriques, Tom Burke, Dominik Kutra, Maksim Novikov, and Deborah Schmidt. Some conclusions of the initial discussions were later revised. Throughout this chapter I mark decisions or details as *initial* or *revised* accordingly. The ongoing refinement of the specification takes place alongside the development of the bioimage.spec Python library. As I detail in section 3.2.4, this library defines the specification and implements validation of respective metadata.

Supported Weight Formats

In order to share DL models, model weights and metadata need to be saved in a suitable format. There are several formats available to share model weights depending on the DL framework used to train the model. PyTorch[47] and TensorFlow[48] are popular frameworks in the DL research community. Many models have already been published using these frameworks on bioimage analysis problems. Hence, both of these need to be supported by the BMZ. The open Open Neural Network Exchange (ONNX) format seemed

to be a promising candidate to represent models from either ecosystem, but with its limited set of operators not every model architecture can be represented[126]. Nevertheless, for supported model architectures ONNX is a convenient format to share model weights with, due to its easy deployment and cross-platform compatibility.

PyTorch and TensorFlow both offer multiple ways of serializing trained models. For PyTorch, the native weights format is the 'state dict' format[127]. It holds all learned model parameters. A model can only be restored from these weights in combination with the model architecture defined in the model's source code. Due to this dependence on source code this approach is not ideal for deployment as it requires a suitable Python environment (with any libraries installed that are used in that source code) and the execution of untrusted (potentially harmful) code — code that needs to be reviewed. Deploying with Python is also not ideal for inference performance[128]. Fine-tuning and retraining is another concern, which the 'state dict' format natively supports. Additionally, BioImage.IO should encourage model contributors to share their source code for transparency and to promote further development of contributed models.

For improved deployment PyTorch offers functionality to convert models to a TorchScript code[128]. This conversion can be done either via scripting — "translating" the PyTorch code to TorchScript code — or by tracing — "recording" operations for exemplary inputs. Scripting only supports a subset of operations similar to converting to ONNX, but it is possible to train TorchScript models. Tracing supports a wider range of operations, but traced models cannot be trained and have other disadvantages. For example a lack of generalizability when the model relies on dynamic control flow, because the traced example input follows only one path through the computational graph; i.e with dynamic control flow logic present, operators are chosen based on input values or the input tensor's shape, which might lead to a traced set of operators that is not applicable to a different input. Scripting and tracing PyTorch models can be combined, but this requires manually annotating the PyTorch model source code with the appropriate decorators (special functions in Python code)[128].

TensorFlow offers two alternatives to serialize Python based models: The officially recommended SavedModel format and the Keras H5 format. Both formats save the model architecture and weights. Deserializing the model architecture from the Keras H5 format requires source code for all custom Python objects, whereas the SavedModel format can

include traced TensorFlow subgraphs, which allow deserialization without the original source code. Especially for models without custom layers the Keras H5 format is a lightweight alternative to the SavedModel format. Based on the wide use of these model formats, both are supported by the BMZ. Developments like WebGPU bring DL workloads closer to the browser by allowing to access a system's GPUs efficiently. TensorFlow for JavaScript can run networks directly in the browser if they are saved in a compatible TensorFlow.js[129], which we also support in the BMZ.

NN models describe their in- and outputs only in a rudimentary form. In order for software applications to be able to run different types of models, input and output semantics, but also other properties such as acceptable input tensor sizes, halos to account for or model provenance need to be described in additional metadata. To discover suitable models with user-friendly tools in the first place, model documentation and meaningful annotations are required. With model weights serialized to different file formats that largely do not support additional metadata, we need to use an independent file format for metadata in which the model weights are referenced. In addition to models we also want to describe other resources like datasets and notebooks as a collection of files referenced alongside metadata.

We call a file that holds all metadata of a resource a resource description file (RDF). Different types of resources may specify the metadata in more detail, for example a model and a dataset RDF share a common structure, but also differ in the information they hold.

3.2.1 Selecting a file format

A suitable file format for such resource metadata should be human and machine friendly to read and write to allow for easy editing and processing. For a fast adoption of the BMZ specifications it should be light-weight and ideally already common in the community. To convey the intend of specific aspects of the specification in working examples or template files the ability to include comments may be beneficial. Well documented examples lower the entry requirements needed to contribute to BioImage.IO significantly, which is particularly important for any community driven project. Comments also allow model developers to communicate concisely and conveniently with each other without impairing user experience by mixing metadata with technical details about the format.

As widely used open standard file formats, the JSON[130] or Extensible Markup Language (XML) could be suitable choices. Both of these formats are widespread and well supported by many programming languages and libraries. In comparison to the compact JSON format, XML is more complex and verbose, with data being enclosed in special tags, which makes it harder to read for humans. While XML supports comments, JSON does not. YAML Ain't Markup Language™ (YAML) 1.2[131], a superset of JSON does support comments and is a very human friendly data interchange file format with its indentation syntax reminiscent of Python — the dominant programming language in the DL community, where YAML itself is widely used as a common choice for configuration files. Tom's Obvious Minimal Language (TOML) is as an alternative to YAML. It also supports comments and is human and machine friendly to read and write, but nested data structures are less intuitive compared to YAML's indentation syntax with the option to use compact JSON syntax within YAML interchangeably. While a flat data hierarchy is desirable for simplicity, it might not be attainable to fit a growing amount of detail into a developing format. For these reasons, we, the initial BioImage.IO development team (see section 3.2) — decided to use the YAML format to store and exchange RDFs in. To organize the metadata we chose a mapping of field names to values defined with the (nested) "three basic [YAML] primitives: mappings (hashes/dictionaries), sequences (arrays/lists) and scalars (strings/numbers)"[131].

3.2.2 Defining the BioImage.IO RDF Format

The following paragraphs describe the fields of a model RDF as defined in the Python package `bioimageio.spec` 0.5.3[118] with models in mind as the most central resources of the BMZ. However, many fields described here also serve to represent other resources types such as datasets, notebooks and software applications. The BioImage.IO RDF format also supports linking these resource between themselves to enable FAIR ML. The emphasis here lies on the rational behind adding these fields and not primarily on details of their definitions, which can be found in the `bioimageio.spec` documentation[132].

A developing metadata description format — any metadata format — has to contain a reference to its own version. As BioImage.IO differentiates between different resource types, an RDF's `type` and `format_version` are stored in dedicated fields for this purpose. All specification fields are saved in the RDF yaml file as a mapping, see listing 3.1 for an

example. Model contributors should be able to not only name their creation, but also to provide appropriate levels of detail to describe the domain and task a model is suited for. The mandatory fields `name`, `description` and `documentation` allow to describe what the model (or other resource) is and does in three levels of detail. This allows potential users to gauge the relevancy of a given model quickly and dive into its details when needed. The linked `documentation` file is intended to aid in understanding of potential use-cases. Additionally it may document high level details of the model architecture, the training algorithm, the training data, or any other relevant aspect. While `name` and `description` are given as plain text, we enforce the use of the lightweight markup language Markdown[133, Markdown: Syntax] to format the `documentation` file content, which guarantees that it can be rendered in a wide variety of application environments, yet is easily human-readable as plain text and easy to edit. With these benefits Markdown is already used ubiquitously throughout the community, for example its dialect GitHub Flavored Markdown (GFM) on github.com for documentation and comments[134]. As the de-facto Markdown standard, we allow GFM to be used for `documentation`. Listing A.1 gives an example for the content of a GFM file listed under `documentation`.

```yaml
type: model
format_version: 0.5.3
name: HyLFM-Net-stat
description: >-
  HyLFM-Net trained on static images of arrested medaka hatchling
  hearts. The network reconstructs a volumentric image from a given
  light-field.
documentation: README.md
covers: [cover.png]
# id: 10.5281/zenodo.7614645/7642674  # initial
id: ambitious-sloth  # revised
id_emoji: "\u1F9A5"  # or simply " 🦥 "
version: "1.2"  # revised
```

Listing 3.1: RDF content example for the `name`, `description`, `documentation`, `covers`, `id` and `id_emoji` fields.

An image, famously, is worth more than a thousand words. Therefore, to illustrate the model's capabilities and applicable use-cases, `covers` references image files in either JPEG 1[135], PNG[136], or GIF[137] with the file extensions `` `.jpeg' ``, `` `.png' ``, or `` `.gif' ``, respectively[138, Supported Formats]. This limited set of very common image formats was chosen for simplicity, while avoiding to force contributors to convert existing images to one specific format, i.e. we hope that if a contributor already has suitable images these do not have to be converted in most cases. Contributors were asked to include cover images that are no larger than 500 kB to ease browsing of the BMZ and recommend an aspect ratio of 2:1 for these images to promote a uniform model representation. The 2:1 aspect ratio is well suitable for showcasing one square input image side-by-side with its output. Not all contributors were aware of these recommendations, which is why I later implemented the generation of thumbnail images in the collection management software, see section 3.3. This allows contributors to upload high-quality cover images, while browsing the BioImage.IO Collection can still be blazingly fast.

The `name` field of a model is not required to be unique and minor updates to it are conceivable. In order to uniquely identify models and versioning them, an `id` field is needed. Initially it held a version specific identifier, which included a version unspecific part always referencing the latest model version. This was inspired by Zenodo's record versioning with a concept Digital Object Identifier (DOI)[139] and multiple version specific DOIs[140]. With the BioImage.IO models initially hosted exclusively on zenodo.org (see section 3.3), a model's generated `id` was composed of the Zenodo concept DOI and the version specific DOI without the redundant Zenodo DOI prefix, which is already contained in the `id` as part of the Zenodo concept DOI. In listing 3.1 "10.5281/zenodo.7614645" corresponds to the Zenodo concept DOI, which is merged by a forward slash with "7642674" — the last part of the version specific DOI "10.5281/zenodo.7642674". I revised this rather complicated versioning scheme by making a previously optional `version` field following Semantic Versioning Specification (SemVer) 2.0[141] mandatory. Now the `id` field is intended to stay unchanged for minor updates to the model — like a revised description — that do not alter its logic — like changing its model weights. This `version` field should not be confused with the `format_version` field. BioImage.IO compatible Ids follow specific pattern (see section 3.3.5 *Memorable Resource Identifiers*), which can be matched with an associated value for the `id_emoji` field that adds a friendly detail for display — for example the sloth Unicode character  for any "-sloth"

models.

Multiple individuals may be credited for their contributions to a BioImage.IO model: The `authors` field lists all primary contributors to the model. If others contributed to the model's metadata and combined it with existing model weights, these are listed under `packaged_by`. Whoever takes on the responsibility of maintaining the model may be listed under `maintainers`, if not equal to `author`, who have the maintainer role by default. For all maintainers `github_user` needs to be specified, which allows to notify them via mentioning on . In the example listing 3.2 `github_user` could also be specified under `authors`. Multiple `weights` (in the formats described above) may be linked, each associated with their own `authors` if different from the base `authors` mentioned above. All authors must agree on a `license` from the Software Package Data Exchange (SPDX) License List[142]; during validation (see section 3.2.4) a warning is generated if this license is not listed as a free license by the Free Software Foundation (FSF)[143] to encourage openly sharing resources.

```yaml
authors:
- name: Fynn Beuttenmueller
  affiliation: EMBL Heidelberg
  orcid: 0000-0002-8567-6389
  github_user: thefynnbe  # needed if there is no dedicated maintainer
# maintainers:  # if someone other than an author takes over
  ↪ maintenance
# - name: ***
#   github_user: ***
# packaged_by: ***  # not specified if identical to 'authors'
license: MIT
```

Listing 3.2: RDF content example for `authors`, `maintainers`, and `license`.

As shown in listing 3.3 multiple weight formats may be listed under `weights`. Each `weights` entry has a `source` field and an associated Secure Hash Algorithm 2 (SHA-2) hash value, a SHA-256 value in particular, stored in `sha256`. For simplicity the specification does not support arbitrary hash algorithms, but only the SHA-256 — a common and sufficiently secure choice. One of the `weights` entries (here: `pytorch_state_dict`)

is the original weights format of the model. All other sets of model weights are derived from these original weights — directly or indirectly — and should specify their `parent` (here: `pytorch_state_dict`). As stated in section 3.2 for PyTorch 'state dict' weights (`pytorch_state_dict`) the model architecture needs to be stored in addition to the learned weights. To this end `architecture:source` points to a class/function name in a Python file, whose integrity can be verified with a SHA-256 hash value stored in `architecture:sha256`. If the `pytorch_version` or `tensorflow_version` is insufficient to define a suitable Python environment, `dependencies` can be used to reference a conda[144] environment file.

```yaml
weights:
  pytorch_state_dict:
    source: weights.pt
    sha256: >-  # SHA-256 of weights.pt
      461f1151d7fea5857ce8f9ceaf9cdf08b5f78ce41785725e39a77d154ccea90a
    architecture:
      callable: HyLFM_Net
      kwargs:
        c_in_3d: 64
        c_res2d: [768, 768, 512, 256, u384, 512, 256]
        c_res3d: [32, u16, 8, 8]
        init_fn: xavier_uniform
        kernel2d: 3
        kernel3d: 3
        last_kernel2d: 5
        nnum: 19
        z_out: 49
      source: model.py
      sha256: 14e7777576ed54dbf8614a461465f02a9dca99d882878ac35...
    dependencies:
      source: environment.yaml
      sha256: e0c059d829fa03193eede76961746f464ac9b07d072b1e6ee...
    pytorch_version: 1.13  # may not collide with environment.yaml
  torchscript:
    source: weights_torchscript.pt
    sha256: ec01e0c212b5eb422dda208af004665799637a2f2729d0ebf...
    pytorch_version: 1.13
    parent: pytorch_state_dict
  onnx:
    opset_version: 15
    source: weights.onnx
    sha256: 55ad061651840fe4b6da6733c52402ae3296392f74fed8a94...
    parent: pytorch_state_dict
    # authors: who converted the weights from the parent weights
    #          (if different from general 'authors')
```

Listing 3.3: RDF content example for weights.

An example for a conda environment file is given in listing 3.4. Note that pip installs packages (pip) Python requirements[145] (a very common format for Python dependencies) can be specified within a conda environment[146]. In such a suitable Python environment the model's class/function can be imported and called with additional key word arguments (`kwargs`). The weights then can be loaded into the returned NN model: ready for inference or fine-tuning. In addition to `pytorch_state_dict`, `torchscript` (TorchScript), and `onnx` (ONNX) entries, weights may be specified under `tensorflow_saved_model_bundle` in the TensorFlow SavedModel format), `keras_hdf5` for Keras H5 weights, or `tensorflow_js` for weights directly supported by TensorFlow for JavaScript.

```yaml
name: hylfm

channels:
- pytorch
- conda-forge

dependencies:
- python >=3.7,<3.8
- pytorch >=1.7,<1.8
- inferno ==v0.4.2  # non-standard dependency for PyTorch models
```

Listing 3.4: Content of the conda environment file "environment.yaml" as specified under `dependencies` in listing 3.3. Details of this format can be found in the conda documentation: *Managing environments: Sharing an environment*[146].

Often models build on or are part of published work, which should be cited when a model is used in a publication. These references can be listed under `cite`.

```yaml
cite:
- text: >-
    Beuttenmueller, Wagner, N., F., Norlin, N. et al.
    Deep learning-enhanced light-field imaging with continuous
    ↪ validation.
    Nat Methods 18, 557-563 (2021).
  doi: https://doi.org/10.1038/s41592-021-01136-0
git_repo: https://github.com/kreshuklab/hylfm-net
training_data:
  id: 10.5281/zenodo.7612115
# parent: <parent id>  # no parent, model was trained from scratch
links:
- imjoy/BioImageIO-Packager  # for downloading model package
# attachments:
#   files: [<file URI>]  # included when packaging the model
# run_mode: name: deepimagej
# icon: <URI to resource specific icon>
# badges: [icon: <>, label: <>, url: <>]
```

Listing 3.5: RDF content example for various references.

A relevant Git[147] repository specifically can be linked under `git_repo`. This repository should be the location where the model is developed. `training_data` optionally references the data used to train the model. This training data is either described as an in-place dataset RDF or referenced by it's `id` subfield. A dataset RDF is simply a general RDF of `type` "dataset". If the model was derived from another, `parent` holds the `id` of that other model. `links` holds references to other related BioImage.IO resources. Like the example in listing 3.5 all models link the BioImage.IO Packager, which allows to download them as a model package — a ZIP archive containing the RDF and all files it references. A URL pointing to a (prepackaged) resource may be inserted under `download_url`. To decrease the file size the BioImage.IO Packager allows to select one of the specified weight formats to only include one set of weights in the desired format. Model packaging is described in more detail in section 3.2.4.

Additional files that need to be included in the resource package can be listed under

`attachments:files`. These may be locally linked images in the `documentation` markdown file. So far the most commonly attached files are ImageJ Macro language (IJM) scripts for pre- and postprocessing, as the `bioimageio.core` Python library cannot directly be used in JavaScript. For standard pre- and postprocessing operations (see section 3.2.2) this is an interim solution. As the pre- and postprocessing operations are now supported by the JDLL, future models will not have to include these macros for standard pre- and postprocessing operations to be compatible with community partner software written in Java/JavaScript. The standardized pre- and postprocessing operations are a fixed set of operations for which language cross-compatibility is feasible. To make custom processing steps, outside of this set, available to BioImage.IO compatible software I developed `bioimageio.workflows`, see section 3.5.

Special circumstances to consider during model inference should be marked with the `run_mode` field, which specifies a `name` and any associated key word arguments under `kwargs`. The field remains largely unused. Its only use-case thus far has been to indicate the presence of ImageJ macro scripts for custom pre- or postprocessing that only the DeepImageJ plugin is automatically executing. Therefore, with the same model, other community partner software likely return different results. Respective models may still be used in other community partner software, but at least a warning should be issued by that software, if the `run_mode` is not recognized. The model's `documentation` should be consulted in these cases.

The `icon` field allows to specify a resource specific icon. This icon is intended primarily for application descriptions, as models, datasets and notebooks typically do not have a dedicated icon (except for the randomly assigned `id_emoji`). On bioimage.io these icons are used to display linked (application) resources, see figure 3.2. To highlight external links in a similar manner `badges` can be used. Each entry in `badges` has its own `icon` field, a text `label`, and a target `url`. For example, notebook descriptions may include an "Open in Colab[148]" badge.

To keep track of the origin of an RDF, initially an `rdf_source` field was used to save the where an RDF was loaded from. The revised `bioimageio.spec` library attaches a validation summary to the Python representation of an RDF instead that includes the RDF source. This avoids mixing temporary/context dependent information with persistent metadata.

For improved model provenance the time of model creation should be recorded. The

`timestamp` field records the time following the ISO standard number 8601[149].

```yaml
timestamp: 2021-02-11T11:40:46Z
tags:
- light-field-microscopy
- pytorch
- fluorescence-light-microscopy
- image-reconstruction
- nuclei
- hylfm
config:
  bioimageio:  # BioImage.IO specific fields
    thumbnails: { cover.png: cover.thumbnail.png }
    created: '2023-02-15 09:34:43.433531'
    doi: 10.5281/zenodo.7642674
    nickname: ambitious-sloth  # initial
    nickname_icon: "\u1F9A5"  # or "🦥"  # initial
    owners: [96309]   # initial
    status: accepted  # initial
    version_id: '7642674'    # Zenodo version specifc  # initial
    version_name: version 2  # Zenodo version name  # initial
```

Listing 3.6: RDF content example for `timestamp`, `tags` and `config`.

To improve search results `tags` holds a list of relevant keywords. We define groups of keywords, for example 'task' or 'modality', to assist developer in tagging their models comprehensively. These groups are not fixed and used only to generate hints in the form of warnings during validation (see section 3.2.4). The same keyword groups are use on the bioimage.io website to display available tags in groups.

> ... But if you think of standards as confining, then progress stops.
>
> (Henry Ford, Today and Tomorrow, 1926)

In case the fields defined in the specification fall short of describing a given resource, `config` allows to store arbitrary (YAML compatible) data. This mainly serves two purposes: to store software specific data and to develop new, experimental fields. We allow to store software

specific data for cases where a model/resource can be used across BioImage.IO's partner applications, but also benefits from additional input for specific software environments. This liberty eliminates the need for separate, more specific model collections within the BioImage.IO community. At the same time some of these specific fields may be adopted across the BMZ. Such fields can then be formalized and moved out of `config` in an updated RDF specification with an increased `format_version` following SemVer 2.0. Of course the `config` field also allows to develop new, experimental fields that are not software specific. In the context of the BioImage.IO Collection, specific fields are added to `config:bioimageio`. Initially, a model nickname and icon were added there automatically for resources accepted into the BMZ via Zenodo as described in section 3.3.1. This mechanism has since changed; `nickname` and `nickname_icon` were adopted into the general RDF specification as `id` and `id_emoji`.

Besides `weights`, `training_data`, `timestamp`, and `run_mode` the above mentioned spec fields also apply to describe a general, non-model resource. For general resources `source` is an additional optional field to store a URL to the actual resource described by the RDF. A public download link for a dataset, for example, or a Jupyter Notebook[150] for notebook descriptions. The `source` field is not allowed in a model RDF to make the model RDF format more concise and avoid confusion with the `source` fields nested under `weights` or `architecture`.

In addition to a specific model RDF, which is further described in the following section 3.2.2, I defined a collection RDF specification to validate partner collections and a workflow RDF specification to use in the context of `bioimageio.workflows` (see section 3.5). We also explicitly define a dataset RDFwhich, as mentioned above, does not differ from the general RDF, but is referenced in both, the model RDF specification, and on the bioimage.io website.

More model specific fields

Understanding the inputs and outputs of a model is crucial for its correct deployment. Also from a technical perspective the model's interface needs to be defined clearly to integrate a model into an analysis workflow or specialized software. Therefore, `inputs` and `outputs` are described in detail as part of the model RDF. Both descriptions con-

sist of lists of n-dimensional arrays, or "tensors" — the central data structure in image processing. The `inputs` and `outputs` tensor descriptions have some fields in common: a unique `name` for referencing, a `description`, `axes` from a predefined set to annotate the tensor dimensions, expected `data_type`, and `data_range` — the tensor's minimum and maximum values. Both tensor types also define a `shape` field, which either holds a fixed shape — the length of each tensor dimension — or a variable shape: Input tensor shapes may be parametrized with a minimal shape `min` and a `step`, such that $\texttt{input.shape} = \texttt{min} + k \cdot \texttt{step}\ \forall k \in \{0, 1, ...\}$. Output tensor shapes on the other hand may be defined by applying a `scale` and an `offset` to the shape of a `reference_tensor`, such that $\texttt{output.shape} = \mathit{shape}(\texttt{reference\_tensor}) \cdot \texttt{scale} + 2 \cdot \texttt{offset}$.

```yaml
inputs:
- name: lf
  description: raw light-field
  axes: bcyx
  data_type: float32
  data_range: [-.inf, .inf]
  shape: [1, 1,  1235, 1425]
  preprocessing:
  - name: scale_range
    kwargs:
      mode: per_dataset
      axes: yx
      min_percentile:  5.0
      max_percentile: 99.8

outputs:
- name: prediction
  description: predicted fluorescent volume
  axes: bczyx
  data_type: float32
  data_range: [-.inf, .inf]
  shape: [1, 1, 49, 244, 284]
  halo: [0, 0, 0, 0, 0]
  # postprocessing:  # postprocessing example
  #  - {name: binarize, theshold: 0.5}

test_inputs: [test_input.npy]
test_outputs: [test_output.npy]

sample_inputs: [sample_input.tif]
sample_outputs: [sample_output.tif]
```

Listing 3.7: RDF content example for model 0.4 fields (`test_`/`sample_`)`inputs` and (`test_`/`sample_`)`outputs`. An updated version of this example is shown in listing A.2.

Output tensors have a `halo` field to indicate how much of their output might be effected

by edge artifacts. Such border effects can occur due to "same" padding — virtually enlarging the input image to compute output pixels at the image border, even though they dependent on non-existing neighbor pixels. Alternatively some NN architectures prefer the use of "valid" convolutions — convolutional layers that do not compute output values at the image border, thus returning a smaller output compared to their input. This can be reflected by the above mentioned `offset` field specified under `shape` for any output tensor.

Although NNs are universal function approximators[1] and all DL frameworks support basic tensor operations, `preprocessing` and `postprocessing` are best separated from the NN architecture. This is common practice in the field for several conceptual and practical reasons. In this context image pre- and postprocessing consists of tensor operations without learnable parameters. Such operations are typically computed on central processing unit (CPU) cores in parallel for multiple samples. Preprocessed samples are then stacked to mini-batches, which are send to a GPU (or TPU), ready for processing by an NN, which — in simple terms — greatly benefits from the speedup of matrix multiplication on GPU/TPU hardware. Postprocessing can follow again efficiently on the CPU, freeing up the specialized hardware for the succeeding mini-batch. Aside from this division of labor pre- and postprocessing may also be configured independently of the NN, for example to adapt to a new data source (also without transfer learning or fine-tuning) or simply to compute derived outputs by adding postprocessing steps, for example to binarize output probabilities.

We limit the set of pre- and postprocessing operations that we incorporate into the model RDF specification for practical purposes: We need reference implementations, ensure equivalent implementations across officially supported libraries and maintain these implementations jointly. This restriction is alleviated with `bioimageio.workflows` that I describe in section 3.5. Any entry of `inputs` may specify a list of `preprocessing` steps, while every entry of `outputs` specifies a list of `postprocessing` steps analogously. An overview of supported operations is given in section 3.2.2.

Corresponding to these `inputs` and `outputs` `test_inputs` and `test_outputs` hold a mini-batch with test data and expected outputs. Each tensor of this test data is saved in the NumPy[151] `.npy` format, which is the canonical choice in a scientific Python environment and simple to read with other languages[152]. The test inputs and outputs constitute a test case for programmatic validation. To provide example data for users to interact with a given model, `sample_inputs` and `sample_outputs` hold a list of local file paths or URLs

to image files corresponding to the model's in- and outputs. These sample files may be stored in various common image formats, for example TIFF, PNG, or HDF5. They should be aligned, but may be larger than what the model can process at the same time (require tiling, see section 3.4.1).

`name` arguments	description
`binarize`	Binarize the tensor with a fixed threshold, values above the threshold will be set to one, values below the threshold to zero.
`threshold`	The fixed threshold.
`clip`	Set tensor values below.`min` to `min` and above `max` to `max`.
`max`	Maximum value for clipping.
`min`	Minimum value for clipping.
`scale_linear`	Fixed linear scaling of the value range.
`axes`	The subset of axes to scale jointly. For example 'xy' to scale the two image axes for 2d data jointly. The batch axis 'b' is not valid here. Default: scale all non-batch axes jointly.
`gain`	multiplicative factor. Default: 1.0.
`offset`	additive term. Default: 0.0.
`scale_mean_variance`	(Only available in postprocessing) Scale the tensor s.t. its mean and variance match a reference tensor.
`mode`	Either 'per_dataset' (mean and variance are computed for the entire dataset) or 'per_sample' (mean and variance are computed for each sample individually).
`reference_tensor`	Name of tensor to match.
`axes`	The subset of axes to normalize jointly. For example 'xy' to normalize the two image axes for 2d data jointly. The batch axis 'b' is not valid here. Default: scale all non-batch axes jointly.

Table 3.1: Pre- and postprocessing operations for model RDF specification 0.4.9. Reference implementation are part of `bioimageio.core` (section 3.4)[122]. Adapted from [153, 154]. Continues on next page.

Continued from previous page

name arguments	description
eps	Epsilon for numeric stability: out = (tensor − mean)/(std + eps). Default: 10^{-6}.
scale_range	Scale with percentiles.
mode	One of 'per_dataset' (mean and variance are computed for the entire dataset), 'per_sample' (mean and variance are computed for each sample individually).
axes	The subset of axes to normalize jointly. For example 'xy' to normalize the two image axes for 2d data jointly. The batch axis 'b' is not valid here. Default: scale all non-batch axes jointly.
eps	Epsilon for numeric stability: out = (tensor − mean)/(std + eps). Default: 10^{-6}.
max_percentile	The upper percentile used for normalization, in range 1 to 100. Has to be bigger than min_percentile. Default: 100. The range is 1 to 100 instead of 0 to 100 to avoid mistakenly accepting percentiles specified in the range 0.0 to 1.0.
min_percentile	The lower percentile used for normalization, in range 0 to 100. Default: 0.
reference_tensor	(Only available in postprocessing) Tensor name to compute the percentiles from. Default: The tensor itself. In 'per_dataset' mode this needs to be the name of an input tensor.
sigmoid	The logistic function $S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} = 1 - S(-x)$.

Table 3.1: Pre- and postprocessing operations for model RDF specification 0.4.9. Reference implementation are part of `bioimageio.core` (section 3.4)[122]. Adapted from [153, 154]. Continues on next page.

Continued from previous page

name arguments	description
`zero_mean_unit_variance`	Subtract mean and divide by variance.
`mode`	Either 'fixed' (fixed values for mean and variance), 'per_dataset' (mean and variance are computed for the entire dataset), or 'per_sample' (mean and variance are computed for each sample individually).
`axes`	The subset of axes to normalize jointly. For example 'xy' to normalize the two image axes for 2d data jointly. The batch axis 'b' is not valid here. Default: scale all non-batch axes jointly.
`eps`	Epsilon for numeric stability: `out` = (`tensor` − `mean`)/(`std` + `eps`). Default: 10^{-6}.
`mean`	The mean value(s) to use in 'fixed'. For example '[1.1, 2.2, 3.3]' in the case of a 3 channel image where the channels are not normalized jointly.
`std`	The standard deviation values to use in 'fixed' mode. Analogous to mean.

Table 3.1: Pre- and postprocessing operations for model RDF specification 0.4.9. Reference implementation are part of `bioimageio.core` (section 3.4)[122]. Adapted from [153, 154].

3.2.3 Use of the RDF initialism

The early decision to adopt the initialism RDF for resource description file and its use as a canonical file name 'rdf.yaml' in the BMZ clashes with the Word Wide Web Consortium (W3C) recommendation "Resource Description Framework (RDF): Concepts and Abstract Syntax"[155] and can lead to confusion. I therefore started to refer to our RDFs as the 'bioimageio.yaml' file instead. This expression is explicit and bioimage.io specific, which avoids any collisions with common terms — current or future — and is more easily comprehensible than

an initialism. To spread this change throughout the BMZ while keeping community partner software backwards compatible is an ongoing effort. In the interim period all community partner developers are encouraged to implement forward compatibility, for example already searching for the 'bioimageio.yaml' file in a folder opened by a user.

3.2.4 The `bioimageio.spec` Python library

To contribute models and other resources to the BMZ, users are required to construct an RDF following the BioImage.IO specification. This extensive and evolving specification can be difficult to follow and fully take advantage of when constructed manually. The `bioimageio.spec` Python library described in this section is the first in a series of Python packages to address these difficulties; `bioimageio.core` (section 3.4.1) and `bioimageio.workflows` (section 3.5) continue this series.

In addition to defining the BioImage.IO RDF specification, `bioimageio.spec` provides means to validate an RDF. It performs *static* validation — checking if the RDF's metadata follows the applicable format specification, for example that mandatory fields are present and have an appropriate data type. This excludes *dynamic* validation, such as validating the model weights with provided test inputs and test outputs, but does include validating file hashes and pinging specified URLs to report on their availability (this can be disabled to stay offline). *Dynamic* validation of inference reproducibility is provided by `bioimageio.core` as described in section 3.4.2. Validating an RDF with `bioimageio.spec` returns concise information on any format errors and applicable warning messages. These errors and warnings are referencing specific RDF fields for precise feedback. `bioimageio.spec` is the single source of truth for the RDF specification. The github.com/bioimage-io/spec-bioimage-io GitHub (GH) repository where it is developed also generates documentation, not only for developers and users of the `bioimageio.spec` Python library, but also for any user interactions with RDF metadata outside of Python.

In the following paragraphs I present selected implementation details that are at the heart of `bioimageio.spec`. Before discussing implementation details, the programming language needs to be mentioned. Why is `bioimageio.spec` written in Python? Python is the most popular language within the DL community and very popular throughout the scientific community at large. It is beginner-friendly, which makes it an ideal choice for

interdisciplinary development in a scientific environment with many junior developers. Thus there is a good chance to continue to find sufficient expertise in the BioImage.IO community to keep developing and maintaining such a Python library. Finally, using Python allows to integrate nicely with common DL frameworks, namely PyTorch and TensorFlow, which are most commonly used through their respective Python Application Programming Interfaces (APIs). I designed the `bioimageio.spec` library to be compact and focused on its two main tasks: defining and validating the RDF specification. A clear API and dedicated representations for every element in the specified RDF metadata aim to make it a powerful, but easy to incorporate library to programmatically process BioImage.IO resource descriptions. To integrate well with existing Python projects it has only few, mostly common, dependencies.

To load and save YAML files I chose the `ruamel-yaml`[156] Python library, because it supports the latest YAML 1.2[131] and allows to preserve YAML comments and formatting. With the focus on validation of RDFs, Maksim Novikov and I decided to utilize the `marshmallow`[157] Python library, which is a popular choice for validation, serialization, and deserialization in Python. The `marshmallow` framework provides functionality for concise validation without a lot of boilerplate code. Important for the purpose of the BMZ: it provides the flexibility to add custom validation functions. This is important to ensure high quality contributions to the BMZ with minimal supervision by BioImage.IO reviewers. The `marshmallow` library is also very light-weight — it only depends on the ubiquitous Python "packaging" package. This is beneficial as it allows our `bioimageio.spec` library to be added to other Python projects without causing dependency conflicts.

To make `bioimageio.spec` the single source of truth of the RDF specifications, I added functionality to generate up-to-date documentation from the very same code responsible for validation.[1] This allows to programmatically document default values and descriptions of defined fields where their validation is implemented. This ensures keeping documentation and validation in sync. I also added *Pythonic* representations of RDFs in the form of nested Python Data Classes[158, dataclasses] to work with RDFs in Python code, see section 3.4. This allows developers of community partner software to integrate the RDF specification efficiently.

RDFs fields are defined in in nested `marshmallow` schemas. These schemas are Python

[1] An improved script for generating more user-friendly documentation was contributed by Tomaz Vieira.

classes, which provide a framework to formalize serialization of custom Python objects to a set of native Python data types and vice-versa (deserialization). This set of native Python data types can be further serialized to file formats like JSON or YAML in a straight-forward manner. To describe BioImage.IO models (or other resources) I defined custom Python objects (based on Python Data Classes), which I refer to as *raw resource descriptions*. When `bioimageio.spec` is used to deserialize a BioImage.IO model a schema is chosen based on the value of the `type` and `format_version` stated in the RDF. If the RDF content is not given as a Python dictionary, the RDF source is analyzed and interpreted, downloaded, and/or unzipped as required and then loaded using `ruamel-yaml`[156]. This allows to reference the BioImage.IO model in the examples from listings 3.1 to 3.7 in various ways:

- `id`: 'ambitious-sloth'
- concept DOI: '10.5281/zenodo.7614645'
- version DOI: '10.5281/zenodo.7642674'
- RDF path/URL: 'https://.../rdf.yaml'
- package path/URL: './downloads/hylfm-net-stat_onnx.zip'.

Validation of an RDF is performed during its deserialization or programmatic creation. Using `marshmallow` an error report is generated in form of a nested dictionary with error messages associated to their respective RDF fields. In addition I added functionality to generate a warning report of the same form to hold warnings or hints depending on RDF content. These reports are extended with the `bioimageio.core` package (section 3.4) and inspected during incorporation of new resources into the BMZ as described in section 3.3. Upon successful loading of an RDF, the returned Python object is a fully type annotated Data Class instance, which allows for *Pythonic* use and excellent integrated development environment (IDE) support, for example auto-completion of attribute/field names.

Each dedicated RDF type has its own schema and Python Data Class. For serialization of a resource description Python object, the corresponding RDF schema is used to serialize to native Python data types with `marshmallow`, and — if desired — the resulting nested Python dictionary is serialized to string/disk using as YAML file.

Meanwhile, I have integrated the `marshmallow` based validation code and the *Pythonic* Data Class representations by switching from `marshmallow` to the `pydantic`[159] Python

library. This simplified the code base, improved type annotations and adds additional `pydantic` based features such as an improved[2] generation of a corresponding JSON Schema — a limited representation of the RDF specification defined by `bioimageio.spec`, but a compatible subset that allows to use a wider range of established tools. The generated (and continuously updated) BioImage.IO JSON Schema has been registered with the JSON Schema store[161] — a collection of popular JSON Schemas. This collection is utilized by various software programs to validate file content with. Various IDEs for example generate highlights and hover texts in applicable files based on these schemas, much like grammar checkers in everyday text editors. This made editing a `bioimageio.yaml` file in an IDE even easier!

I added a Command-line interface (CLI) — initially implemented with typer[162]. The CLI supports various commands to validate and update RDFs. RDF updates of the format version are aided by auto-conversion rules. Additional update functionality allows for partial updates specified by the user. The bioimageio CLI makes it possible to use `bioimageio.spec` outside of Python code as a standalone tool, improving its accessibility. The revised BioImage.IO CLI is moved to `bioimageio.core`, which previously extended its functionality. It is now based on to the CLI features recently added to the `pydantic` library further stream-lining and modernizing the `biomageio.core` code base.

Packaging BioImage.IO models

The RDF specification also determines which files to include in a model package. A model package is simply a ZIP archive with an rdf.yaml/bioimageio.yaml file and all relevant files, like the linked `documentation` Markdown file and, of course, serialized model weights in one or more framework specific formats. A model package allows community members to make use of previously downloaded models and aids reproducibility. Packaged models also enable users to use BioImage.IO models independently of a stable internet connection or completely offline and facilitate sharing of BioImage.IO compatible models independently of central services. While the basic packaging functionality is implemented in `bioimageio.spec`, the *bioimageio-package* CLI command is implemented in `bioimageio.core` (with all BioImage.IO CLI commands). When packaging a model with

[2] Previously I generated a less extensive JSON Schema with the `marshmallow-jsonschema` library[160], which is no longer actively developed.

the BioImage.IO CLI is is subsequently also tested *dynamically* to ensure full compliance with the BioImage.IO specificationssections 3.4.2 and 3.4.3.

3.3 Content of the BMZ: The BioImage.IO Collection

Content and implementation of the bioimage.io website are separated. The BioImage.IO Collection denotes the content shared by community members — it consists of all resources uploaded to the BMZ and displayed on the website `https://bioimage.io`. These resources are DL models, Jupyter[150] notebooks, (BioEngine) applications and descriptions of datasets.

I am the main author of github.com/bioimage-io/collection-bioimage-io[116], which served as the initial platform to manage the BioImage.IO Collection at the time of publishing the W. Ouyang*, F. Beuttenmueller*, E. Gómez-de-Mariscal*, C. Pape* et al. [107] preprint. In this section I will first describe this initial setup while mentioning select differences to a revised implementation of the Collection management software hosted and developed at github.com/bioimage-io/collection[117]. In section 3.3.4 I will further motivate and highlight these changes.

We, the BioImage.IO developers — in the context of the initial BioImage.IO Collection: Fynn Beuttenmüller, Wei Ouyang, Constantin Pape, Estibaliz Gómez-de-Mariscal, and Carlos Garcia-López-de-Haro[3] — initially decided to provide two ways in which to contribute resources: Through Zenodo — a well established open science data repository, ensuring data persistence through DOIs — and through GitHub (GH) — for accelerated resource development. Creating Zenodo records for BioImage.IO resources was made more convenient through the BioImage.IO upload interface[115]. While the wider community was invited to contribute via Zenodo, contributing via GH was reserved for trusted community partners only.

The BioImage.IO upload interface includes metadata validation with `bioimageio.spec` (described in section 3.2.4), more thorough testing (with `bioimageio.core`; section 3.4) of contributed models is required to ensure high quality, ready-to-use models. At the same time, the manual revision burden on BioImage.IO developers in their role as maintainers needs

[3] involved in the discussions were also Anna Kreshuk, Florian Jug, Arrate Muñoz-Barrutia, Tom Burke, Maksim Novikov, and Daniel Sage.

to be kept to a minimum. To make reviewing of potential resource contributions efficient, I implemented model testing with Continuous Integration and Continuous Deployment (CI/CD) mechanisms — a series of Python scripts executed in GitHub Actions workflows[163, GitHub Actions] that I describe in next section.

3.3.1 (Initial) Automation with CI/CD

CI/CD is the common practice of automated testing, merging, and deployment of changes in a software project. CI: Every proposed change is tested such that accepted changes to the main code base keep the software in an ever-changing, but functional state. CD goes one step further and also automatically deploys the successfully tested main version of the code to a production environment — the software is continuously deployed. The default mechanism to implement CI/CD for software projects hosted on GH are GitHub Actions workflows — YAML files describing a set of GitHub Actions[163, GitHub Actions]. Each GH Action is a small program, for example a predefined routine to checkout ("download") a project's repository or a custom script.

The update collection workflow [4] was the BioImage.IO Collection's primary GitHub Actions workflow that delegated updating, testing, and deployment of the BioImage.IO Collection. As illustrated in figure 3.4 its execution was triggered by four types of events:

1. Any changes to the code base — a push to main, for example a new PR merge commit by a BioImage.IO Collection developer.
2. The dispatch event could trigger a *collection update* through a webhook. This was used by external tools like the bioimage.io upload interface when a contributor uploaded a new resource version to Zenodo where the subsequent *update collection* detected it. The same dispatch event could be triggered manually through the GH browser interface.
3. For relevant Zenodo records (records tagged with 'bioimage.io') that were edited directly on zenodo.org, a *collection update* was triggered regularly by a schedule. These regular updates also ensured that updates to community partner collections were incorporated in a timely manner if the community partner did not use the

[4] highlighted text refers to figures 3.4 and 3.5

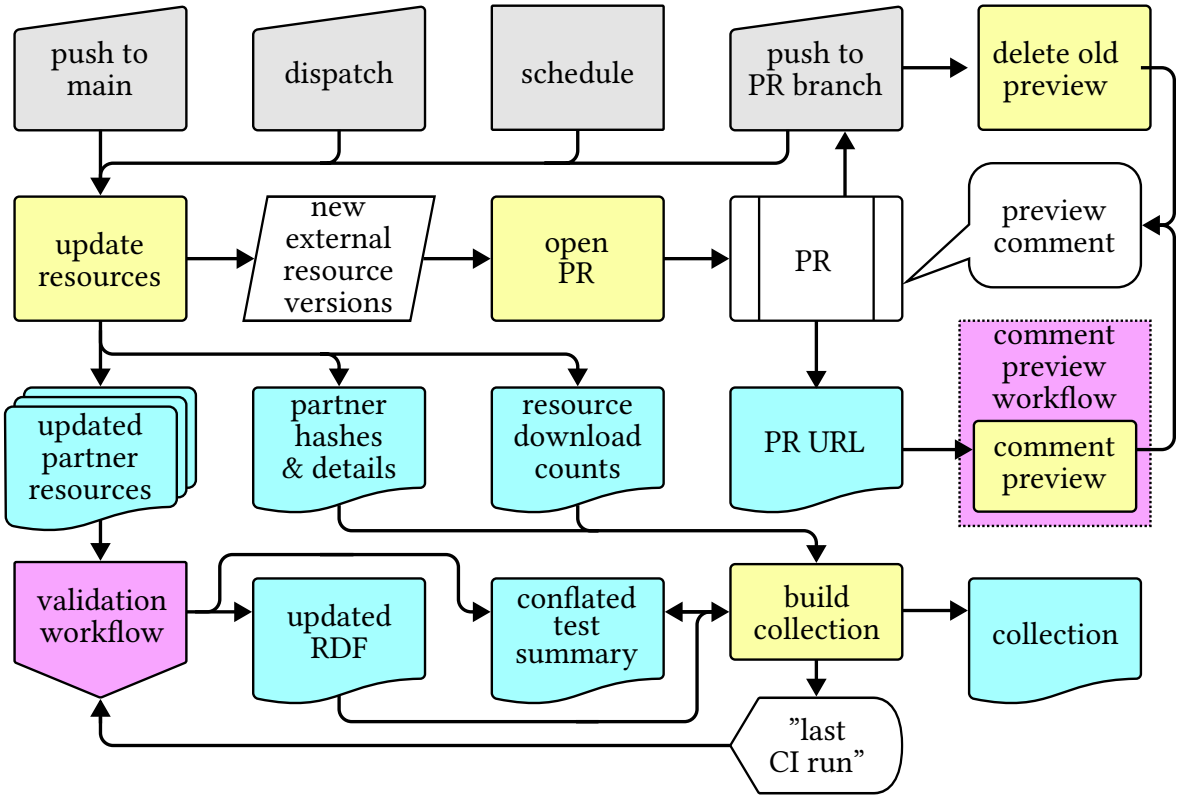


Figure 3.4: Overview of the initial BioImage.IO Collection's update collection GitHub Actions workflow. Its main functionalities were updating, testing, and deploying the BioImage.IO Collection.
An overview: update resources is triggered by different events. When it ran it detected new external resource versions from Zenodo and opens a PR for each.
The validation workflow (see figure 3.5) generates an updated RDF and an associated conflated test summary. New resources versions are given by updated partner resources and by comparing the active commit with the commit tagged after the successful last CI run.
All output documents are either deployed via the *gh-pages* branch or saved as GH artifacts. These artifacts are linked by the comment preview workflow in a preview comment to the respective PR.
Delete old preview removes outdated preview comments automatically.
Finally build collection generates the collection JSON file that lists all BMZ resources with a few details and (if operating on the main branch) tags the current commit as last CI run.

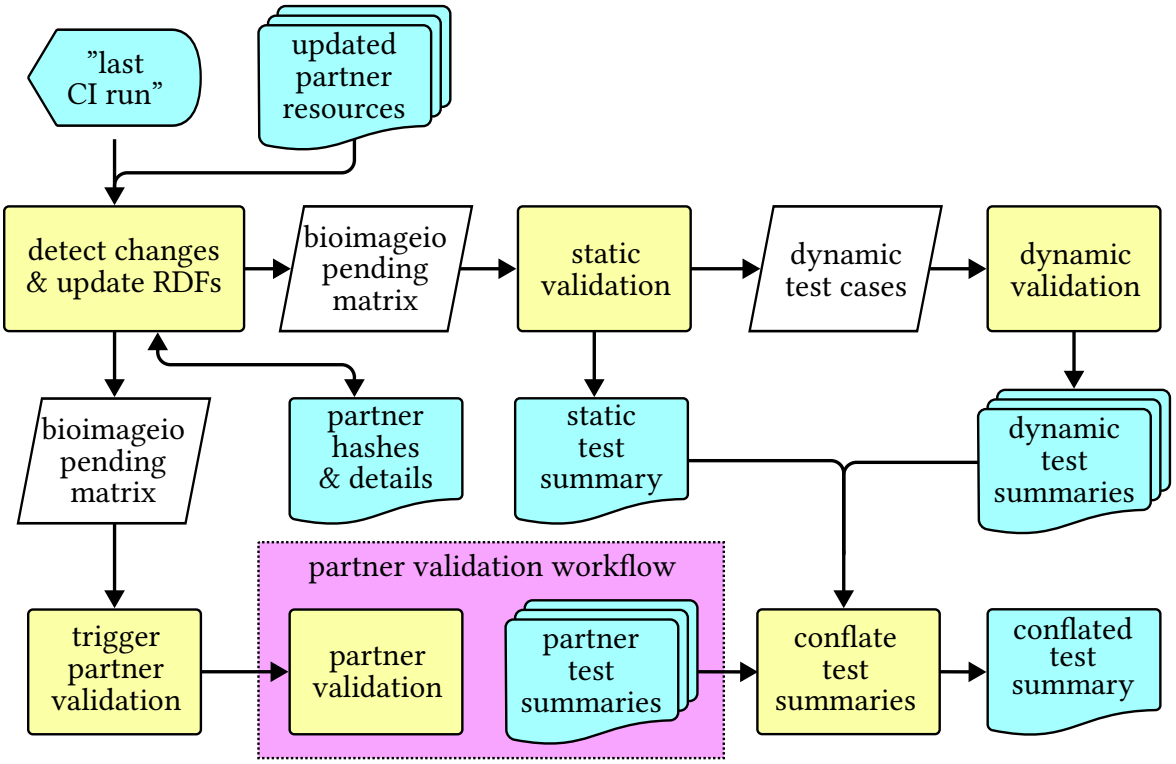


Figure 3.5: (Initial) GitHub Actions workflow to automatically validate contributed resources. The detect changes & update RDFs assembles the bioimageio pending matrix by comparing to the "last CI run", processing the updated partner resources and finding new external resource contributions through the Zenodo API. The bioimageio pending matrix is used by the static validation job to generate dynamic test cases for the dynamic validation job. The bioimageio pending matrix is also send to partner validation workflows via dispatch events triggered by the trigger partner validation job. All validation jobs generate static/dynamic/partner test summaries, which conflate test summaries joins to incorporate them as a single conflated test summary per model in the BioImage.IO Collection.

dispatch mechanism.

4. When proposing code changes or an (automatic) addition of a new resource (version), a push to PR branch also triggered the *collection update*. In this case the workflow was aware that it was operating on a PR branch and did not push any of the output documents to the *gh-pages* branch for deployment, but instead created preview artifacts. An independent comment preview workflow created a preview comment with download links to these artifacts.

The update collection workflow's update resources job (figure 3.4) detected new external resource versions hosted on Zenodo that were tagged with 'bioimage.io' and opened a maximum of 20 concurrent PRs. For resources hosted by community partners the update resources detected updated partner resources by comparing partner hashes (SHA-256) for the respective partner collection RDFs and saved the partner details for the later build collection step to reduce the recurrent number of calls to the GH API (which is rate-limited[163, Rate limits]). The update resources job also updated the resource download counts.

At the time of publishing the W. Ouyang*, F. Beuttenmueller*, E. Gómez-de-Mariscal*, C. Pape* et al. [107] preprint, the resource download counts were extracted through Zenodo's API. These download statistics do not only count downloads of the model weights files, but any file in the record. This inflated the model download count, as any expansion of a model card on the bioimage.io website caused a download of the model RDF, as well as associated cover images and the Markdown documentation file. This issue is addressed in section 3.3.4 *The Revised Collection Setup*.

In the automatically generated PRs a dedicated GH bot account *bioimageiobot*, that I setup, proposed to add/amend a resource.yaml file, which held basic information about the detected Zenodo record version(s) and any *rdf.yaml* files in them. All resource.yaml files were stored at github.com/bioimage-io/collection-bioimage-io/tree/main/collection in subfolders named by the corresponding resource `id`. A resource.yaml file could be edited by the BioImage.IO maintainers to block the resource as a whole or one of its versions.
Also by editing this file, fields of the associated RDF could be overridden. This allowed to improve the externally contributed RDF with minor edits, for example to correct typos, elaborate on the description, or mention an additional resource maintainer. Additionally

through the resource.yaml file the RDF was enriched with BioImage.IO specific `config` entries like the `nickname`. This limit of the BioImage.IO maintainers ability to edit a resource is lifted in the collection design, see section 3.3.4.

3.3.2 (Initial) Community Partner Contributions

The BioImage.IO Collection's initial update collection workflow processed all community partner resources hosted on GH. Community partners were defined and configured in github.com/bioimage-io/collection-bioimage-io/blob/main/collection_rdf_template.yaml. There, each community partner's collection RDF was referenced by the partner's `repository`, `branch`, and `collection_file_name`.

In addition to tests in the update collection workflow, community partners were required to test their collection RDF themselves to catch any potential errors early. I provided an example CI/CD setup for ilastik to be used as a template for other community partner software repositories. It is a simple GitHub Actions workflow calling `bioimageio.core`'s (deprecated) 'validate-partner-collection' command. Through such integrated testing in respective community partner GH repositories, errors could be caught before deployment. Processed community partner resources were cached in github.com/bioimage-io/collection-bioimage-io/tree/gh-pages/partner_collection to become part of the BMZ Collection. The separate processing of resources contributed by community partner and external contributors made the BioImage.IO Collection more complicated than it had to be. The revised, simplified collection design treats internal and external contributions equally.

3.3.3 (Initial) Test Summaries

As part of the CI/CD, each final, extended RDF's content was thoroughly tested — on the main branch when `bioimageio.spec`/`bioimageio.core` library updates occur and in the PRs for any new resource contributions. Static checks of the RDF are performed with `bioimageio.spec` (section 3.2.4). More extensive tests were conducted using `bioimageio.core` (section 3.4.1). For these tests, suitable (virtual) Python environments were needed. A Python environment is a Python interpreter (possibly of a specific Python version) with a specific set of additional Python libraries installed. Python environment

key	description
name	The name of the test
source_name	The Id, BioImage.IO nickname, URL or other identifier of the source that was tested
status	Weather the test ”passed” or ”failed”
error	‘null’ or an error message, if the test failed
nested_errors	For tested collection RDFs only, errors of collection items should be specified here keyed by their `id`
traceback	‘null’ or or a stack traceback as list of strings, if the test failed
warnings	A mapping of warning messages (for RDF fields), for example: ’documentation’: [’Missing validation section’], ’non-validation-warnings’: [’Computed dataset percentiles naively by averaging percentiles of samples.’, ’Device management is not implemented for keras yet, cannot unload model’]
bioimageio_spec_version	Version of `bioimageio.spec`, if used
bioimageio_core_version	Version of `bioimageio.core`, if used

Table 3.2: Fields that constitute a test summary.

management tools exist to create multiple, independent Python environments on the same machine. Beside a “base” environment such additional Python environments are called virtual environments. In the BioImage.IO Collection CI/CD, the required virtual Python environments to test model inference were created dynamically with Micromamba[164] — a Python environment management tool — based on the specified dependencies for each present model weights format.

The *update collection* GitHub Actions workflow deployed generated output documents with the GH Pages feature[163, GitHub Pages]. From there bioimage.io retrieved it on demand for display and download by users.

The BioImage.IO Collection CI/CD also generated test summaries based on `bioimageio.spec` and `bioimageio.core` for each accepted resource. A test summary is a mapping/dictionary with specific keys as described in table 3.2.

The Collection CI/CD, through the GH bot account “bioimageiobot”, triggered a GitHub

Actions workflow for every BioImage.IO community partner that registered their own CI/CD by configuring the collection template located at github.com/bioimage-io/collection-bioimage-io accordingly. To use available resource efficiently the partner's test workflow was dispatched with an input specifying only newly accepted or updated versions of RDFs with a `type` relevant to the community partner. This aided community partners to maintain up-to-date test summaries for all relevant BioImage.IO resources.

On a subsequent Collection CI/CD run, test summaries generated and hosted by the community partners were conflated into the Collection's test summaries, see figure 3.5. In addition, the Collection CI/CD continuously updated a *collection.json* file, listing all accepted resources with only a subset of available detail. The new Collection design produces an equivalent *collection.json* file. This *collection.json* is loaded by the bioimage.io website to display an overview of the BMZ and support searching it by tags or free text. Once a user clicks on a resource card, additional details are loaded from the BioImage.IO Collection — the full RDF and associated test summaries.

3.3.4 The Revised Collection Setup

The initial setup of the BioImage.IO Collection was centered around Zenodo as a storage backend. Reliance on this third party services comes with several drawbacks. Changes of the Zenodo API are out of the control of the BMZ and have interrupted the download functionality provided on bioimage.io in the past. A storage backend under control of the BMZ was needed. The revised Collection uploads resources to a dedicated (Amazon) Simple Storage Service (S3)-bucket in the EMBL-EBI Embassy Cloud (`https://uk1s3.embassy.ebi.ac.uk/public-datasets/bioimage.io`), where BMZ reviewers have full access. Using S3 — a common object storage API — makes transferring to alternative cloud storage providers simple if ever needed. An additional benefit is better scalability and control over rate limiting. With the data storage under control of the BMZ, the issue of download statistics can now be tackled independently of Zenodo. With full data access, maintaining the Collection is also easier because it does not rely on the responsiveness of individual resource contributors — BioImage.IO maintainers can independently create updated resource version when needed. Of course resource contributors are still able to contribute updated versions themselves by uploading a resource with the same identifier —

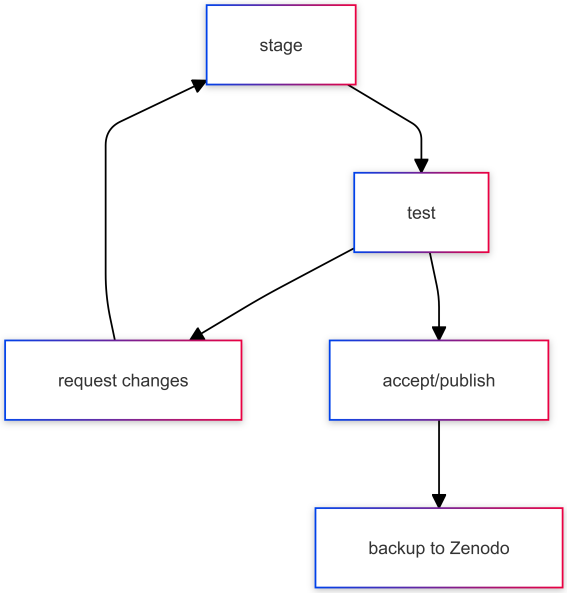


Figure 3.6: Control flow of the revised collection as documented in [117] New uploads through bioimage.io/upload or programmatic GitHub Actions workflow dispatch requests are processed by the 'stage' workflow to create a draft version. The draft is tested which generate a validation summary. This step can be triggered by reviewers directly to test any changes made directly on the S3 storage. Reviewers may also 'request changes', which sends an e-mail to the uploader with a message from the reviewer. Resource contributors are also informed via e-mail when their contributions are 'published' (which happens automatically after a reviewer 'accepted' a draft into the BMZ). Published resource descriptions are 'backed up to Zenodo'. The registered DOI may subsequently be used to reference the BMZ resource — for example on the bioimage.io website, or loading it with `bioimageio.spec` or `bioimageio.core`.

only the original uploaders and BMZ reviewers are able to do so.

While implementing these changes I simplified the involved workflow, see figure 3.6. I integrated all required workflow steps into a Python library — `bioimageio_collection_backoffice` — whose CLI is used by a reduced number of simplified GitHub Actions workflow to perform the same tasks. The `bioimageio_collection_backoffice` library can be used locally for easier development as well.

The revised design also enables the operation of independent Collection sandbox

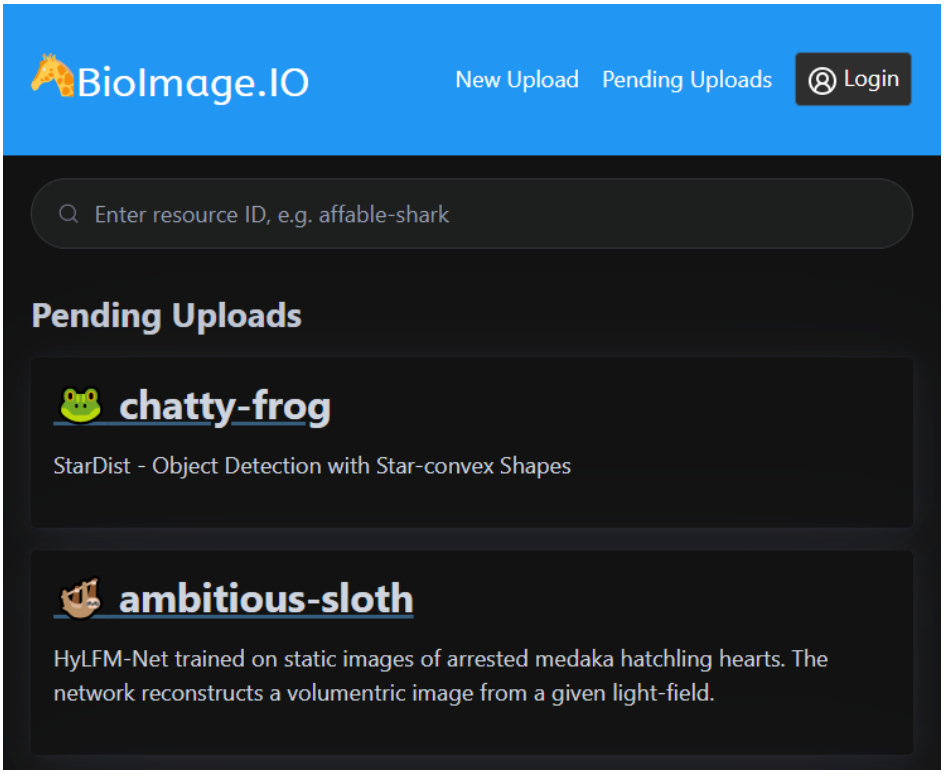


Figure 3.7: Overview of resource drafts in review [115, screenshot].

instances by using GH deployment environments. The Collection sandbox allows reviewers to quickly test new or modified resources in a fully functional BMZ without interfering with the production instance. The review process is no longer connected to PRs, which makes it more accessible to reviewers without extensive GH experience — the BMZ website provides an overview of resource drafts in review shown in figure 3.7. Each pending resource entry can extend to show a report with their validation summaries and other logged details, such as the GitHub Actions workflow run logs of the workflows that staged (uploaded) and tested the draft. Reviewers can use the BMZ upload interface to upload revised resource versions or edit drafts directly on S3 (for which many standard clients exist, even integration in a file explorer is easily setup).

adjectives	animals
acclaimed adaptable adventurous affable	angelfish 🐠 ant 🐜
affectionate agreeable ambitious amiable	badger 🦡 bat 🦇 bear 🐻
amusing angry artistic beautiful blithering	bird 🐦 blowfish 🐡
bold brave bright broad-minded calm careful	boar 🐗 bug 🐛 butterfly 🦋
charismatic charming chatty cheerful clean	camel 🐪 cat 🐈 chick 🐤
clever committed communicative compassionate	chicken 🐔 chipmunk 🐿️
confused conscientious considerate content	cow 🐄 crab 🦀 cricket 🦗
convivial courageous courteous creative	crocodile 🐊 deer 🦌
dangerous dazzling decisive demanding	dog 🐕 dolphin 🐬 dove 🕊️
dependable determined diligent diplomatic	dromedary 🐪 duck 🦆
discreet dynamic easy-going educated efficient	eagle 🦅 elephant 🐘
elegant emotional energetic enthusiastic	fish 🐟 flamingo 🦩 fox 🦊
enticing extroverted exuberant fair-minded	frog 🐸 goat 🐐 gorilla 🦍
faithful famous fast fearless forceful frank	hamster 🐹 hedgehog 🦔
friendly frisky funny generous gentle gleeful	hippo 🦛 honeybee 🐝
good gorgeous greedy gregarious handsome	horse 🐎 kangaroo 🦘
happy hardworking helpful heroic hiding	koala 🐨 ladybird 🐞
hilarious honest humorous idealistic	leopard 🐆 lion 🦁
imaginative impartial independent insecure	lizard 🦎 llama 🦙
intellectual intelligent intuitive inventive jolly	lobster 🦞 microbe 🦠
joyful kind knowledgeable kooky laid-back	monkey 🐒 mosquito 🦟
law-abiding likable lively loving loyal lucky	mouse 🐁 octopus 🐙
merry modest moody naked nervous nice	orangutan 🦧 otter 🦦
noisy non-judgemental observant optimistic	owl 🦉 ox 🐂 panda 🐼
organized passionate patient persistent	parrot 🦜 peacock 🦚
philosophical pioneering placid playful plucky	penguin 🐧 pig 🐖
poisonous polite popular powerful practical	poodle 🐩 rabbit 🐇
proactive quick-witted quiet rational rebellious	raccoon 🦝 rat 🐀 rhino 🦏
reliable reserved resourceful romantic	rooster 🐓 sauropod 🦕
self-confident self-disciplined sensible sensitive	scorpion 🦂 seashell 🐚
serious shivering shy sincere smart sneaky	shark 🦈 sheep 🐑
sociable solitary straightforward stupendous	shrimp 🦐 skunk 🦨
sympathetic talented talkative territorial	sloth 🦥 snail 🐌 snake 🐍
thoughtful tidy tough trustworthy unassuming	spider 🕷️ squid 🦑 swan 🦢
understanding upbeat upright vain vengeful	t-rex 🦖 tiger 🐅 turkey 🦃
venomous versatile warmhearted wild willing	turtle 🐢 water-buffalo 🐃
wise witty youthful zealous zestful	whale 🐋 wolf 🐺 zebra 🦓

Table 3.3: Possible BioImage.IO model Ids from *acclaimed-angelfish* to *zestful-zebra*. Each Id consists of one positive or neutral adjective paired with an animal, represented by a Unicode character. 174 adjectives and 87 animals yield 15 138 possible model Ids.

3.3.5 Memorable Resource Identifiers

Memorable resource identifiers are given to models, datasets and notebooks in the BMZ to identify them in a natural, human-friendly manner. Additionally a DOI, registered by Zenodo, is assigned to each Collection entry after the new Collection CI/CD has successfully uploaded the resource to Zenodo. The automatic upload to Zenodo also serves as a backup for the BMZ Collection and may be used as an archive as well at some point.

Initially the memorable resource identifiers were only an alternative to an identifier derived from the Zenodo DOIs, but the revised Collection uses the memorable identifiers first and foremost to allow for a draft stage without immediately registering a DOI.

For models, the version unspecific animal nickname, for example "ambitious-sloth", is a combination of one of 174 random, positive adjectives and one of 87 animals — allowing to uniquely identify a total of 15 138 models. Each animal has a Unicode character representation, for example 🦥, that is used for illustration. An overview of all possible model, dataset and notebook identifiers is given in table 3.3, table A.3 and table A.4, respectively. For new contributions an Id is randomly generated, but may be regenerated during the upload process. The adjectives, nouns and their associated Unicode characters are published in github.com/bioimage-io/collection/blob/main/bioimageio_collection_config.json.

3.4 Working with BioImage.IO Resources

To accelerate the dissemination of shared models it is of utmost importance that they are easy to use. One avenue for emerging use-cases is through established software tools. Many members of the bio-image community are familiar with at least one of the tools already provided by the current BioImage.IO community partners: ilastik[124], the DeepImageJ plugin for Fiji, ImageJ, and ImageJ2[108], ZeroCostDL4Mic[165], ImJoy, and Icy[166]. We, the BioImage.IO developers, aim to update these tools to proficiently access and use BioImage.IO resources. To support this integration with the BMZ, I implemented the `bioimageio.core` Python library, hosted at github.com/bioimage-io/core-bioimage-io-python with contributions from Constantin Pape and Maksim Novikov and Dominik Kutra. Additional contributions were made by William Patton, Carlos Garcia-López-de-Haro, Wei Ouyang, Joran Deschamps, Julian Hennies, Grzegorz Bokota, and Martin Hjelmare. The library, detailed in section 3.4.1,

lowers the development burden of integrating BMZ functionality into Python based software. Java analogs are also in development, using the `bioimageio.spec` and `bioimageio.core` Python package as references[167, 168, 123]. To provide a high-quality Python library that is easy be maintain, extend and work with, `bioimageio.core` is extensively type annotated and implements a large test suit to test its code base with pytest[169].

Besides channeling development efforts across the BioImage.IO community partners, the `bioimageio.core` library enables the wider bioimage community to integrate shared BioImage.IO resources in their work efficiently. Currently, examples of integrating `bioimageio.core` into existing projects include ilastik's *Neural Network Classification Workflow*[120], current versions of StarDist[9] and PlangSeg[170] 2.0[171].

3.4.1 The `bioimageio.core` Python library

The `bioimageio.spec` library serves not only to validate and document RDFs, but also to represent them in Python, see section 3.2.4. These representations are used in `bioimageio.core` to implement NN inference with support for automatic tiling of large images for out of Video Random Access Memory (VRAM) inference and other related tasks. Furthermore, reference implementations for pre- and postprocessing operations are implemented in this library, as well as agglomerating statistics informing for those processing steps. While RDF representation in `bioimageio.spec` are implemented for all `types` and `format_versions`, `bioimageio.core` may use a recent subset of possibly auto-converted metadata formats. This reduces code complexity by eliminating control flow statements separating distinct RDF format versions.

Model adapters

The most important objective of `bioimageio.core` is to provide model inference or "prediction" capabilities. Framework specific differences in NN weight formats and device handling — utilizing assigned GPUs — are addressed by *model adapter* classes. The shared *model adapter* interface provides a convenient abstraction layer for working with any NN instance, independent of its origin. The framework specific *model adapters* are available depending on the Python environment that `bioimageio.core` is used in. This keeps the essential

dependencies light-weight — allowing to add `bioimageio.core` into a Python environment with minimal overhead in terms of potential dependency conflicts and download size. It is possible, however, to enable all *model adapters*, for example with the current versions of PyTorch, TensorFlow, and the ONNX runtime.

To alleviate the issue of model deployment in any model consuming software, Constantin Pape contributed wrappers for weight format converters: Functions to convert weights saved as PyTorch 'state dict' to the TorchScript[128] or ONNX format using the PyTorch library. As explained in section 3.2 this conversion is not always possible. Model weight converters help to list more weight formats in a model RDF, increasing the chance a given software consumer can run model inference at all or with model weights in an optimized format like TorchScript.

Prediction pipeline

One abstraction layer above the *model adapters* is the *prediction pipeline*. It combines pre- and postprocessing routines with a *model adapter's* NN inference calls. For *Pythonic* use and safe device handling *prediction pipelines* can be used as context manager. The *model adapters* and *prediction pipeline* layout was initially designed and developed by me and Maksim Novikov as part of `tiktorch`[172] library. The current version of `tiktorch` — the library used in ilastik's *Neural Network Classification Workflow*[120] — uses `bioimageio.core`'s *prediction pipeline* and through it the *model adapters* as well[172].

In the bioimage community the most important data structure is the n-dimensional array or tensor. To benefit from tensor metadata described in section 3.2.2, `bioimageio.core` uses `xarray.DataArrays`[173, 174] throughout its code base to represent tensors with annotated dimensions. `bioimage.core` supports many common image formats via the `imageio`[175] library and .npy files via NumPy. The shape of loaded image tensors have to match the dimensionality of their corresponding axis annotations in the model RDF.

Predict with tiling

Large data samples often cannot be processed at once, but need to be split up into smaller sections to fit into VRAM. I implemented functionality to *predict with tiling* and *predict with*

padding. *Predict with tiling* allows to process samples larger than VRAM with control over tile overlap and padding. *Predict with padding* can be used when a sample can be processed at once, but should be virtually enlarged to account for edge artifacts. Suitable values for overlap/padding parameters are either specified or inferred from the `halo` field described in section 3.2.2.

Compute sample and dataset statistics

Computing data statistics is another task addressed by `bioimageio.core`. Implemented pre- and postprocessing operations (listed in section 3.2.2) specify which statistical measures need to be computed depending on their input arguments. Statistics computed for each sample independently are distinguished from dataset statistics. Furthermore, each statistical measure — arithmetic mean, standard deviation, variance, and percentiles — may be computed along an axis or a set of axes, for example to compute statistics per image channel. All required dataset statistics are computed jointly to minimize I/O overhead and deploy optimized algorithms to compute multiple statistical measures together. For datasets, the variance (and mean) are computed using the "General Updating Formula" by Chan et al. [176]. (If only the dataset mean is required and not the variance, only the mean is computed in an online manner.) To compute (approximate) percentiles for a dataset efficiently, using the t-digest[177] algorithm is supported through the experimental `crick`[178] library. To keep `bioimageio.core`'s dependencies light weight and not rely on such an experimental library, a naive fallback — averaging sample percentiles — is also implemented.

For a fast, interactive user-experience dataset statistics may also be updated online by a *prediction pipeline* while executing NN inference. With an optional set of samples to bootstrap the dataset statistics estimate this feature is intended for live, interactive workflows only and should not be used when computing final results, for example when exporting NN predictions for further analysis.

3.4.2 Dynamic RDF validation

I extended the validation capabilities of `bioimageio.spec` in `bioimageio.core`. For all present weight formats model inference is run with the specified `test_inputs` and com-

```
(sloth) C:\repos\bioimage-io\spec-bioimage-io>bioimageio --help
usage: bioimageio [-h] {validate_format,test,package,predict} ...

bioimageio - CLI for bioimage.io resources 🤟

library versions:
  bioimageio.core 0.6.9
  bioimageio.spec 0.6.9

spec format versions:
        model RDF 0.5.3
      dataset RDF 0.3.0
     notebook RDF 0.3.0

optional arguments:
  -h, --help            show this help message and exit

subcommands:
  {validate_format,test,package,predict}
    validate_format     bioimageio-validate-format - validate the meta data format of a bioimageio resource.
    test                bioimageio-test - Test a bioimageio resource (beyond meta data formatting)
    package             bioimageio-package - save a resource's metadata with its associated files.
    predict             bioimageio-predict - Run inference on your data with a bioimage.io model.
```

Figure 3.8: Running the `bioimageio-help` CLI command in the terminal gives an overview of its functionality.

pared to the `test_outputs`. The numerical precision of this comparison can be adjusted to account for small numerical differences in predicted outputs arising from small numerical differences in converted weights and exacerbated by small differences in hardware and low-level software. Also only applicable to model RDFs: a warning is generated if the linked `documentation` Markdown file does not contain a dedicated validation section. Such a validation section is highly encouraged to describe how a user can validate the models performance on their own data — validation beyond reproducing results on known test data.

3.4.3 The bioimageio CLI

The `bioimageio` CLI is implemented in `bioimageio.core`. It provides commands to test resources, run inference on one or more images and package resources to a self-contained ZIP archive. This CLI can also be used in CI/CD systems to aid model development. Its help-command output is given in figure 3.8; a validation summary output of the `bioimageio-test` command in figure 3.9.

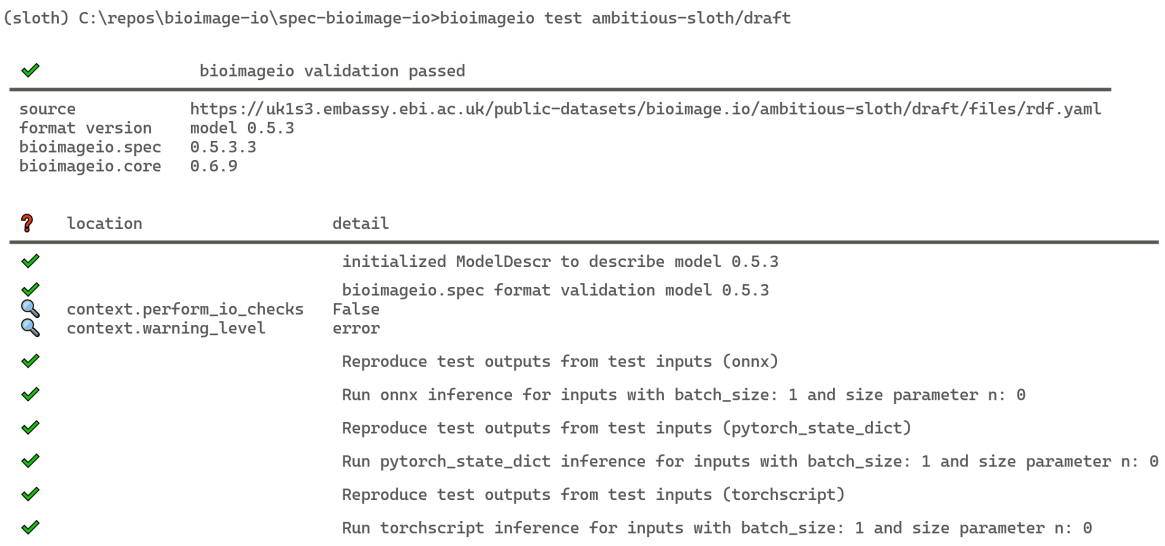


Figure 3.9: Running the `bioimageio-test` CLI command on 'ambitious-sloth' renders a validation summary in the terminal.

3.5 BioImage.IO Workflows

The BioImage.IO RDF Specification has limits: arbitrary pre- and postprocessing operations and more complex analysis workflows cannot be expressed. While the pre- and postprocessing operations listed in section 3.2.2 cover many common use-cases, it simply is not feasible to incorporate many more or arbitrary operations into the RDF specification, as their implementations need to be maintained and synchronized across compatible software tools. Making partner software BioImage.IO compatible should not entail high development costs — even if the shared Python or Java core libraries are not used. Similar limitation applies to extending the specification to more complex image analysis workflows, for example deploying model ensembles or combining object detection with image classification models. Even integrating de-facto standard methods like Cellpose[179] or StarDist[10] can be challenging due to special processing operations and additional dependencies unique to the method. Adding such additional dependencies to `bioimageio.core` would impose limitations on integrating into divers application environments.

3.5.1 The `bioimageio.workflows` Python package

I created the `bioimageio.workflows` Python package, hosted at github.com/bioimage-io/workflows-bioimage-io-python, to address the issue of integrating 'arbitrary' computational workflows into the BMZ. The package is designed to include Python based bioimage analysis workflows to be contributed by the BioImage.IO developers and the growing BioImage.IO community. To reconcile differences in Python environment requirements for these workflows, the package is designed to support distinct Python environments for groups of compatible workflows. Many workflows use a popular subset of the Scientific Python stack[180]: NumPy, SciPy[181], scikit-learn[182], scikit-image[183], Matplotlib[184], and Pandas[185]. Therefore I expect that maintaining a manageable number of Python environments can cover a large portion of Python based bio-image analysis workflows. I chose to facilitate communication between workflow processes with a Hypha[186][186] server using the ImJoy RPC[187][119, 187], which also allows to execute workflows remotely. As demonstrated in listing 3.8, a developer using `bioimageio.workflows` does not need to differentiate between execution in the main Python process, a subprocess with a different Python environment via a local, optionally auto-started Hypha server, or entirely remote execution through a remote Hypha server. Connecting to a specific server is configured by setting an environment variable. This setup has the additional benefit that regardless of local or remote code execution, any IDE finds the local workflow definitions — which allows for code completion, type checking, and integrated display of the workflow documentation. The use of a Hypha server communicating through the ImJoy RPC makes integration with the BioEngine trivial, as the BioEngine itself is managed by Hypha using the ImJoy RPC. Through the ImJoy RPC the `bioimageio.workflows` Python package can also be integrated in Java based analysis scripts and applications.

Selected implementation details

> "Your nascent developer community needs to have something runnable and testable to play with."
>
> (Raymond [188, Necessary Preconditions for the Bazaar Style])

Weather using `bioimageio.workflows` with a local or a remote Hypha server, integrating it into a Python script is straight-forward and keeps dependencies light-weight —

```
1  $ python -m asyncio
2  >>> import asyncio
3  >>> from bioimageio.workflows import hello
4  >>> await hello()
5  Hello!  # print within hello() function call
6  'Hello!' # hello() function return value
7  >>>
8  >>> from bioimageio.workflows.envs.default.local import hello
9  >>> await hello()
10 Hello!  # print within hello() function call
11 'Hello!' # hello() function return value
12 >>>
13 >>> from bioimageio.workflows.envs.default.remote import hello
14 >>> await hello()
15 'Hello!' # hello() function return value
```

Listing 3.8: The example workflow `hello` of the `bioimageio.workflows` Package is used to illustrate local and remote workflow execution. The first line starts the Python REPL with an `asyncio` concurrent context, which allows to `await` an asynchronous workflow function. Line 3 imports the `hello` demonstration workflow. Executing it in line 4 prints "Hello!" (line 5) and returns the string "'Hello!'" (line 6). The print function's output is only visible, because `hello()` was executed in the main Python process running in the minimal Python environment compatible with `bioimageio.workflows`. Line 8 imports `hello` explicitly in the main process from the `default.local` submodule: The behavior (line 9-11) is the same. Importing `hello` from the `default.remote` submodule in line 13 causes `hello()` to be executed in a subprocess — `hello`'s print call is not visible. The workflow return value `'Hello!'` is still returned through the ImJoy RPC in the last line. This demonstrates that a program using `bioimageio.workflows` can run the same code regardless of local or remote workflow execution.

its base functionality only relies on bioimageio.core, Dask[189] and the ImJoy RPC. Dask implements a chunked array that allows lazy evaluation of an n-dimensional array represented as a grid of NumPy arrays[189, Dask arrays]. The `xarray.DataArray`[174] used in bioimageio.core to work with annotated NumPy arrays, can also be created from a Dask array[189, Dask arrays]. bioimageio.core operates on `xarray.DataArrays` backed by NumPy. Its tiling functionality to process images still requires the image to fit into Random Access Memory (RAM) (just not into VRAM). To overcome this limitation `bioimageio.workflows` uses Dask. Taking full advantage of lazy evaluation is in principle supported by the ImJoy RPC. Implementations for lazy encoders of n-dimensional arrays already exist[187], but further development of a lazy array decoder is required for the `bioimageio.workflows` use-case. Development efforts in the wider open science community will likely aid this effort, for example further development of async-zarr[190] — Zarr[191][191] arrays with asynchronous data access. The current, initial implementation of `bioimageio.workflows` therefore still requires tensors send through the ImJoy RPC to fit in memory. For local processing in a suitable Python environment, lazy evaluation with Dask can already be used, though. I illustrate the potential of `bioimageio.workflows` with three examples:

1. A minimal example workflow to serve as a template and demo (definition in listing A.3, use demonstrated in listing 3.8).
2. The *Inference with Dask* workflow (see listing 3.10) computes NN predictions on a chunked `xarray.DataArray` (backed by a Dask array). This showcases the benefits of using Dask: The lazy output is returned almost instantly and computation only occurs for selected regions upon access.
3. *2D StarDist prediction* demonstrates a workflow with a dedicated Python dependency: the stardist[9] Python module. It combines inference using a specified 2D StarDist[9] model with the StarDist[9] specific postprocessing. The use of this `stardist_prediction_2d` workflow is demonstrated in listing 3.9.

3.5.2 The workflow RDF

To make workflows discoverable on the bioimage.io website I added a workflow RDF specification to bioimageio.spec. This specification of `type` "workflow" adds `inputs`, `options`, and `outputs` to the general RDF specification. These additional fields list parameters described

```
1  $ python -m asyncio
2  >>> import asyncio
3  >>> from bioimageio.core import load_resource_description
4  >>> model = load_resource_description("chatty-frog")
5  >>> from bioimageio.core.image_helper import load_image
6  >>> raw = load_image(model.test_inputs[0], model.inputs[0].axes)
7  >>>
8  >>> from bioimageio.workflows import stardist_prediction_2d
9  >>> labels, polys = await stardist_prediction_2d("chatty-frog", raw)
10 >>> labels.shape
11 (1, 304, 512)
12 >>> from bioimageio.workflows.envs.default.remote import
   ↪  stardist_prediction_2d
13 >>> from bioimageio.workflows.envs.default.local import
   ↪  stardist_prediction_2d
14 ImportError: cannot import name 'stardist_prediction_2d' from
   ↪  'bioimageio.workflows.envs.default.local'
```

Listing 3.9: To setup the example use of the `stardist_prediction_2d` workflow `bioimageio.core` is used in line 3-6 to load a suitable `raw` input image. The `stardist_prediction_2d` is imported (line 8) and used (line 9) to compute a label image. While the stardist[9] model "chatty-frog", used in this example returns 33 stardist specific output channels, the workflow returns the postprocessed `label` image with a single channel as shown in lines 10 and 11. For this special postprocessing the workflow implementation uses the stardist Python module[9]. With this special dependency the workflow is only compatible with the `bioimageio.workflows.envs.stardist` environment and is not available in the default environment — importing the workflow explicitly from the `remote` submodule (line 12) succeeds, while importing it explicitly from the `local` submodule fails (line 13,14).

```python
async def inference_with_dask(
    model_rdf: Union[str, PathLike, dict, IO, bytes, URI, RawResourceDescription],
    tensors: Sequence[xr.DataArray],
    boundary_mode: Union[BoundaryMode, Sequence[BoundaryMode]] = "reflect",
    enable_preprocessing: bool = True,
    enable_postprocessing: bool = True,
    devices: Sequence[str] = ("cpu",),
    tiles: Optional[Sequence[Dict[str, int]]] = None,
) -> OrderedDict[str, xr.DataArray]:
    """Model inference with chunked dask arrays for tiling

    To run inference on arbitrary input tensors they are chunked such that
    ↪  considering
    halo and offset all inputs to the model have `tiles` shape.

    .. code-block:: yaml
    authors:
    - {name: Fynn Beuttenmüller, github_user: fynnbe, affiliation: EMBL Heidelberg}
    cite:
    - {text: BioImage.IO, url: "https://doi.org/10.1101/2022.06.07.495102"}
    - text: "Dask Development Team (2016). Dask: Library for dynamic task
    ↪  scheduling"
      url: "https://dask.org"
    tags: [workflow, inference, dask, tiling]
    covers: ["https://zenodo.org/record/7609747/files/dask_inference_cover.png"]

    Args:
        model_rdf: model RDF that describes the model to be used for inference
        tensors: model input tensors
        boundary_mode: How to pad missing values (for all or each model input)
        enable_preprocessing: If true, apply the preprocessing specified by...
        enable_postprocessing: If true, apply the postprocessing specified...
        devices: devices to use by the created model adapter
        tiles: Tile shapes for model inputs. Defaults to estimates based on...

    Returns:
        outputs. named model outputs
    """
```

Listing 3.10: Definition of the *Inference with Dask* workflow adapted from [192]. The function name `inference_with_dask` is used as Id with the "bioimageio/" prefix. The function arguments (lines 2-11) are type annotationstype annotated. Together with their Google-style docstring[193, 383-functions-and-methods] descriptions (lines 29-35) these arguments are processed by a script to populate the `inputs` and `outputs` fields of the generated workflow RDF. Similarly the return value is annotated in line 12 and described in line 38. The *Inference with Dask* workflow returns an ordered dictionary containing the named output tensors of the model specified by the `model_rdf` input. Multiple workflow return values can be annotated and returned as a tuple.

by a `name`, a `description`, a `type`, and an `axes` field. For `type` "tensor" the `axes` field holds an updated version of the model RDF's `axes` fields[5].

The workflow definition is sufficient to generate workflow RDFs programmatically. Relevant fields and values are derived from type annotations and the function's docstring. This eases contributing a workflow, as only a well documented and tested Python function needs to be added that follows the pattern of the three initial workflows. CI/CD (a GitHub Actions workflow) is setup to check that for all implemented workflows at least one dedicated test exists. I also added tests to verify that each workflow environment submodule is setup such that workflow functions imported locally or executed remotely are identical.

3.5.3 Contributing workflows

Designing new bio-image analysis workflows requires more programming expertise than contributing datasets or models to the BMZ — workflows are described in Python code. Proposed workflow functions need to be reviewed by BioImage.IO developers to ensure quality and protect the community from malicious intent. Luckily no new process needs to be created to meet these demands; there are well established platforms for collaborative software development. As bioimageio.workflows is hosted by GH like all BioImage.IO related repositories, managing contributions to it through GH PRs is the natural choice.

3.5.4 Distinction from notebooks

The BioImage.IO community partner ZeroCostDL4Mic[165] uses Jupyter notebooks with great success to provide guided workflows for training NN models on user data[165]. The ZeroCostDL4Mic bioimage analysis pipelines are meant for interactive use and designed to accommodate users who are not experts in DL[165]. Their users do not even need to know how to code[165]. Through the provided explanations and comments in-between the executable code their Jupyter notebooks educate the user in a hands-on manner and aid the user in correctly applying their pipelines.
Arbitrary, Python based analysis workflows could also be expressed in Jupyter notebooks. However, integrating Jupyter notebooks into another Jupyter notebook or (Python) script is

[5]The `axes` fields of the model RDF are planned to be updated to match the workflow RDF's

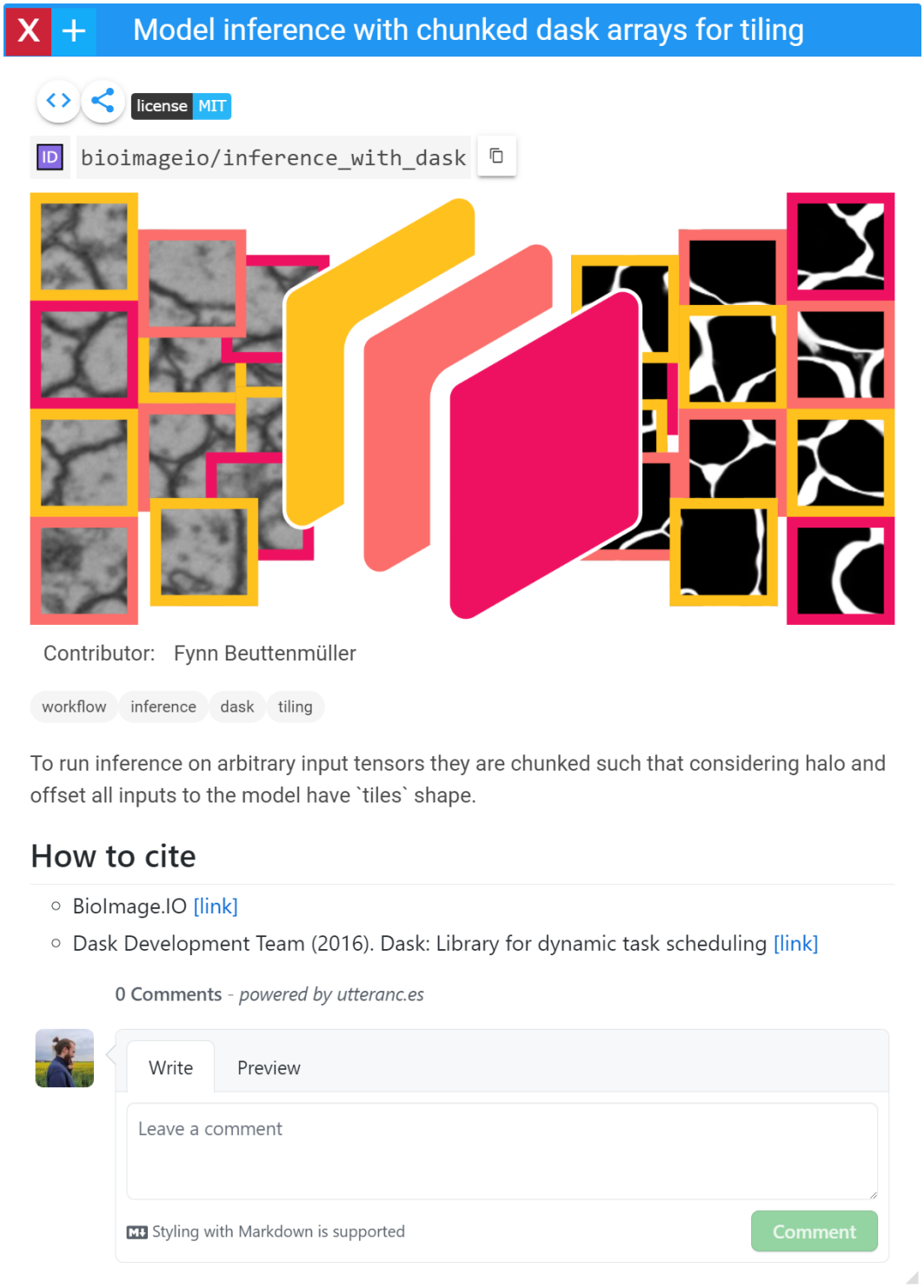


Figure 3.10: A resource card preview[a] on bioimage.io generated from the workflow RDF, which is generated from the workflow definition shown in listing 3.10.

[a]BioImage.IO workflows are not yet listed on the bioimage.io website at the time of writing.

infeasible.

With their documented in- and outputs BioImage.IO workflows are suitable for programmatic use. They are not an alternative to Jupyter notebooks, but rather an addition — a reusable step in a more complex pipeline. Or, on their own, a workflow ready for deployment.

3.5.5 Distinction from BioEngine applications

BioEngine applications are programs that communicate with a BioEngine, which is managed by a Hypha server[107]. The `bioimageio.workflows` Python package can be seen as a BioEngine application, because it allows executing workflows through the BioEngine.

BioEngine *web* applications are intended to be used in the browser and usually include visualization of the results[107]. Developing workflows as such *web* applications limits their applicability. Creating any custom BioEngine application for an individual workflow is possible and may provide workflow users with additional guidance through a custom GUI. However, contributing a workflow to `bioimageio.workflows` is much simpler than writing a standalone application. Workflows can be used in Python code by calling an imported Python function — a function documented and structured in a unified manner, which makes it easier to work with compared to individual applications or Python packages.

With further development of the BMZ, BioEngine (web) applications may be developed to execute workflows using the `bioimageio.workflows` package. One generic BioEngine application could offer a GUI suitable for all BioImage.IO workflows, analogous to the existing generic BioEngine web application that can run inference on example data for most of the BioImage.IO models, using the `bioimageio.core` package (see section 3.4.1).

Designed for community contributions, `bioimageio.workflows` allows to develop workflows as if they would exist in their own simple Python package. At the same time these BioImage.IO workflows are ready for cloud computing and more accessible through a unified interface.

3.6 BioImage.IO Conclusion

I have substantially contributed to the BioImage Model Zoo (BMZ) community effort to make deep learning models, datasets, notebooks and specialized workflows sharable in a FAIR way through the BioImage.IO Collection. This makes DL based workflows for bioimage analysis more accessible and reproducible. I lead the development of the `bioimageio.spec` library, which serves to define, document, and statically validate RDFs. Furthermore, I implemented `bioimageio.core` to make working with RDFs in Python easier by providing convenient abstraction layers for model inference including pre- and postprocessing operations and aggregation of image statistics. To populate the BMZ with content, I setup the `bioimageio_collection_backoffice` library in the BioImage.IO Collection repository along with extensive CI/CD pipelines to process and ease the review of contributions by the BioImage.IO community. Finally, to expand the `bioimageio` Python libraries' capabilities to more advanced bioimage analysis workflows, I added BioImage.IO workflow RDFs to the BMZ ecosystem, supported by the `bioimageio.workflows` Python library and described by workflow RDFs compatible with the BMZ RDF specification.

The `bioimageio` Python libraries and the BioImage.IO Collection not only provide the backbone functionality of the BMZ, but also serve as reference implementation and as computational backend for additional software in the BMZ ecosystem. All libraries are developed according to open source principles. Packages for integration are available via Python Package Index (PyPI)[194] and conda-forge[195]. In addition to providing entry-points to consolidate further development efforts within the community, they also address the challenge of creating a smooth user experience across multiple open source tools.

The BMZ ecosystem is developing fast. Multiple community partners have joined the BMZ over the recent years, all of them relying on the tools that I have developed. In the future I envision further collaborative development of the specification covering new DL frameworks and especially new methodologies that require approaches like multiple inference runs for diffusion models or test time adaptation for transformers. I also foresee novel biological use-cases through growing the contributing developer community of BMZ. I have already contributed to the improvement of model developer services such as CI/CD to encourage more external submissions to the BMZ.

Future growth of the BioImage.IO community will depend on securing sufficient fund-

ing, but I remain optimistic that the BMZ concept addresses a real need in the bioimage community and will therefore continue to thrive.

3.7 BioImage.IO Outlook

Embedded in the BioImage.IO community, the `bioimageio` libraries will continue to be developed as open source, community driven software projects. The RDF specification will continue to evolve to describe shared resources in growing detail and accuracy, moving towards new use cases. To ease further development of software that interacts with BioImage.IO resources, the RDF specification can be streamlined, taking into account feedback from developers that already integrate their tools with the help of the `bioimageio` libraries.

Resource contributors are guided with intuitive validation summaries that report concisely on incompatible RDF values. This allows to formulate stricter RDF specifications without hampering resource development. These lead to RDFs with little ambiguity, increasing the overall value of the BioImage.IO Collection and support transparent and reproducible experiments.

Implementing changes of the RDF specification to this end will be aided by `bioimageio.spec`'s auto-conversion mechanism to incorporate minor changes into `bioimageio.spec` dependent libraries with ease.

To increase the scale for bioimage analysis reproducibility and fast prototyping, the dataset RDF specification should be further refined and coupled to the emerging Open Microscopy Environment (OME)-Zarr[196] community standard for large microscopy images. This could lead to routinely processing large datasets with `bioimageio.core`, which is already well equipped for this task with its use of optionally Dask-backed `xarray.DataArrays` as showcased in the *Inference with Dask* workflow.

Fine-tuning and training utilities could be a valuable addition to `biomiageio.core`, which could draw from an extended model RDF specification with training in mind. When combining models and datasets for training, semantic channel descriptions could help to establish meaningful connections. Expanding semantic annotations could also strengthen trustworthy interoperability across community partner software, improving user experience. Development of a suitable ontology should follow community efforts such as Next-generation

file formats (NGFF).

Otherwise no additional functionality for `bioimageio.spec` or `bioimageio.core` has to be added to generate a meaningful impact in the bioimage community as `bioimageio.spec` and `bioimageio.core` have already been incorporated successfully in community partner software. Short term development should focus on further improving the user experience and extent of the documentation to further ease the adoption into other software projects.

Additional work on `bioimageio.workflows` is required in collaboration with community partner software developers ready to incorporate it in their software, to better integrate the library in the BMZ ecosystem. With wider adoption of `bioimageio.workflows`, the BMZ has the opportunity to cover vastly more use-cases, benefiting from its existing infrastructure around models, datasets and notebooks.

4 Conclusion

In this thesis I have substantially contributed to two high-impact, interdisciplinary research projects. My contributions in the context of HyLFM to high-speed, high-fidelity DL powered imaging have shown a path to increase trustability in quantitative DL approaches for image reconstruction. Integrating different microscopy principles might open more experimental possibilities. With computational methods we can exploit complementary microscopy approaches enabling new streams of answers for today's and tomorrows biological questions. Further aiding biologists with their scientific endeavors does not only depend on the continued development of ever more capable microscopes intertwined with AI systems, but also on ever growing bioimage analysis community, brought together by shared resources, shared methods and a shared visions of interoperability will stimulate biological research.

In my contributions to the BMZ I have paved the way to standardized, FAIR pretrained DL model exchange. Supported by `bioimageio.core` and the infrastructure I have developed the BMZ has emerged as a central hub for users and developers of DL methods in microscopy. The field of AI is developing rapidly and I hope my contributions to DL powered bioimage analysis will help the community to keep the open source and open science spirit.

Appendix

dataset	URL
fluorescent beads	`https://www.ebi.ac.uk/biostudies/studies/S-BSST622`
medaka heart	`https://www.ebi.ac.uk/biostudies/studies/S-BSST604`
zebrafish neural activity	`https://www.ebi.ac.uk/biostudies/studies/S-BSST633`

Table A.1: Datasets used to train and evaluate HyLFM-Nets. They contain light-sheet planes/volumes and corresponding light fields recorded with the HyLFM setup described in section 2.4.1.

NA	$M_{180\,\mathrm{mm}}$	$M_{200\,\mathrm{mm}}$	working distance	model
0.5	20	22.22	3.50 mm	Olympus UMPLFLN20XW
0.8	20	22.22	0.61 mm	Olympus UPLXAPO20X[a]
1.0	20	22.22	2.00 mm	Olympus XLUMPLFLN20XW
1.05	25	27.78	2.00 mm	Olympus XLPLN25XWMP2

Table A.2: Selected examples of objectives that may be suitable for LFM applications with optical setups similar to the one described in section 2.3.1. Objective specifications from [197]. $M_{180\,\mathrm{mm}}$ is the magnification when using a tube lens with a focal length of 180 mm. $M_{200\,\mathrm{mm}}$ is the effective magnification when using a tube lens with a focal length of 200 mm.

[a] unlike the other objectives listed this is not a water-dipping objective, but a dry objective.

adjectives	food items
ambrosial appealing appetizing aromatic beautiful bitter blissful bountiful buttery chewy chilled creamy crispy crumbly crunchy decadent delectable delicious delightful divine enticing exotic expensive exquisite extraordinary flavorful flavorsome fluffy fragrant fresh frosty fruity fulfilling gooey gourmet gratifying heavenly homely impressive indulgent irresistible juicy luscious mellow moist mouthwatering nourishing nutty palatable piquant pleasing pungent resourceful rich robust sapid satisfactory savory scintillating scrumptious smooth spicy splendid sticky sublime succulent sumptuous superb sweet tangible tangy tantalizing tasteful tasty tempting tender toothsome uplifting velvety vibrant yummy zesty zingy	apple 🍎 avocado 🥑 bacon 🥓 bagel 🥯 baguette 🥖 banana 🍌 bento-box 🍱 bread 🍞 broccoli 🥦 burger 🍔 burrito 🌯 butter 🧈 cake 🍰 candy 🍬 carrot 🥕 cheese 🧀 cherry 🍒 chocolate 🍫 coconut 🥥 cookie 🍪 croissant 🥐 cucumber 🥒 cupcake 🧁 curry 🍛 custard 🍮 doughnut 🍩 drumstick 🍗 dumpling 🥟 egg 🥚 eggplant 🍆 falafel 🧆 fortune-cookie 🥠 french-fries 🍟 fried-shrimp 🍤 garlic 🧄 grapes 🍇 honey 🍯 hot-dog 🌭 ice-cream 🍨 kebab 🥙 kiwi 🥝 knuckle 🍖 leafy-green 🥬 lemon 🍋 lollipop 🍭 mango 🥭 melon 🍈 moon-cake 🥮 mushroom 🍄 onion 🧅 orange 🍊 paella 🥘 pancakes 🥞 pasta 🍝 peach 🍑 peanut 🥜 pear 🍐 pepper 🌶 pie 🥧 pineapple 🍍 pizza 🍕 popcorn 🍿 potato 🥔 pretzel 🥨 rice 🍚 salad 🥗 sandwich 🥪 steak 🥩 strawberry 🍓 sushi 🍣 sweet-potato 🍠 taco 🌮 takeout 🥡 tomato 🍅 waffle 🧇 watermelon 🍉

Table A.3: Overview of possible bioimage.io dataset Ids from *ambrosial-apple* to *zingy-watermelon*. Each valid dataset Id consists of one positive or neutral adjective paired with a food item. Each food item is represented by a Unicode character to make it easier for developers to implement friendly UIs without having to include icons as images. The adjectives, food names and their associated Unicode characters are published in github.com/bioimage-io/collection/blob/main/bioimageio_collection_config.json. 83 adjectives and 76 food items yield 6308 possible dataset Ids.

adjectives	clothing items
adaptable adorable aerodynamic airy all-purpose alluring appealing assertive attractive authoritative bespoke bold breezy bright brilliant brisk captivating charming cheerful chic classic classy clean comfortable compelling convenient cozy creative crisp cuddly cushioned customized cute dapper delightful distinctive dramatic draping dynamic easy effervescent efficient effortless electrifying elegant enchanting encouraging endearing energetic energizing enlivening enthralling euphoric exceptional exciting exclusive exhilarating exquisite extraordinary fashionable flattering flexible flowing fluffy fluid fresh fun functional fuzzy glamorous gleaming glistening glittering glorious glossy graceful handy happy incredible individual innovative inspiring inventive invigorating inviting irresistible joyful lavish light limited-edition lively lovely luminous lush luxurious magnificent majestic mesmerizing modish motivating multifunctional opalescent opulent original outstanding padded personalized phenomenal playful pleasant plush polished potent powerful practical radiant rare refined refreshing regal relaxing remarkable resourceful resplendent revitalizing satiny seamless sensuous sharp shimmering shining silken silky simple sleek slick slinky smooth snug soft soothing sophisticated sparkling special spirited sprightly stimulating streamlined striking strong stunning stylish sumptuous sunny supple sweet tailored thrilling timeless trendy uncommon uncomplicated unique uplifting useful velvety versatile vibrant vigorous warm well-fitted whimsical winsome	backpack 🎒 ballet-shoes 🩰 bikini 👙 blouse 👚 boot 👢 briefs 🩲 cap 🧢 coat 🧥 crown 👑 diving-mask 🤿 dress 👗 flat-shoe 🥿 glasses 👓 gloves 🧤 googles 🥽 graduation-hat 🎓 handbag 👜 hat 👒 helmet ⛑️ high-heels 👠 hiking-boot 🥾 jeans 👖 kimono 👘 lab-coat 🥼 necktie 👔 pouch 👝 purse 👛 ribbon 🎀 running-shirt 🎽 safety-vest 🦺 sandal 👡 sari 🥻 scarf 🧣 shoe 👞 shopping-bag 🛍️ shorts 🩳 sneaker 👟 socks 🧦 sunglasses 🕶️ swimsuit 🩱 t-shirt 👕 top-hat 🎩

Table A.4: Overview of possible bioimage.io notebook Ids from *adabtable-backback* to *winsome-top-hat*. Each valid notebook Id consists of one positive or neutral adjective paired with a clothing item. Each clothing item is represented by a Unicode character to make it easier for developers to implement friendly UIs without having to include icons as images. The adjectives, clothes names and their associated Unicode characters are published in github.com/bioimage-io/collection/blob/main/bioimageio_collection_config.json. 173 adjectives and 42 clothing items yield 7266 possible notebook Ids.

```
# statically trained HyLFM-Net

This network was trained on light-field and corresponding light-sheet data acquired with a HyLFM microscope-a hybrid light-field light-sheet microscope.
For a description of the implementation and information about the microscope please refer to "Deep learning-enhanced light-field imaging with continuous validation." at https://doi.org/10.1038/s41592-021-01136-0.
For implementation details and how to train such networks please additionally refer to https://github.com/kreshuklab/hylfm-net.

This network is not expected to transfer to data acquired with a different microscope.
Even if your light-field data recorded medaka heart nuclei with a 20x objective represented with $n_\text{num}=19$, this model might not yield faithful reconstructions.

## Validation
With a HyLFM microscope continuous validation is enabled by the acquisition of groundtruth-like light-sheet planes.
This network should only be used in such a hybrid optical setup to ensure its reconstruction bias toward Medaka heart nuclei is justified and the implicitly learned noise and scattering distributions are applicable.

## Training details
|  |  |
| --- | --- |
| training id | celestial-sweep-13 |
| checkpoint | val00099_ep196_it173.pth |
| OS | Linux-3.10.0-1160.6.1.el7.x86_64-x86_64-with-centos-7.9.2009-Core |
| Python verison | 3.7.9  |
| commit | a402ea6b92fc4abc994309887729493d97a9990a |
| duration | 2d 6h 27m 5s |
| GPU type | GeForce RTX 2080 Ti |

training command:
```
```
```

````
hylfm/train.py heart_static_fish2_f4 --batch_multiplier=2
 ↪ --batch_size=1 --c00_2d=768 --c00_3d=32 --c01_2d=768 --c02_2d=512
 ↪ --c03_2d=256 --c10_2d=512 --c10_3d=8 --c11_2d=256 --c11_3d=8
 ↪ --c12_2d=0 --cin_3d=64 --crit_beta=1
 ↪ --crit_decay_weight_by=0.8949745643842519
 ↪ --crit_decay_weight_every_unit=epoch
 ↪ --crit_decay_weight_every_value=1 --crit_ms_ssim_weight=0.01
 ↪ --crit_threshold=2.394979173266996
 ↪ --crit_weight=0.007110228990791768
 ↪ --criterion=WeightedSmoothL1_MS_SSIM --init_fn=xavier_uniform
 ↪ --kernel2d=3 --kernel3d=3 --last_kernel2d=5 --lr_sched_factor=0.5
 ↪ --lr_sched_patience=8 --lr_sched_thres=0.0001
 ↪ --lr_sched_thres_mode=abs --lr_scheduler=ReduceLROnPlateau
 ↪ --max_epochs=200 --nnum=19 --opt_lr=0.0002663980703785583
 ↪ --optimizer=Adam --patience=15 --up0_2d=384 --up0_3d=16
 ↪ --validate_every_unit=epoch --validate_every_value=2 --z_out=49
```
````

Listing A.1: Content example for a GFM file linked under `documentation`.
````

```yaml
authors:
  - affiliation: EMBL Heidelberg
    github_user: fynnbe
    name: Fynn Beuttenmueller
    orcid: 0000-0002-8567-6389
cite:
  - doi: 10.1038/s41592-021-01136-0
    text: "Beuttenmueller, Wagner, N., F., Norlin,
      N. et al. Deep learning-enhanced light-field imaging with continuous
      ↪ validation.
      Nat Methods 18, 557-563 (2021)."
config:
  bioimageio:
    thumbnails: { cover.png: cover.thumbnail.png }
covers: [cover.png]
description:
  HyLFM-Net trained on static images of arrested medaka hatchling hearts.
  The network reconstructs a volumentric image from a given light-field.
documentation: README.md
format_version: 0.5.3
git_repo: https://github.com/kreshuklab/hylfm-net
id: ambitious-sloth
id_emoji:
inputs:
  - axes:
      - type: batch
        size: 1  # only due to onnx weights
      - channel_names: [channel0]
        id: channel
        type: channel
      - concatenable: false
        id: y
        scale: 1.0
        size: 1235
        type: space
      - concatenable: false
        id: x
        scale: 1.0
        size: 1425
        type: space
    data:
      scale: 1.0
      type: float32
      unit: arbitrary unit
    description: ""
    id: lf
    optional: false
    preprocessing:
      - id: ensure_dtype
        kwargs: { dtype: float32 }
      - id: scale_range
        kwargs:
```

```
            axes: [y, x]
            eps: 1e-06
            max_percentile: 99.8
            min_percentile: 5.0
            reference_tensor: null
        - id: ensure_dtype
          kwargs: { dtype: float32 }
      sample_tensor:
        sha256: dc47897c483e5ba42965b00171cbf5b96de33c4bf46b0521cb888358657c8f37
        source: sample_input.tif
      test_tensor:
        sha256: 39d9cb7a8aeca76e4c915ab21b3ef317429dba81b8fef601fe7a0b18f1338538
        source: test_input.npy
  license: MIT
  links: [imjoy/BioImageIO-Packager]
  maintainers:
    - affiliation: null,
      email: null,
      github_user: fynnbe,
      name: Fynn Beuttenmueller,
      orcid: null,
  name: HyLFM-Net-stat
  outputs:
    - axes:
        - type: batch
          size: 1
        - channel_names: [channel0]
          description: ""
          id: channel
          type: channel
        - id: z
          scale: 1.0
          size: 49
          type: space
        - id: y
          scale: 1.0
          size: 244
          type: space
        - id: x
          scale: 1.0
          size: 284
          type: space
      data:
        scale: 1.0
        type: float32
        unit: arbitrary unit
      description: "predicted volume of fluorescence signal"
      id: prediction
      postprocessing:
        - id: ensure_dtype
          kwargs: { dtype: float32 }
      sample_tensor:
        sha256: 15eb3df516c6c7dd06b23f11f4b0ea5479c342d9fb9a88c870e0e75f48103733
        source: sample_output.tif
```

```yaml
    test_tensor:
      sha256: 5c31cdd9d99e67fd10f9d387b4315dc978257c8805e2c8300621212b4a122ef7
      source: test_output.npy
tags:
  - light-field-microscopy
  - pytorch
  - fluorescence-light-microscopy
  - image-reconstruction
  - nuclei
  - hylfm
timestamp: "2024-06-05T15:34:43.433531Z"
training_data: { id: 10.5281/zenodo.7612115 }
type: model
uploader: { email: thefynnbe@gmail.com, name: "Fynn Beuttenmueller" }
version: 1.2
weights:
  onnx:
    opset_version: 15
    parent: pytorch_state_dict
    sha256: 55ad061651840fe4b6da6733c52402ae3296392f74fed8a9482e4bc62040938f
    source: weights.onnx
  pytorch_state_dict:
    architecture:
      callable: HyLFM_Net
      kwargs:
        c_in_3d: 64
        c_res2d: [768, 768, 512, 256, u384, 512, 256]
        c_res3d: [32, u16, 8, 8]
        init_fn: xavier_uniform
        kernel2d: 3
        kernel3d: 3
        last_kernel2d: 5
        nnum: 19
        z_out: 49
      sha256: 14e7777576ed54dbf8614a461465f02a9dca99d882878ac35e9b0d20ca6556b1
      source: model.py
    dependencies:
      sha256: e0c059d829fa03193eede76961746f464ac9b07d072b1e6ee62395d5c03c8606
      source: environment.yaml
    pytorch_version: 1.13
    sha256: 461f1151d7fea5857ce8f9ceaf9cdf08b5f78ce41785725e39a77d154ccea90a
    source: weights.pt
  torchscript:
    parent: pytorch_state_dict
    pytorch_version: 1.13
    sha256: ec01e0c212b5eb422dda208af004665799637a2f2729d0ebf2e884e5d9966fc2
    source: weights_torchscript.pt
```

Listing A.2: Full content example of a model RDF in format version 0.5.3. This RDF describes the HyLFM-Net-stat from chapter 2. The model is named ‘HyLFM-Net-stat’, but its BMZ identifier is ‘ambitious-sloth’.

```python
from typing import Optional

import xarray as xr


async def hello(msg: str = "Hello!", tensor: Optional[xr.DataArray] = None) -> str:
    """dummy workflow printing msg

    This dummy workflow is intended as a demonstration and for testing.

    .. code-block:: yaml
    authors: [{name: Fynn Beuttenmüller, github_user: fynnbe}]
    cite: [{text: BioImage.IO, url: "https://doi.org/10.1101/2022.06.07.495102"}]
    tags: [demo]
    covers: ["https://github.com/.../bioimage-io-icon.png"]

    Args:
        msg: Message
        tensor: tensor whose shape is added to message
            axes:
            - type: batch
            - type: space
              name: x
              description: demo space x
              unit: millimeter
              step: 1.5
            - type: index
              name: demo index
              description: a special index axis

    Returns:
        msg. A message, optionally enriched with the tensor shape.
    """
    if tensor is not None:
        msg += f" The tensor shape is {tensor.shape}"

    print(msg)
    return msg
```

Listing A.3: Definition of the *hello* minimal example workflow defined in [192]. The function name `hello` is used as Id with the "bioimageio/" prefix. The function arguments (line 6) are type annotationstype annotated. Together with their Google-style docstring[193, 383-functions-and-methods] descriptions (lines 7-29) these arguments are processed by a script to populate the `inputs` and `outputs` fields of the generated workflow RDF. Similarly the return value — a string — is annotated in line 6 and described in line 32.

List of Figures

List of Tables

List of Listings

Acronyms

2PE-SPIM Two-photon excitation selective plane illumination microscopy *(see Glossary)* 14, 48, 145

2PLSM two photon–excited fluorescence laser-scanning microscopy *(see Glossary)* 12, 14, 142, 145

AI artificial intelligence 1, 3, 5, 21, 53, 55, 111, 133

API Application Programming Interface *(see Glossary)* 80, 86, 87, 90, 135, 144, 146

BMZ BioImage Model Zoo 53–56, 58, 59, 61, 62, 64, 72, 78–81, 83, 85, 88, 90–92, 94, 95, 100, 105, 107–111, 120, 123, 135

CARE content-aware image restoration[11] 2, 17, 32, 34–38, 49, 50

CCD charge-coupled device 12

CI/CD Continuous Integration and Continuous Deployment *(see Glossary)* 84, 88–90, 94, 98, 105, 108, 136, 138

CLI Command-line interface 82, 83, 91, 98, 99, 141, 145, 146

CMOS Complementary metal–oxide–semiconductor 132

CNN convolutional neural network *(see Glossary)* 5, 24, 137, 139

CNNT Convolutional Neural Network Transformer 50

CPU central processing unit 75

DL deep learning 1–4, 6–8, 16, 17, 19, 21, 38, 44, 48, 49, 53, 54, 59, 61, 62, 75, 79, 80, 83, 105, 108, 111, 141, 144, 146

DNN deep neural network *(see Glossary)* 2

DOF depth of field 42, 43, 45

DOI Digital Object Identifier *(see Glossary)* 64, 81, 83, 91, 94, 137

dpf days postfertilization 23, 29, 32, 37

EBI European Bioinformatics Institute 28, 135, 137

eGFP enhanced green fluorescent protein (GFP) 29

ELIXIR European Life-Science Infrastructure 28, 135

EM electron microscopy 1

EMBL European Molecular Biology Laboratory 28, 135, 137

FAIR Findable, Accessible, Interoperable and Reusable *(see Glossary)* 5, 53, 62, 108, 111, 137

FOV field of view 12, 14–16, 18, 24, 25, 27–29, 34, 45–47, 49

FSF Free Software Foundation 65

GFM GitHub Flavored Markdown *(see Glossary)* 63, 117, 127, 138

GFP green fluorescent protein 29, 130

GH GitHub *(see Glossary)* 65, 79, 83–85, 87–89, 92, 105, 138, 143

GIF Graphics Interchange Format *(see Glossary)* 64, 138

GPU graphics processing unit 6, 7, 32, 42, 45, 61, 75, 95, 143, 144, 146

GUI graphical user interface *(see Glossary)* 56, 58, 107, 138, 139, 146

HDF5 Hierarchical Data Format version 5[198] 76

HTML HyperText Markup Language 141

HyLFM hybrid light-field light-sheet microscopy 16–18, 21–26, 28, 32, 36, 38, 40, 43–45, 51, 111, 113, 139

I/O input/output 42, 97

Id identifier 64, 89, 93, 94, 114, 115, 121

IDE integrated development environment 81, 82, 100, 145

IJM ImageJ Macro language *(see Glossary)* 70, 140

ISO International Organization for Standardization 71, 137

JDLL Java deep learning library 56, 70

JPEG Joint Photographic Experts Group *(see Glossary)* 64, 140

JSON JavaScript Object Notation 58, 62, 81, 82, 85, 135

LF light field 40, 42

LFD Richardson-Lucy light field deconvolution *(see Glossary)* 16, 17, 19, 26, 32, 34–41, 43, 49, 143

LFM light-field microscopy[69] 14, 16–19, 24–28, 32, 34, 37, 43–46, 48–51, 113, 130

LSCM laser SCM 12

MAE mean absolute error 30

ML machine learning 1, 4, 6, 62, 138–140, 143, 144, 146

MLA microlens array 18, 19, 21, 29, 45

MLP multilayer perceptron 5, 137

MS-SSIM multi-scale-structural similarity index measure 30, 32, 34, 36, 37, 40, 41

MSE mean squared error 5, 30

NA numerical aperture *(see Glossary)* 29, 142

NAS neural architecture search *(see Glossary)* 7, 142

NeRF neural radiance field 51

NGFF Next-generation file formats *(see Glossary)* 109, 142

NLP natural language processing 54

NMRSE normalized root mean square error 34

NN (artificial) neural network 1–7, 16, 21, 31, 32, 38, 44, 45, 47–51, 61, 68, 75, 95–97, 102, 105, 136, 137, 141–144

OME Open Microscopy Environment 109

ONNX Open Neural Network Exchange 59, 60, 68, 96

PDF Portable Document Format[199] 20

pip pip installs packages *(see Glossary)* 68, 142

PMT photomultiplier tube 12

PNG Portable Network Graphics *(see Glossary)* 64, 76, 142

PR Pull Request *(see Glossary)* 84, 85, 87, 88, 92, 105, 143

PSF point spread function 34

PSNR peak signal-to-noise ration 32, 34, 36, 37

PSTPM point-scanning two- (or three-) photon microscopy *(see Glossary)* 48, 142, 145

PyPI Python Package Index *(see Glossary)* 108, 142, 143

RAM Random Access Memory 102

RDF resource description file 61, 62, 69, 70, 72, 75, 78–82, 87–90, 95, 96, 98, 99, 102, 104–106, 108, 109, 120, 121

REPL read-eval-print loop 101

RPC remote procedure call *(see Glossary)* 140, 143

S3 (Amazon) Simple Storage Service *(see Glossary)* 90–92, 144

SCM scanning confocal microscopy 12, 14, 131, 145

sCMOS scientific CMOS 29

SemVer Semantic Versioning Specification 64, 72

SGD stochastic gradient descent *(see Glossary)* 6, 144

SHA-2 Secure Hash Algorithm 2 *(see Glossary)* 65, 66, 87, 144

SMPC secure multi-party computation 4

SNR Signal-to-noise ratio 3, 14, 15, 47, 48

SPDX Software Package Data Exchange 65

SPIM selective-plane illumination microscopy[53] 10, 12–14, 16–18, 21–30, 32, 34–39, 41, 43, 44, 47–50, 130, 139, 145

STED stimulated emission depletion (microscopy) *(see Glossary)* 12, 14, 144

TIFF Tagged Image File Format[200] 76

TIRFM total internal reflection microscopy *(see Glossary)* 12, 14, 145

TOML Tom's Obvious Minimal Language 62

TPU Tensor Processing Unit *(see Glossary)* 7, 45, 75, 144

UI user interface *(see Glossary)* 114, 115, 146

URL Uniform Resource Locator 69, 72, 75, 79, 81, 89, 113

ViT Vision Transformer 50

VRAM Video Random Access Memory 95–97, 102

W3C Word Wide Web Consortium 78

W&B Weights & Biases 30

XAI explainable AI 3

XHTML Extensible HyperText Markup Language 141

XML Extensible Markup Language 62

YAML YAML Ain't Markup Language™ 62, 71, 80, 81, 84

Glossary

JSON Schema JSON Schema[201] allows to define a specification to validate JSON data with. 82

Application Programming Interface An Application Programming Interface (API) is a part of a software program that another software program can use to access services and resources provided by that program. 80, 86, 87, 90, 135, 144, 146

asyncio The Python standard library includes the asyncio module, which provides high- and low-level APIs to control concurrent program flow[158, asyncio]. The associated `async`/`await` syntax allows to define asynchronous functions, which return an awaitable object, for example a `Future`, without having to compute the associated result[158, asyncio]. This result can be obtained (at a later stage of execution) by `await`ing the awaitable object, which blocks program flow until the result is computed[158, asyncio]. 101

BioEngine The BioEngine is a BioImage.IO specific application framework powered by ImJoy[107]. 56, 83, 100, 107

BioImage.IO community partner The BioImage.IO community partners are the groups that contribute to the development and maintenance of the BMZ. These currently are the groups behind ilastik, the DeepImageJ plugin for Fiji, ImageJ, and ImageJ2[108], ZeroCostDL4Mic, ImJoy, and Icy. 53, 70, 79, 80, 83, 87, 88, 90, 94, 95, 108–110

BioStudies A cloud storage service for biological data. It is part of the ELIXIR infrastructure and hosted by EMBL-EBI at `https://www.ebi.ac.uk/biostudies/`. 28

boilerplate Boilerplate or boilerplate code describes repeated parts of code within a project that add little functionality, yet cannot simply be removed. 80

branch Flora enthusiasts might think of trees, but in the context of programming a branch refers to a Git branch — a named pointer to Git commit. Every Git repository has a main branch, typically called "main". 85, 88, 136

chunk Input data to computations or data transmissions can be split into several blocks or "chunks" to enable parallel or sequential processing. 102, 140

Colab Colaboratory or Colab for short is a Google product offering online Python code execution in the form of a Jupyter notebook service[148]. 70, 146

commit A Git commit is a specific version of the Git repository. This version is uniquely identified by its commit hash — a long hexadecimal number. For all practical purposes the first six digits of the commit hash are typically sufficient for unambiguously referencing a Git commit. Nonetheless, Git branches are used to maintain named pointers to (an evolving chain of) commits. 136

conda "Conda is an open source package management system and environment management system that runs on Windows, macOS, and Linux. Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on your local computer. It was created for Python programs, but it can package and distribute software for any language."[144] 66, 68

context manager "A *context manager* is an object that defines the runtime context to be established when executing a `with` statement. The context manager handles the entry into, and the exit from, the desired runtime context for the execution of the block of code. [...] Typical uses of context managers include saving and restoring various kinds of global state, locking and unlocking resources, closing opened files, etc."[158, 3.3.9. With Statement Context Managers] 96

Continuous Integration and Continuous Deployment Continuous Integration (CI) in conjunction with Continuous Deployment (CD) refers to the practice of updating a code base regularly and automatically run programs on the developing code base. As a project's code changes these programs test and—if appropriate—deploy the project's software. 84, 88–90, 94, 98, 105, 108, 138

convolutional neural network A NN with convolutional layers, which implement discrete n-dimensional convolution operations. This allows to process variable input

shapes, keeps a certain neighborhood relation of the input values and typically reduces the number of parameters drastically compared to a fully connected layer found in MLPs. 5, 24, 139

Dask "Dask enables parallel and out-of-core computation. [It] couple[s] blocked algorithms with dynamic and memory aware task scheduling to achieve a parallel and out-of-core NumPy clone."[189] 102, 104, 109

Data Class (Python) Python Data Classes, introduced in Python 3.7, provide a convenient way of defining C-struct-like composite data types using type annotations[158, dataclasses]. 80, 81

DeepImageJ "DeepImageJ is a user-friendly solution that enables the generic use of pretrained deep learning models for biomedical image analysis in ImageJ"[121]. 53, 56, 70, 94, 135

Digital Object Identifier "The [Digital Object Identifier (DOI)] system provides a technical and social infrastructure for the registration and use of persistent interoperable identifiers, called DOIs, for use on digital networks."[139] The DOI Foundation[139] is the registration authority for the ISO standard 26324 "Information and documentation — Digital object identifier system"[202]. 64, 81, 83, 91, 94

EMBL-EBI Embassy Cloud The EMBL-EBI Embassy Cloud is "infrastructure-as-a-service, [...] based on the OpenStack cloud platform (http://www.embassy-cloud.org/about/)."[203] 90

Fiji "Fiji is a distribution of the popular open-source software ImageJ focused on biological-image analysis"[79]. 26, 53, 94, 135

Findable, Accessible, Interoperable and Reusable The Findable, Accessible, Interoperable and Reusable (FAIR) Guiding Principles for scientific data management and stewardship are a guideline to improve data reusability of scholarly data[204]. 5, 53, 62, 108, 111

fine-tuning Fine-tuning NNs means to continue the NN training on different data. Commonly with adapted hyperparameters like a reduced learning-rate to prevent catastrophic forgetting. 7, 75, 143

GCaMP6s A calcium indicator that fluoresces green when bound to Ca^{2+}. 29

Git "Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency."[147] 69, 136, 138

GitHub The GitHub.com platform or simply GitHub (GH) is operated by GitHub, Inc. and provides services to host software projects that are version controlled with Git[205]. It is popular among open source software development and similar to GitLab[206]. 79, 83–85, 87–89, 92, 105, 138, 143

GitHub Actions GitHub (GH) Actions is the main Continuous Integration and Continuous Deployment feature of GH. Individual GH Actions and can be combined in GitHub Actions workflows to perform more complex or customized tasks[163, GitHub Actions]. 84, 138

GitHub Actions workflow A combination of GitHub Actions constitutes a workflow to automate parts of software development[163, GitHub Actions]. 84–86, 88, 89, 91, 92, 105, 138

GitHub Flavored Markdown "GitHub Flavored Markdown, often shortened as GFM, is the dialect of Markdown that is currently supported for user content on GitHub.com and GitHub Enterprise."[207] It is a lightweight markup language whose features include to structure text with headings, (task/definition) lists, tables, code (blocks), and footnotes; among other options style text as bold, italic, highlighted or strike-through; and insert images, hyperlinks, or emojis. 63, 117

GitLab The GitLab platform by GitLab, Inc. provides services to host software projects that are version controlled with Git[206]. It is popular among open source software development and similar to GH[205]. 143

graphical user interface A graphical user interface (GUI) allows a user of a software program to interact with a visual representation of the program's state, rather than relying on text only. 56, 58, 107, 139, 146

Graphics Interchange Format The Graphics Interchange Format (GIF) is a lossless raster-graphics image file format[137]. 64

ground truth In the context of ML ground truth is the desired/expected output of a model

for a given input. It is required for supervised training, validation and testing. 3–6, 18, 21, 28, 30, 32, 36, 38, 41, 45, 47, 48, 51, 139, 144

HyLFM-Net A CNN trained for a HyLFM microscope. 16, 18, 26–28, 30–32, 34, 36–45, 47–51, 139

HyLFM-Net-beads A HyLFM-Net trained on volumes of fluorescent beads. 31, 32, 35, 38, 44

HyLFM-Net-brain A HyLFM-Net trained on individual slices of single color calcium imaging with the goal of tracking neural activity. 31, 38, 39

HyLFM-Net-dyn A HyLFM-Net trained on dynamic data — ground truth was presented in the form of single SPIM slices. 28, 30–32, 38, 42

HyLFM-Net-stat A HyLFM-Net trained on static data — volumetric ground truth has been used. 23, 28, 31, 32, 37, 38, 120

hyperparameter In the context of ML, "hyperparameters" are parameters of the model and training algorithm, which are not learned (except through metalearning), e.g. number of model layers or initial learning rate. 5, 6, 30, 137

Hypha "Hypha is an application framework for large-scale data management and AI model serving, it allows creating computational platforms consists of computational and user interface components. Hypha server act as a hub for connecting different components through imjoy-rpc."[186] 100, 107

Icy "Icy [is] a collaborative bioimage informatics platform that combines a community website [(`https://icy.bioimageanalysis.org/`)] for contributing and sharing tools and material, and software with a high-end visual programming framework for seamless development of sophisticated imaging workflows."[166] 53, 94, 135

ilastik ilastik is in interactive software tool to bring easy-to-use image analysis workflows to a wide range of users. From its intuitive GUI to programmatic use via command line integration users with various skill levels can analyze their images efficiently[124]. 53, 56, 94, 135

ImageJ ImageJ is a plug-in based image analysis software written in Java[208][108]. 53, 56, 70, 94, 135, 137, 140

ImageJ Macro language "The ImageJ Macro language (IJM) is a scripting language built into ImageJ that allows controlling many aspects of ImageJ. Programs written in the IJM, or macros, can be used to perform sequences of actions in a fashion expressed by the program's design."[209] 70

ImJoy ImJoy is an open-source computational platform that facilitates using JavaScript and Python based plugins to provide programs running in the browser[119]. 53, 56, 94, 135, 140

ImJoy RPC The ImJoy remote procedure call (RPC) was developed as part of ImJoy, but can be used as a standalone RPC library[187]. 100–102

inference In the context of ML, inference refers to computing predictions from inputs using a trained/fitted model. 7, 141

Joint Photographic Experts Group JPEG is a still image coding standard. The lossy compression JPEG 1 was created by and named after the Joint Photographic Experts Group (JPEG) in 1992 [135]. In 2001 it was extended by JPEG 2000, which supports lossy and lossless coding[210]. 64

Jupyter notebook A Jupyter notebook is a publishing format combining sections of plane text or Markdown with executable code cells. Its default kernel language is Python, but more than 50 other programming languages are supported as well[150]. 83, 105, 107, 136

knowledge distillation Knowledge distillation encompasses techniques to create a ML model from a bigger one or from an ensemble of models. 7

lazy evaluation Lazy evaluation is an evaluation strategy applied in computer programs to limit computations conducted to produce a desired output. For example, when displaying an image, to only load or compute data that lies in the field of view. To deploy this strategy data is typically processed in chunks. 102

light field "The light field is a four dimensional (4D) function representing radiance along rays as a function of position and direction in space."[87] 14, 16–19, 21–24, 26, 27, 32, 34, 36, 37, 47–49, 113, 131

mamba "Mamba is a fast, robust, and cross-platform package manager. It runs on Windows,

OS X and Linux (ARM64 and PPC64LE included) and is fully compatible with conda packages and supports most of conda's commands."[164] Beside the Python-based `mamba` CLI that provides many commands to manage conda environments the C++-based `micromamba` CLI includes the most essential of those commands[164]. 89

Markdown "Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML). Thus, "Markdown" is two things: (1) a plain text formatting syntax; and (2) a software tool, written in Perl, that converts the plain text formatting to HTML."[133, Markdown: Syntax] Markdown files are typically saved with the '.md' extension. 63, 82, 87, 98, 138, 140

Matplotlib "Matplotlib is a 2D graphics package used for Python for application development, interactive scripting, and publication-quality image generation across user interfaces and operating systems."[184] 100

mCherry A red fluorescent protein. 29

metalearning "1. A metalearning system must include a learning subsystem, which adapts with experience. 2. Experience is gained by exploiting metaknowledge extracted (a) ...in a previous learning episode on a single dataset, and/or (b) ...from different domains or problems."[211]. 6, 139

mini-batch In the context of NN training and inference, a mini-batch is a — typically small — subset of a dataset ("batch"), which is processed simultaneously as one input tensor with an additional batch dimension. 6, 75

model architecture The model or NN architecture is the combination of mathematical operations, which — given concrete model weights — allows to compute the output of the model for a given input. 2, 5–7, 26, 30, 45, 50, 60

model weights In this work model weights refers to all learnable/learned model parameters including any bias parameters. This is mostly due to the fact that "parameters" is used ubiquitously and should not be confused with hyperparameters. "Weights" is also the dominant term used in the DL community. 5, 30, 56, 59, 61, 64–66, 68, 79, 82, 87, 95, 96

neural architecture search With neural architecture search (NAS) the architecture of an

NN is determined automatically in a predefined search space. 7

Next-generation file formats Next-generation file formats (NGFF) is a specification for image file formats aiming for a unified and efficient representation of bioimage data[212, 213]. 110

numerical aperture The numerical aperture (NA) of an optical system such as a lens describes the maximum angle of light that can enter or exit whilst taking the refractiveindex into account. 29

NumPy "NumPy is the primary array programming library for the Python language. It has an essential role in research analysis pipelines in fields as diverse as physics, chemistry, astronomy, geoscience, biology, psychology, materials science, engineering, finance and economics."[151] 75, 96, 100, 102, 137, 146

overfitting A model that overfits to its training data does not generalize to unseen test data — it is fitted too closely to the training data. If a significant domain gap exists between test and training data, a model may not generalize even though it did not overfit. 4

Pandas "Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language."[214] 100

pip installs packages pip is a program to install Python packages, typically from PyPI[215]. pip stands for "pip install packages"[216]. 68

point-scanning two- (or three-) photon microscopy Point-scanning two- (or three-) photon microscopy (PSTPM) is an advanced microscopy technology for imaging highly scattering samples, for example deep in vivo tissue[217, 218]. The advantage over conventional fluorescence microscopy lies in the fact that in PSTPM fluorophores are excited by a combination of photons, thus only in the cross section of two (or more) laser beams (deep) within the sample. This avoids out-of-focus excitation by scattered illumination photons. Two photon microscopy was originally referred to as 2PLSM[64]. 48, 145

Portable Network Graphics The Portable Network Graphics (PNG) file format (and datastream) serve to exchange images with lossless compression[136]. 64, 76

pruning Pruning an NN is a popular approach to reduce the computational cost of model inference by removing parameters of a readily trained NN. Pruning is typically followed by fine-tuning to partially compensate the incurred decrease in prediction accuracy. 7

Pull Request A Pull or Merge Request is a process to collaboratively review and amend proposed changes to a code base with the intend of merging these changes. GH refers to these events as pull requests[163, Pull requests], while GitLab[206] uses the term merge request[206, Merge requests]. 84, 85, 87, 88, 92, 105

Python Package Index "The Python Package Index (PyPI) is a repository of software for the Python programming language."[194] 108, 142

Pythonic *Pythonic* means "the Python way". Python code is considered *Pythonic*, if it uses Python idioms as intended and adheres to common style guidelines[219, Code Style]. 80, 81, 96, 143

PyTorch "[PyTorch is a ML library that] provides an imperative and *Pythonic* programming style that supports code as a model, makes debugging easy and is consistent with other popular scientific computing libraries, while remaining efficient and supporting hardware accelerators such as GPUs."[47] 59, 60, 66, 80, 96, 145

refractiveindex The refractive index of an optical medium is a dimensionless number that describes how much incoming light is bend at its boundary. 29, 142

remote procedure call A remote procedure call (RPC) is a request-response protocol that allows a software program to trigger execution of procedures (subroutines) on a local or remote server and obtain the results. 140

Richardson-Lucy light field deconvolution In the context of this thesis light field deconvolution (LFD) refers to light field deconvolution deploying the Richardson-Lucy algorithm as described in [71] 16, 17, 19, 26, 32, 34, 35, 37–41, 43, 49

scikit-image "Scikit-image is an image processing library that implements algorithms and utilities for use in research, education and industry applications."[183] 100

scikit-learn "Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised

problems."[182] 100

SciPy "SciPy is an open-source scientific computing library for the Python programming language. Since its initial release in 2001, SciPy has become a de facto standard for leveraging scientific algorithms in Python [...]. SciPy is a library of numerical routines [...] that provides fundamental building blocks for modeling and solving scientific problems."[181] 100

Secure Hash Algorithm 2 The Secure Hash Algorithm 2 (SHA-2) defines a group of hash functions, namely SHA-224, SHA-256, SHA-384 and SHA-512, which output a 224 bit to 512 bit hash value, respectively[220]. 65

(Amazon) Simple Storage Service The Amazon Simple Storage Service is a data storage service developed and provided by Amazon. Other service providers are using the S3 API to offer object storage with a similar user experience. 90–92

stereopsis depth perception through binocular vision 18

stimulated emission depletion (microscopy) Stimulated emission depletion (STED) is a super-resolution method for fluorescence microscopy that allows imaging beyond the diffraction limit by depleting fluorophores around the point of excitation prior to recording each image point[63]. 12, 14

stochastic gradient descent stochastic gradient descent (SGD) is a simple but efficient optimization algorithm to train NNs with[221]. 6

supervised learning In supervised learning a ML model is fitted using pairs of input samples and known desired output (ground truth, labeled data). In the context of NNs this is also referred to as supervised training. 6, 139, 146

Tensor Processing Unit A Tensor Processing Unit (TPU) is an integrated circuit specifically designed to speed up parallelizable operations, specifically matrix multiplications, commonly used in ML algorithms, in particular in DL[222]. Alternatively GPUs can be deployed to similar effect. 7, 45, 75, 144

TensorFlow TensorFlow is a library to describe ML algorithms and execute them efficiently, especially on TPUs and GPUs[48]. 59–61, 68, 80, 96

thumbnail image A thumbnail image is a smaller version of an image used to implement

a fast image preview. The difference in loading speed is particularly noticeable for large groups of (large) images. 64

TorchScript "TorchScript is a way to create serializable and optimizable models from PyTorch code. Any TorchScript program can be saved from a Python process and loaded in a process where there is no Python dependency."[128] 96

total internal reflection microscopy Total internal reflection microscopy (TIRFM) makes use of a physical phenomenon, the evanescent wave — an electromagnetic field at the surface of a medium in which a laser is totally internally reflected[62]. This electromagnetic field decays exponentially, which results in excitation of fluorophores in a a very thin optical slice at the interface, effectively eliminating out of focus signal. 12, 14

transfer learning "Transfer learning aims at improving the performance of target learners on target domains by transferring the knowledge contained in different but related source domains."[223] 7, 75

two photon–excited fluorescence laser-scanning microscopy Rather than exciting fluorophores with a single laser as in SCM, in two photon–excited fluorescence laser-scanning microscopy two laser beams illuminate a sample point cooperatively[64]. As only the point in focus is effectively illuminated, out-of-focus signal is drastically reduced, which allows imaging deep into highly scattering tissue[64]. 2PLSM is also referred to as PSTPM 12, 14, 142, 145

Two-photon excitation selective plane illumination microscopy A SPIM combined with the concept of two-photon excitation (see 2PLSM)[65]. 14, 48

type annotations (Python) Standardized type hints are defined by the "typing" module added with Python 3.5 to the standard library[158, typing — Support for type hints]. These type annotations allow static type checkers and IDEs to support Python developers. They may also be used during runtime, for example to aid construction of a CLI with basic validation like "typer"[162] provides or to validate data by enforcing type hints like "pydantic"[159] does. 81, 82, 95, 100, 104, 105, 121, 137

Unicode Unicode, formally *The Unicode Standard* is a "[s]tandardization of graphic character sets and their characteristics, including string ordering, associated control func-

tions, their coded representation for information interchange and code extension techniques"[224]. 64, 93, 94, 114, 115

unsupervised learning In contrast to supervised learning, in unsupervised learning a ML model is fitted to unlabeled data. Goals include to represent, cluster, or mimic input data. 6

user interface A user interface (UI) is any part of a software program that allows a user to interact with the program. Examples include Command-line interfaces (CLIs) and graphical user interfaces (GUIs). 114, 115

voxel A voxel is the volumetric analog to a pixel. 28, 30, 51

WebGPU WebGPU is a JavaSript API to give programs running in a web browser access to a computer's GPU. 61

Xarray "Xarray introduces labels in the form of dimensions, coordinates and attributes on top of raw NumPy-like multidimensional arrays, which allows for a more intuitive, more concise, and less error-prone developer experience."[174] 102

Zarr "Zarr is a file storage format for chunked, compressed, N-dimensional arrays based on an open-source specification."[191] 102, 109

Zenodo The Zenodo service provides a digital research data repository at zenodo.org. It is developed and operated as part of the OpenAIRE network by CERN[125]. 58, 64, 72, 83–87, 90, 91, 94

ZeroCostDL4Mic "ZeroCostDL4Mic [is] an entry-level platform simplifying DL access by leveraging the free, cloud-based computational resources of Google Colab."[165] 53, 94, 105, 135

References

[1] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5 (1989), pp. 359–366 (cit. on pp. 2, 75).

[2] C. Angermueller et al. "Deep learning for computational biology." In: *Molecular systems biology* 12.7 (2016), p. 878 (cit. on pp. 2, 4).

[3] P. Mamoshina et al. "Applications of deep learning in biomedicine." In: *Molecular pharmaceutics* 13.5 (2016), pp. 1445–1454 (cit. on pp. 2–4).

[4] T. Ching et al. "Opportunities and obstacles for deep learning in biology and medicine." In: *Journal of The Royal Society Interface* 15.141 (2018), p. 20170387. DOI: 10.1098/rsif.2017.0387 (cit. on pp. 2–4, 7).

[5] A. M. Lucas et al. "Open-source deep-learning software for bioimage segmentation." In: *Molecular Biology of the Cell* 32.9 (2021), pp. 823–829 (cit. on p. 2).

[6] A. Hallou et al. "Deep learning for bioimage analysis in developmental biology." In: *Development* 148.18 (2021), dev199616 (cit. on p. 2).

[7] A. W. Senior et al. "Improved protein structure prediction using potentials from deep learning." In: *Nature* 577.7792 (Jan. 2020), pp. 706–710. DOI: 10.1038/s41586-019-1923-7 (cit. on p. 2).

[8] K. Tunyasuvunakool et al. "Highly accurate protein structure prediction for the human proteome." In: *Nature* 596.7873 (July 2021), pp. 590–596. DOI: 10.1038/s41586-021-03828-1 (cit. on p. 2).

[9] U. Schmidt et al. "Cell detection with star-convex polygons." In: *Medical Image Computing and Computer Assisted Intervention–MICCAI 2018: 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part II 11*. Springer. 2018, pp. 265–273 (cit. on pp. 2, 95, 102, 103).

[10] M. Weigert et al. "Star-convex polyhedra for 3D object detection and segmentation in microscopy." In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2020, pp. 3666–3673 (cit. on pp. 2, 99).

[11] M. Weigert et al. "Content-aware image restoration: pushing the limits of fluorescence microscopy." In: *Nature Methods* 15.12 (Nov. 2018), pp. 1090–1097. DOI: 10.1038/s41592-018-0216-7 (cit. on pp. 2, 3, 21, 49, 129).

[12] A. Krull, T.-O. Buchholz, and F. Jug. "Noise2Void - Learning Denoising From Single Noisy Images." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019 (cit. on p. 2).

[13] J. Lehtinen et al. "Noise2Noise: Learning Image Restoration without Clean Data." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2965–2974 (cit. on p. 2).

[14] N. Sapoval et al. "Current progress and open challenges for applying deep learning across the biosciences." In: *Nature Communications* 13.1 (Apr. 2022). DOI: 10.1038/s41467-022-29268-7 (cit. on pp. 2, 4).

[15] J. Gawlikowski et al. "A survey of uncertainty in deep neural networks." In: *Artificial Intelligence Review* 56.Suppl 1 (2023), pp. 1513–1589 (cit. on p. 2).

[16] M. Van Lent, W. Fisher, and M. Mancuso. "An explainable artificial intelligence system for small-unit tactical behavior." In: *Proceedings of the national conference on artificial intelligence*. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. 2004, pp. 900–907 (cit. on p. 3).

[17] M. Du, N. Liu, and X. Hu. "Techniques for interpretable machine learning." In: *Communications of the ACM* 63.1 (Dec. 2019), pp. 68–77. DOI: 10.1145/3359786 (cit. on p. 3).

[18] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso. "Machine Learning Interpretability: A Survey on Methods and Metrics." In: *Electronics* 8.8 (July 2019), p. 832. DOI: 10.3390/electronics8080832 (cit. on p. 3).

[19] A. Holzinger et al. "Explainable AI Methods - A Brief Overview." In: *xxAI - Beyond Explainable AI: International Workshop, Held in Conjunction with ICML 2020, July 18, 2020, Vienna, Austria, Revised and Extended Papers*. Ed. by A. Holzinger et al. Cham: Springer International Publishing, 2022, pp. 13–38. ISBN: 978-3-031-04083-2. DOI: 10.1007/978-3-031-04083-2_2 (cit. on p. 3).

[20] M. El Helou and S. Süsstrunk. "Bigprior: Toward decoupling learned prior hallucination and data fidelity in image restoration." In: *IEEE Transactions on Image Processing* 31 (2022), pp. 1628–1640 (cit. on p. 3).

[21] R. Cohen et al. "Looks Too Good To Be True: An Information-Theoretic Analysis of Hallucinations in Generative Restoration Models." In: *arXiv preprint arXiv:2405.16475* (2024) (cit. on p. 3).

[22] A. Krogh and J. Hertz. "A simple weight decay can improve generalization." In: *Advances in neural information processing systems* 4 (1991) (cit. on p. 4).

[23] N. Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958 (cit. on pp. 4, 5).

[24] V. Gulshan et al. "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs." In: *Jama* 316.22 (2016), pp. 2402–2410 (cit. on p. 4).

[25] D. Erhan et al. "Why Does Unsupervised Pre-training Help Deep Learning?" In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 201–208. URL: https://proceedings.mlr.press/v9/erhan10a.html (cit. on p. 4).

[26] R. Gilad-Bachrach et al. "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy." In: *International conference on machine learning*. PMLR. 2016, pp. 201–210 (cit. on p. 4).

[27] S. Wagh, D. Gupta, and N. Chandran. "Securenn: Efficient and private neural network training." In: *Cryptology ePrint Archive* (2018) (cit. on p. 4).

[28] F. Regazzoni et al. "SECURED for Health: Scaling Up Privacy to Enable the Integration of the European Health Data Space." In: *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. Mar. 2024, pp. 1–4. DOI: 10.23919/date58400.2024.10546514. URL: http://dx.doi.org/10.23919/DATE58400.2024.10546514 (cit. on p. 4).

[29] F. Levet et al. "Developing open-source software for bioimage analysis: opportunities and challenges." In: *F1000Research* 10 (Apr. 2021), p. 302. DOI: 10.12688/f1000research.52531.1. URL: https://doi.org/10.12688/f1000research.52531.1 (cit. on p. 5).

[30] "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 5).

[31] O. Ronneberger, P. Fischer, and T. Brox. "U-net: Convolutional networks for biomedical image segmentation." In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pp. 234–241 (cit. on pp. 5, 27).

[32] K. He et al. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on p. 5).

[33] G. Huang et al. "Densely connected convolutional networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708 (cit. on p. 5).

[34] A. Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale." In: *arXiv preprint arXiv:2010.11929* (2020) (cit. on pp. 5, 50).

[35] A. Vaswani et al. "Attention is all you need." In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 5).

[36] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by F. Bach and D. Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 448–456. URL: https://proceedings.mlr.press/v37/ioffe15.html (cit. on p. 5).

[37] Y. Wu and K. He. "Group Normalization." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018 (cit. on p. 5).

[38] C. Szegedy et al. "Rethinking the inception architecture for computer vision." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826 (cit. on p. 5).

[39] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 6).

[40] S. Albert et al. "Comparison of Image Normalization Methods for Multi-Site Deep Learning." In: *Applied Sciences* 13.15 (2023). ISSN: 2076-3417. DOI: 10.3390/app13158923. URL: https://www.mdpi.com/2076-3417/13/15/8923 (cit. on p. 6).

[41] D. Blalock et al. "What is the state of neural network pruning?" In: *Proceedings of machine learning and systems* 2 (2020), pp. 129–146 (cit. on p. 7).

[42] J. Gou et al. "Knowledge distillation: A survey." In: *International Journal of Computer Vision* 129 (2021), pp. 1789–1819 (cit. on p. 7).

[43] S. Gupta et al. "Deep Learning with Limited Numerical Precision." In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by F. Bach and D. Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1737–1746. URL: https://proceedings.mlr.press/v37/gupta15.html (cit. on p. 7).

[44] M. Courbariaux, Y. Bengio, and J.-P. David. "Training deep neural networks with low precision multiplications." In: *arXiv preprint arXiv:1412.7024* (2014) (cit. on p. 7).

[45] T. Elsken, J. H. Metzen, and F. Hutter. "Neural Architecture Search: A Survey." In: *Journal of Machine Learning Research* 20.55 (2019), pp. 1–21. URL: http://jmlr.org/papers/v20/18-598.html (cit. on p. 7).

[46] F. Zhuang et al. "A Comprehensive Survey on Transfer Learning." In: *Proceedings of the IEEE* 109.1 (2021), pp. 43–76. DOI: 10.1109/JPROC.2020.3004555 (cit. on p. 7).

[47] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on pp. 7, 26, 59, 143).

[48] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on pp. 7, 59, 144).

[49] R. Hooke. *Micrographia: or, Some physiological descriptions of minute bodies made by magnifying glasses*. J. Martyn and J. Allestry, 1665 (cit. on pp. 9, 10).

[50] S. Fowler, R. Roush, and J. Wise. "How Cells Are Studied." In: *Concepts of Biology*. Houston, Texas: OpenStax, 2013. URL: https://openstax.org/books/concepts-biology/pages/3-1-how-cells-are-studied (cit. on pp. 9, 11).

[51] F. BALUsKA. "Eukaryotic Cells and their Cell Bodies: Cell Theory Revised." In: *Annals of Botany* 94.1 (May 2004), pp. 9–32. ISSN: 1095-8290. DOI: 10.1093/aob/mch109. URL: http://dx.doi.org/10.1093/aob/mch109 (cit. on p. 9).

[52] V. Wood et al. "Hidden in plain sight: what remains to be discovered in the eukaryotic proteome?" In: *Open Biology* 9.2 (Feb. 2019), p. 180241. ISSN: 2046-2441. DOI: 10.1098/rsob.180241. URL: http://dx.doi.org/10.1098/rsob.180241 (cit. on p. 9).

[53] J. Huisken et al. "Optical Sectioning Deep Inside Live Embryos by Selective Plane Illumination Microscopy." In: *Science* 305.5686 (2004), pp. 1007–1009. DOI: 10.1126/science.1100035. eprint: https://www.science.org/doi/pdf/10.1126/science.1100035. URL: https://www.science.org/doi/abs/10.1126/science.1100035 (cit. on pp. 10, 133).

[54] P. Davidovits and M. D. Egger. "Scanning laser microscope." In: *Nature* 223.5208 (1969), pp. 831–831 (cit. on p. 12).

[55] J. Pawley. *Handbook of biological confocal microscopy*. Vol. 236. Springer Science & Business Media, 2006 (cit. on p. 12).

[56] L. D. Favro et al. "Confocal microscope." 5162941. Nov. 1992. URL: https://patents.google.com/patent/US5162941A (visited on 08/27/2024) (cit. on p. 12).

[57] J. Icha et al. "Phototoxicity in live fluorescence microscopy, and how to avoid it." In: *BioEssays* 39.8 (2017), p. 1700003 (cit. on pp. 12, 13).

[58] B.-C. Chen et al. "Lattice light-sheet microscopy: Imaging molecules to embryos at high spatiotemporal resolution." In: *Science* 346.6208 (Oct. 2014). ISSN: 1095-9203. DOI: 10.1126/science.1257998. URL: http://dx.doi.org/10.1126/science.1257998 (cit. on pp. 12, 14).

[59] M. J. Booth. "Adaptive optics in microscopy." In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 365.1861 (2007), pp. 2829–2843 (cit. on p. 12).

[60] N. Ji. "Adaptive optical fluorescence microscopy." In: *Nature methods* 14.4 (2017), pp. 374–380 (cit. on p. 12).

[61] E. J. AMBROSE. "A Surface Contact Microscope for the study of Cell Movements." In: *Nature* 178.4543 (Nov. 1956), pp. 1194–1194. ISSN: 1476-4687. DOI: 10.1038/1781194a0. URL: http://dx.doi.org/10.1038/1781194a0 (cit. on p. 14).

[62] D. Axelrod. "Cell-substrate contacts illuminated by total internal reflection fluorescence." In: *The Journal of cell biology* 89.1 (Apr. 1981), pp. 141–145. ISSN: 1540-8140. DOI: 10.1083/jcb.89.1.141. URL: http://dx.doi.org/10.1083/jcb.89.1.141 (cit. on pp. 14, 145).

[63] S. W. Hell and J. Wichmann. "Breaking the diffraction resolution limit by stimulated emission: stimulated-emission-depletion fluorescence microscopy." In: *Optics Letters* 19.11 (June 1994), p. 780. ISSN: 1539-4794. DOI: 10.1364/ol.19.000780. URL: http://dx.doi.org/10.1364/OL.19.000780 (cit. on pp. 14, 144).

[64] W. Denk, J. H. Strickler, and W. W. Webb. "Two-photon laser scanning fluorescence microscopy." In: *Science* 248.4951 (1990), pp. 73–76 (cit. on pp. 14, 142, 145).

[65] Z. Lavagnino et al. "Two-photon excitation selective plane illumination microscopy (2PE-SPIM) of highly scattering samples: characterization and application." In: *Opt. Express* 21.5 (Mar. 2013), pp. 5998–6008. DOI: 10.1364/OE.21.005998. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-21-5-5998 (cit. on pp. 14, 48, 145).

[66] W. C. Lemon and K. McDole. "Live-cell imaging in the era of too many microscopes." In: *Current opinion in cell biology* 66 (2020), pp. 34–42 (cit. on p. 14).

[67] M. Brameshuber et al. "Understanding immune signaling using advanced imaging techniques." In: *Biochemical Society Transactions* 50.2 (2022), pp. 853–866 (cit. on pp. 14, 15).

[68] P. P. Laissue et al. "Assessing phototoxicity in live fluorescence imaging." In: *Nature methods* 14.7 (2017), pp. 657–661 (cit. on p. 14).

[69] M. Levoy et al. "Light field microscopy." In: *ACM SIGGRAPH 2006 Papers*. 2006, pp. 924–934 (cit. on pp. 14, 18, 19, 43, 45, 49, 131).

[70] M. Broxton et al. "Wave optics theory and 3-D deconvolution for the light field microscope." In: *Opt. Express* 21.21 (Oct. 2013), pp. 25418–25439. DOI: 10.1364/OE.21.025418. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-21-21-25418 (cit. on p. 16).

[71] R. Prevedel et al. "Simultaneous whole-animal 3D imaging of neuronal activity using light-field microscopy." In: *Nature methods* 11.7 (2014), pp. 727–730 (cit. on pp. 16, 19, 143).

[72] A. Stefanoiu et al. "Artifact-free deconvolution in light field microscopy." In: *Opt. Express* 27.22 (Oct. 2019), pp. 31644–31666. DOI: 10.1364/OE.27.031644. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-27-22-31644 (cit. on p. 16).

[73] N. Wagner et al. "Instantaneous isotropic volumetric imaging of fast biological processes." In: *Nature methods* 16.6 (2019), pp. 497–500 (cit. on pp. 16, 41, 47).

[74] N. Wagner*, F. Beuttenmueller* et al. "Deep learning-enhanced light-field imaging with continuous validation." In: *Nature Methods* 18.5 (May 2021), pp. 557–563. DOI: 10.1038/s41592-021-01136-0 (cit. on pp. 16, 22, 24–27, 29–31, 33–42, 46).

[75] C. Guo et al. "Fourier light-field microscopy." In: *Optics express* 27.18 (2019), pp. 25573–25594 (cit. on pp. 19, 46).

[76] W. H. Richardson. "Bayesian-based iterative method of image restoration." In: *JoSA* 62.1 (1972), pp. 55–59 (cit. on pp. 19, 51).

[77] L. B. Lucy. "An iterative technique for the rectification of observed distributions." In: *Astronomical Journal, Vol. 79, p. 745 (1974)* 79 (1974), p. 745 (cit. on pp. 19, 51).

[78] N. Viganò et al. "Tomographic approach for the quantitative scene reconstruction from light field images." In: *Opt. Express* 26.18 (Sept. 2018), pp. 22574–22602. DOI: 10.1364/OE.26.022574. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-26-18-22574 (cit. on p. 19).

[79] J. Schindelin et al. "Fiji: an open-source platform for biological-image analysis." In: *Nature Methods* 9.7 (June 2012), pp. 676–682. DOI: 10.1038/nmeth.2019 (cit. on pp. 26, 137, 166).

[80] S. Preibisch et al. "Efficient Bayesian-based multiview deconvolution." In: *Nature methods* 11.6 (2014), pp. 645–648 (cit. on p. 26).

[81] M. Jaderberg, K. Simonyan, A. Zisserman, et al. "Spatial transformer networks." In: *Advances in neural information processing systems* 28 (2015) (cit. on p. 26).

[82] G. Brancato et al. "Dual fluorescence through Kasha's rule breaking: An unconventional photomechanism for intracellular probe design." In: *The Journal of Physical Chemistry B* 119.20 (2015), pp. 6144–6154 (cit. on p. 28).

[83] D. Kim et al. "Dual-color fluorescent nanoparticles showing perfect color-specific photoswitching for bioimaging and super-resolution microscopy." In: *Nature communications* 10.1 (2019), p. 3089 (cit. on p. 28).

[84] *ELIXIR*. URL: https://elixir-europe.org/ (visited on 04/28/2023) (cit. on p. 28).

[85] G. M. Hale and M. R. Querry. "Optical constants of water in the 200-nm to 200-μm wavelength region." In: *Applied optics* 12.3 (1973), pp. 555–563 (cit. on p. 29).

[86] L. Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/ (cit. on p. 30).

[87] M. Levoy, Z. Zhang, and I. McDowall. "Recording and controlling the 4D light field in a microscope using microlens arrays." In: *Journal of microscopy* 235.2 (2009), pp. 144–162 (cit. on pp. 46, 140).

[88] L. Cong et al. "Rapid whole brain imaging of neural activity in freely behaving larval zebrafish (Danio rerio)." In: *elife* 6 (2017), e28158 (cit. on p. 46).

[89] J. P. Vizcaino et al. "Real-time light field 3D microscopy via sparsity-driven learned deconvolution." In: *2021 IEEE International Conference on Computational Photography (ICCP)*. IEEE. 2021, pp. 1–11 (cit. on p. 46).

[90] J. Swoger et al. “Multi-view image fusion improves resolution in three-dimensional microscopy.” In: *Optics express* 15.13 (2007), pp. 8029–8042 (cit. on p. 47).

[91] N. Wijethilake et al. “DEEP-squared: deep learning powered De-scattering with Excitation Patterning.” In: *Light: Science & Applications* 12.1 (2023), p. 228 (cit. on p. 48).

[92] R. Guo et al. “EventLFM: Event camera integrated Fourier light field microscopy for ultrafast 3D imaging.” In: *Light: Science & Applications* 13.1 (2024), p. 144 (cit. on pp. 49, 50).

[93] H. W. F. Yeung et al. “Light Field Spatial Super-Resolution Using Deep Efficient Spatial-Angular Separable Convolution.” In: *IEEE Transactions on Image Processing* 28.5 (2019), pp. 2319–2330. DOI: 10.1109/TIP.2018.2885236 (cit. on p. 49).

[94] H. W. F. Yeung et al. “Fast Light Field Reconstruction With Deep Coarse-To-Fine Modeling of Spatial-Angular Clues.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018 (cit. on p. 49).

[95] J. P. Vizcaíno et al. “Learning to Reconstruct Confocal Microscopy Stacks From Single Light Field Images.” In: *IEEE Transactions on Computational Imaging* 7 (2021), pp. 775–788. DOI: 10.1109/TCI.2021.3097611 (cit. on p. 49).

[96] N. Meng et al. “High-Dimensional Dense Residual Convolutional Neural Network for Light Field Reconstruction.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.3 (2021), pp. 873–886. DOI: 10.1109/TPAMI.2019.2945027 (cit. on p. 49).

[97] Y. Xue et al. “Deep-learning-augmented computational miniature mesoscope.” In: *Optica* 9.9 (2022), pp. 1009–1021 (cit. on p. 50).

[98] Z. Lu et al. “Virtual-scanning light-field microscopy for robust snapshot high-resolution volumetric imaging.” In: *Nature Methods* 20.5 (2023), pp. 735–746 (cit. on p. 50).

[99] A. Rehman et al. “Convolutional neural network transformer (CNNT) for fluorescence microscopy image denoising with improved generalization and fast adaptation.” In: *Scientific Reports* 14.1 (2024), p. 18184 (cit. on p. 50).

[100] K. He et al. "Masked autoencoders are scalable vision learners." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 16000–16009 (cit. on p. 50).

[101] S. A. Eslami et al. "Neural scene representation and rendering." In: *Science* 360.6394 (2018), pp. 1204–1210 (cit. on pp. 50, 51).

[102] B. Mildenhall et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." In: *Communications of the ACM* 65.1 (2021), pp. 99–106 (cit. on p. 51).

[103] V. Sitzmann et al. "Deepvoxels: Learning persistent 3d feature embeddings." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2437–2446 (cit. on p. 51).

[104] A. Pumarola et al. "D-NeRF: Neural Radiance Fields for Dynamic Scenes." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 10318–10327 (cit. on p. 51).

[105] J. Ost et al. "Neural Scene Graphs for Dynamic Scenes." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 2856–2865 (cit. on p. 51).

[106] *AI4Life*. URL: https://ai4life.eurobioimaging.eu/ (visited on 09/29/2024) (cit. on p. 53).

[107] W. Ouyang*, F. Beuttenmueller*, E. Gómez-de-Mariscal*, C. Pape* et al. "BioImage Model Zoo: A Community-Driven Resource for Accessible Deep Learning in BioImage Analysis." In: *bioRxiv* (June 2022). DOI: 10.1101/2022.06.07.495102 (cit. on pp. 53–56, 83, 87, 107, 135).

[108] C. T. Rueden et al. "ImageJ2: ImageJ for the next generation of scientific image data." In: *BMC bioinformatics* 18 (2017), pp. 1–26 (cit. on pp. 53, 56, 94, 135, 139).

[109] T. Wolf et al. "Transformers: State-of-the-Art Natural Language Processing." In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://aclanthology.org/2020.emnlp-demos.6 (cit. on p. 54).

[110] R. Chard et al. "DLHub: Model and data serving for science." In: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2019, pp. 283–292 (cit. on p. 54).

[111] The PyTorch team. *PyTorch Hub*. URL: https://www.pytorch.org/hub (visited on 03/27/2023) (cit. on p. 54).

[112] The TensorFlow team. *TensorFlow Hub*. URL: https://www.tensorflow.org/hub (visited on 03/27/2023) (cit. on p. 54).

[113] J. Y. Koh. *Model Zoo*. URL: https://modelzoo.co/ (visited on 03/27/2023) (cit. on p. 54).

[114] M. J. Cardoso et al. "MONAI: An open-source framework for deep learning in healthcare." In: *arXiv preprint arXiv:2211.02701* (2022) (cit. on p. 54).

[115] *BioImage Model Zoo: Advanced AI models in one-click*. Developed at [225]. URL: https://bioimage.io/ (visited on 02/20/2023) (cit. on pp. 55, 57, 83, 92).

[116] F. Beuttenmueller, W. Ouyang, et al. *BioImage.IO RDF collections*. deprecated in favor of [117]. URL: https://github.com/bioimage-io/collection-bioimage-io (visited on 03/08/2023) (cit. on pp. 55, 83).

[117] F. Beuttenmueller, J. Metz, D. Kutra, et al. *bioimage.io collection*. URL: https://github.com/bioimage-io/collection (visited on 09/27/2024) (cit. on pp. 55, 83, 91, 158).

[118] F. Beuttenmueller et al. *bioimageio.spec — Specifications for bioimage.io*. URL: https://github.com/bioimage-io/spec-bioimage-io (visited on 09/27/2024) (cit. on pp. 56, 62).

[119] W. Ouyang et al. "ImJoy: an open-source computational platform for the deep learning era." In: *Nature Methods* 16.12 (Nov. 2019), pp. 1199–1200. DOI: 10.1038/s41592-019-0627-0 (cit. on pp. 56, 100, 140).

[120] *Documentation Neural Network Classification*. The Neural Network Classification workflow is included in ilastik version 1.4.0b1 and newer. For ilastik see [124]. URL: https://www.ilastik.org/documentation/nn/nn (visited on 02/21/2023) (cit. on pp. 56, 95, 96).

[121] E. Gómez-de-Mariscal et al. "DeepImageJ: A user-friendly environment to run deep learning models in ImageJ." In: *Nature Methods* 18.10 (Sept. 2021), pp. 1192–1195. DOI: 10.1038/s41592-021-01262-9 (cit. on pp. 56, 137).

[122] F. Beuttenmueller et al. *bioimageio.core: Python specific core utilities for running models in the BioImage Model Zoo*. URL: https://github.com/bioimage-io/core-bioimage-io-python (visited on 09/27/2024) (cit. on pp. 56, 76–78).

[123] C. García López de Haro et al. "JDLL: a library to run deep learning models on Java bioimage informatics platforms." In: *Nature Methods* 21.1 (2024), pp. 7–8 (cit. on pp. 56, 95).

[124] S. Berg et al. "ilastik: interactive machine learning for (bio)image analysis." In: *Nature Methods* (Sept. 2019). ISSN: 1548-7105. DOI: 10.1038/s41592-019-0582-9 (cit. on pp. 58, 94, 139, 158).

[125] European Organization For Nuclear Research and OpenAIRE. *Zenodo*. en. 2013. DOI: 10.25495/7GXK-RD71. URL: https://www.zenodo.org/ (cit. on pp. 58, 146).

[126] ONNX community. https://onnx.ai/. June 18, 2022. URL: https://github.com/onnx/onnx/releases/tag/v1.12.0 (visited on 10/21/2022) (cit. on p. 60).

[127] The PyTorch team. *Serialization Semantics*. June 28, 2022. URL: https://pytorch.org/docs/1.12/notes/serialization.html (visited on 10/21/2022) (cit. on p. 60).

[128] The PyTorch team. *TorchScript*. June 28, 2022. URL: https://pytorch.org/docs/1.12/jit.html (visited on 10/21/2022) (cit. on pp. 60, 96, 145).

[129] D. Smilkov et al. "TensorFlow.js: Machine Learning For The Web and Beyond." In: *Proceedings of Machine Learning and Systems*. Ed. by A. Talwalkar, V. Smith, and M. Zaharia. Vol. 1. 2019, pp. 309–321. URL: https://proceedings.mlsys.org/paper/2019/file/1d7f7abc18fcb43975065399b0d1e48e-Paper.pdf (cit. on p. 61).

[130] *The JavaScript Object Notation (JSON) Data Interchange Format*. Tech. rep. Dec. 2017. DOI: 10.17487/rfc8259 (cit. on p. 62).

[131] I. döt Net et al. *YAML Ain't Markup Language (YAML™) version 1.2*. Revision 1.2.2 (2021-10-01). Oct. 1, 2021. URL: https://yaml.org/spec/1.2.2/ (visited on 10/21/2022) (cit. on pp. 62, 80).

[132] F. Beuttenmueller et al. *BioImage.IO Model Resource Description File Specification 0.4.9*. URL: https://github.com/bioimage-io/spec-bioimage-io/blob/gh-pages/model_spec_0_4.md (visited on 02/22/2023) (cit. on p. 62).

[133] J. Gruber. *Markdown 1.0.1*. Dec. 2004. URL: https://daringfireball.net/projects/markdown/ (visited on 03/08/2023) (cit. on pp. 63, 141).

[134] Github, Inc. Oct. 17, 2022. URL: https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/quickstart-for-writing-on-github (visited on 10/24/2022) (cit. on p. 63).

[135] ISO/IEC JTC 1/SC 29 Coding of audio, picture, multimedia and hypermedia information. *Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines*. Standard 10918-1:1994. Geneva, CH: International Organization for Standardization, Feb. 1994 (cit. on pp. 64, 140).

[136] ISO/IEC JTC 1/SC 24 Computer graphics, image processing and environmental data representation. *Information technology — Computer graphics and image processing — Portable Network Graphics (PNG): Functional specification*. Standard 15948:2004. Geneva, CH: International Organization for Standardization, Mar. 2004 (cit. on pp. 64, 142).

[137] CompuServe, Inc. *GRAPHICS INTERCHANGE FORMAT™ Version 89a*. July 1989. URL: https://www.w3.org/Graphics/GIF/spec-gif89a.txt (visited on 02/28/2023) (cit. on pp. 64, 138).

[138] imageio. documentation of [175]. Feb. 27, 2023. URL: https://imageio.readthedocs.io/en/v2.26.0/index.html (visited on 02/28/2023) (cit. on pp. 64, 164).

[139] International DOI Foundation. May 13, 2021. URL: https://www.doi.org/ (visited on 10/24/2022) (cit. on pp. 64, 137).

[140] European Organization For Nuclear Research and OpenAIRE. *Zenodo*. en. 2013. DOI: 10.25495/7GXK-RD71. URL: https://help.zenodo.org/ (visited on 10/24/2022) (cit. on p. 64).

[141] SemVer Community. *Semantic Versioning*. URL: https://semver.org/ (visited on 10/24/2022) (cit. on p. 64).

[142] SPDX Community. The SPDX Workgroup is a Linux Foundation Project. Aug. 12, 2022. URL: https://spdx.org/licenses/ (visited on 10/24/2022) (cit. on p. 65).

[143] F. S. F. (Licensing and C. Lab. Jan. 12, 2022. URL: https://www.gnu.org/licenses/license-list.en.html (visited on 10/24/2022) (cit. on p. 65).

[144] *Anaconda Software Distribution*. Version Vers. 2-2.4.0. 2020. URL: https://docs.anaconda.com/ (cit. on pp. 66, 136).

[145] T. pip developers. *pip documentation: Requirements Files*. URL: https://pip.pypa.io/en/stable/user_guide/#requirements-files (visited on 02/14/2023) (cit. on p. 68).

[146] Anaconda, Inc. *Managing environments: Sharing an environment*. URL: https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#sharing-an-environment (visited on 02/14/2023) (cit. on p. 68).

[147] *Git website*. URL: https://git-scm.com/ (visited on 10/24/2022) (cit. on pp. 69, 138).

[148] E. Bisong and E. Bisong. "Google colaboratory." In: *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners* (2019), pp. 59–64 (cit. on pp. 70, 136).

[149] ISO/TC 154 — Processes, data elements and documents in commerce, industry and administration. *Date and time — Representations for information interchange — Part 2: Extensions*. Standard 8601-2:2019. Geneva, CH: International Organization for Standardization, Feb. 2019 (cit. on p. 71).

[150] T. Kluyver et al. “Jupyter Notebooks - a publishing format for reproducible computational workflows.” In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Scmidt. Netherlands: IOS Press, 2016, pp. 87–90. URL: https://eprints.soton.ac.uk/403913/ (cit. on pp. 72, 83, 140).

[151] C. R. Harris et al. “Array programming with NumPy.” In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2 (cit. on pp. 75, 142).

[152] R. Kern. *NEP 1 — A simple file format for NumPy arrays*. URL: https://numpy.org/neps/nep-0001-npy-format.html (visited on 09/29/2024) (cit. on p. 75).

[153] F. Beuttenmueller et al. *Preprocessing operations in model spec 0.4*. URL: https://github.com/bioimage-io/spec-bioimage-io/blob/gh-pages/preprocessing_spec_0_4.md (visited on 02/22/2023) (cit. on pp. 76–78).

[154] F. Beuttenmueller et al. *Postprocessing operations in model spec 0.4*. URL: https://github.com/bioimage-io/spec-bioimage-io/blob/gh-pages/postprocessing_spec_0_4.md (visited on 02/22/2023) (cit. on pp. 76–78).

[155] J. Carroll and G. Klyne. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. W3C, Feb. 2004 (cit. on p. 78).

[156] A. van der Neut. *ruamel.yaml is a YAML 1.2 parser/emitter for Python*. URL: https://sourceforge.net/projects/ruamel-yaml/ (visited on 03/04/2023) (cit. on pp. 80, 81).

[157] S. Loria et al. *marshmallow: simplified object serialization*. URL: https://github.com/marshmallow-code/marshmallow (visited on 03/02/2023) (cit. on p. 80).

[158] P. S. Foundation. *Python 3.11.2 documentation*. URL: https://docs.python.org/3/index.html (visited on 03/05/2023) (cit. on pp. 80, 135–137, 145).

[159] S. Colvin et al. *Pydantic: Data validation using Python type hints*. URL: https://github.com/pydantic/pydantic (visited on 03/05/2023) (cit. on pp. 81, 145).

[160] S. J. Fuhry, S. Rogovskiy, S. Pike, et al. *marshmallow-jsonschema: JSON Schema formatting with marshmallow*. 2023. URL: https://github.com/fuhrysteve/marshmallow-jsonschema (cit. on p. 82).

[161] *JSON Schema Store*. URL: https://www.schemastore.org/json/ (visited on 09/29/2024) (cit. on p. 82).

[162] S. Ramírez. *Typer, build great CLIs. Easy to code. Based on Python type hints*. URL: https://github.com/tiangolo/typer (visited on 03/05/2023) (cit. on pp. 82, 145).

[163] GitHub, Inc. *GitHub Docs*. documents [205]. 2023. URL: https://docs.github.com/en (visited on 03/08/2023) (cit. on pp. 84, 87, 89, 138, 143, 166).

[164] QuantStack & mamba contributors. *Mamba: The Fast Cross-Platform Package Manager*. URL: https://mamba.readthedocs.io/en/latest/user_guide/micromamba.html (visited on 03/23/2023) (cit. on pp. 89, 141).

[165] L. v. Chamier et al. "ZeroCostDL4Mic: an open platform to use Deep-Learning in Microscopy." In: *BioRxiv* (2020), pp. 2020–03 (cit. on pp. 94, 105, 146).

[166] F. De Chaumont et al. "Icy: an open bioimage informatics platform for extended reproducible research." In: *Nature methods* 9.7 (2012), pp. 690–696 (cit. on pp. 94, 139).

[167] C. Garcia-López-de-Haro et al. *Java Library for Deep Learning*. URL: https://github.com/bioimage-io/model-runner-java (visited on 03/06/2023) (cit. on p. 95).

[168] T. Burke et al. *bioimage.io specification implementation for Java*. URL: https://github.com/bioimage-io/model-runner-java (visited on 03/06/2023) (cit. on p. 95).

[169] H. Krekel et al. *pytest 7.2*. 2004. URL: https://github.com/pytest-dev/pytest (visited on 03/07/2023) (cit. on p. 95).

[170] A. Wolny et al. "Accurate and versatile 3D segmentation of plant tissues at cellular resolution." In: *Elife* 9 (2020), e57613 (cit. on pp. 95, 163).

[171] L. Cerrone, Q. Yu, A. Wolny, et al. *PlantSeg 2.0*. [170]. URL: https://github.com/kreshuklab/plant-seg (visited on 09/29/2024) (cit. on p. 95).

[172] M. Novikov et al. *tiktorch*. URL: https://github.com/ilastik/tiktorch (visited on 03/07/2023) (cit. on p. 96).

[173] S. Hoyer and J. Hamman. "xarray: N-D labeled arrays and datasets in Python." In: *Journal of Open Research Software* 5.1 (2017). DOI: 10.5334/jors.148 (cit. on p. 96).

[174] S. Hoyer, C. Fitzgerald, J. Hamman, et al. *xarray: v2023.02.0*. Feb. 2023. DOI: 10.5281/zenodo.7616056 (cit. on pp. 96, 102, 146).

[175] A. Klein et al. *imageio/imageio: v2.26.0*. Version v2.26.0. documented by [138]. Feb. 2023. DOI: 10.5281/zenodo.7679453 (cit. on pp. 96, 160).

[176] T. F. Chan, G. H. Golub, and R. J. LeVeque. "Updating formulae and a pairwise algorithm for computing sample variances." In: *COMPSTAT 1982 5th Symposium held at Toulouse 1982: Part I: Proceedings in Computational Statistics*. Springer. 1982, pp. 30–41 (cit. on p. 97).

[177] T. Dunning. "The t-digest: Efficient estimates of distributions." In: *Software Impacts* 7 (2021), p. 100049. ISSN: 2665-9638. DOI: 10.1016/j.simpa.2020.100049. URL: https://www.sciencedirect.com/science/article/pii/S2665963820300403 (cit. on p. 97).

[178] J. Crist-Harif et al. *Crick*. URL: https://github.com/dask/crick (visited on 03/06/2023) (cit. on p. 97).

[179] M. Pachitariu and C. Stringer. "Cellpose 2.0: how to train your own model." In: *Nature methods* 19.12 (2022), pp. 1634–1641 (cit. on p. 99).

[180] T. E. Oliphant. "Python for Scientific Computing." In: *Computing in Science & Engineering* 9.3 (2007), pp. 10–20. DOI: 10.1109/MCSE.2007.58 (cit. on p. 100).

[181] P. Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2 (cit. on pp. 100, 144).

[182] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 100, 144).

[183] S. van der Walt et al. "scikit-image: image processing in Python." In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453 (cit. on pp. 100, 143).

[184] J. D. Hunter. "Matplotlib: A 2D graphics environment." In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55 (cit. on pp. 100, 141).

[185] W. McKinney et al. "Data structures for statistical computing in python." In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. 1. Austin, TX. 2010, pp. 51–56 (cit. on p. 100).

[186] W. Ouyang et al. *Hypha*. URL: https://github.com/imjoy-team/hypha (visited on 03/08/2023) (cit. on pp. 100, 139).

[187] W. Ouyang, M. Hjelmare, and M. McCormick. *ImJoy RPC*. URL: https://github.com/imjoy-team/imjoy-rpc (visited on 03/09/2023) (cit. on pp. 100, 102, 140).

[188] E. S. Raymond. *The cathedral and the bazaar*. 2000. URL: http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ (visited on 09/16/2024) (cit. on p. 100).

[189] Dask Development Team. *Dask: Library for dynamic task scheduling*. 2016. URL: https://dask.org (visited on 03/09/2023) (cit. on pp. 102, 137).

[190] M. Durant. *async-zarr*. URL: https://github.com/martindurant/async-zarr (visited on 03/09/2023) (cit. on p. 102).

[191] A. Miles et al. *zarr-developers/zarr-python: v2.14.2*. 2023. DOI: 10.5281/ZENODO.7675396. URL: https://zenodo.org/record/7675396 (cit. on pp. 102, 146).

[192] F. Beuttenmueller. *BioImage.IO Workflows*. URL: https://github.com/bioimage-io/workflows-bioimage-io-python (visited on 03/17/2023) (cit. on pp. 104, 121).

[193] Google, Inc. *Google Python Style Guide*. URL: https://google.github.io/styleguide/pyguide.html (visited on 03/17/2023) (cit. on pp. 104, 121).

[194] *Python Package Index - PyPI*. URL: https://pypi.org/ (visited on 03/03/2023) (cit. on pp. 108, 143).

[195] Conda-Forge Community. *The conda-forge Project: Community-based Software Distribution Built on the conda Package Format and Ecosystem*. 2015. DOI: 10.5281/ZENODO.4774217. URL: https://zenodo.org/record/4774217 (cit. on p. 108).

[196] J. Moore et al. "OME-Zarr: a cloud-optimized bioimaging file format with international community support." In: *Histochemistry and Cell Biology* 160.3 (July 2023), pp. 223–251. ISSN: 1432-119X. DOI: 10.1007/s00418-023-02209-1. URL: http://dx.doi.org/10.1007/s00418-023-02209-1 (cit. on p. 109).

[197] M. E. Corporation. *Microscope Objective Lenses*. 2024. URL: https://www.olympus-lifescience.com/en/objectives/ (visited on 09/06/2024) (cit. on p. 113).

[198] The HDF Group. *Hierarchical Data Format, version 5*. https://www.hdfgroup.org/HDF5/. 1997–2023 (cit. on p. 130).

[199] ISO/TC 171/SC 2 — Document file formats, EDMS systems and authenticity of information. *Document management — Portable document format — Part 1: PDF 1.7*. Standard 32000-1:2008. Geneva, CH: International Organization for Standardization, July 2008 (cit. on p. 132).

[200] G. Parsons and J. Rafferty. *Tag Image File Format (TIFF) - image/tiff MIME Sub-type Registration*. RFC 3302. RFC Editor, Sept. 2002 (cit. on p. 133).

[201] *json-schema.org*. URL: https://json-schema.org/ (visited on 09/29/2024) (cit. on p. 135).

[202] ISO/TC 46/SC 9 Identification and description. *Information and documentation — Digital object identifier system*. Standard 26324:2022. Geneva, CH: International Organization for Standardization, Aug. 2022 (cit. on p. 137).

[203] C. E. Cook et al. "The European Bioinformatics Institute in 2017: data coordination and integration." In: *Nucleic acids research* 46.D1 (2018), pp. D21–D29 (cit. on p. 137).

[204] M. D. Wilkinson et al. "The FAIR Guiding Principles for scientific data management and stewardship." In: *Scientific data* 3.1 (2016), pp. 1–9 (cit. on p. 137).

[205] GitHub, Inc. *GitHub*. documented by [163]. URL: https://github.com/ (visited on 03/08/2023) (cit. on pp. 138, 163).

[206] GitLab, Inc. *GitLab Docs*. URL: https://docs.gitlab.com/ (visited on 03/08/2023) (cit. on pp. 138, 143).

[207] GitHub, Inc. *GitHub Flavored Markdown Spec*. 2023. URL: https://github.github.com/gfm/ (visited on 03/08/2023) (cit. on p. 138).

[208] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri. "NIH Image to ImageJ: 25 years of image analysis." In: *Nature methods* 9.7 (2012), pp. 671–675 (cit. on pp. 139, 166).

[209] J. Mutterer and W. Rasband. *ImageJ Macro Language Programmer's Reference Guide v1.46d*. based on ImageJ[208] and Fiji[79]. URL: https://imagej.nih.gov/ij/docs/macro_reference_guide.pdf (visited on 03/01/2023) (cit. on p. 140).

[210] ISO/IEC JTC 1/SC 29 Coding of audio, picture, multimedia and hypermedia information. *Information technology — JPEG 2000 image coding system — Part 4: Conformance Testing*. Standard 15444-4:2021. Geneva, CH: International Organization for Standardization, Oct. 2021 (cit. on p. 140).

[211] C. Lemke, M. Budka, and B. Gabrys. "Metalearning: a survey of trends and technologies." In: *Artificial intelligence review* 44 (2015), pp. 117–130 (cit. on p. 141).

[212] *Next-generation file format (NGFF) specifications for storing bioimaging data in the cloud.* DOI: 10.5281/zenodo.4282107. URL: https://ngff.openmicroscopy.org/0.4/ (visited on 09/16/2024) (cit. on p. 142).

[213] J. Moore et al. "OME-NGFF: a next-generation file format for expanding bioimaging data-access strategies." In: *Nature Methods* 18.12 (Nov. 2021), pp. 1496–1498. ISSN: 1548-7105. DOI: 10.1038/s41592-021-01326-w. URL: http://dx.doi.org/10.1038/s41592-021-01326-w (cit. on p. 142).

[214] The pandas development team. *pandas-dev/pandas: Pandas*. Version 1.5.3. Jan. 2023. DOI: 10.5281/zenodo.7549438 (cit. on p. 142).

[215] T. pip developers. *pip - The Python Package Installer*. Ed. by P. P. Authority. URL: https://github.com/pypa/pip (visited on 03/03/2023) (cit. on p. 142).

[216] I. Bicking. *pyinstall is dead, long live pip!* Oct. 28, 2008. URL: https://ianbicking.org/blog/2008/10/pyinstall-is-dead-long-live-pip.html (visited on 03/03/2023) (cit. on p. 142).

[217] J. V. Rocheleau and W. David. "Piston. 2003."Two-Photon Excitation Microscopy for the Study of Living Cells and Tissues."" In: *Current Protocols in Cell Biology* 20.1 () (cit. on p. 142).

[218] M. Yildirim et al. "Functional imaging of visual cortical layers and subplate in awake mice with optimized three-photon microscopy." In: *Nature communications* 10.1 (2019), p. 177 (cit. on p. 142).

[219] K. Reitz and T. Schlusser. *The Hitchhiker's guide to Python: best practices for development*. O'Reilly Media, Inc., 2016. URL: https://docs.python-guide.org/ (visited on 03/02/2023) (cit. on p. 143).

[220] W. Penard and T. van Werkhoven. "On the secure hash algorithm family." In: *Cryptography in context* (2008), pp. 1–18 (cit. on p. 144).

[221] L. Bottou. "Large-scale machine learning with stochastic gradient descent." In: *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*. Springer. 2010, pp. 177–186 (cit. on p. 144).

[222] *Introduction to Cloud TPU*. URL: https://cloud.google.com/tpu/docs/tpus (visited on 09/20/2024) (cit. on p. 144).

[223] F. Zhuang et al. "A Comprehensive Survey on Transfer Learning." In: *Proceedings of the IEEE* 109.1 (2021), pp. 43–76. DOI: 10.1109/JPROC.2020.3004555 (cit. on p. 145).

[224] ISO/IEC JTC 1/SC 2 — Coded character sets. *Information technology — Universal coded character set (UCS)*. Standard 10646:2020. Geneva, CH: International Organization for Standardization, Dec. 2020 (cit. on p. 146).

[225] W. Ouyang et al. *BioImage.IO [website development]*. 2019. URL: https://github.com/bioimage-io/biomage.io (visited on 02/20/2023) (cit. on p. 158).