university of groningen

faculty of science and engineering

Towards meta-learning based, domain specifc AutoML systems in the example domain of cellular image analyses

Jan-Henner Roberg

university of groningen / faculty of science and engineering

University of Groningen

Towards meta-learning based, domain specific AutoML systems in the example domain of cellular image analyses

Master's Thesis

To fulfill the requirements for the degree of
Master of Science in Artificial Intelligence
at University of Groningen under the supervision of
Lars Leyendecker, MSc. (Fraunhofer Institute for Production Technology)
and
Prof. dr. Herbert Jaeger (Artificial Intelligence, University of Groningen)

Jan-Henner Roberg (s3228851)

December 29, 2022

Contents

Acknowledgments

This thesis represents the largest project I have completed so far, therefore I would like to thank some people that made it possible. First I would like to thank my supervisor Lars Leyendecker from Fraunhofer IPT for his continuous support in a relaxed but goal driven working environment. Moreover, I would like to thank Fraunhofer IPT for allowing me to do my thesis embedded in a larger applied research project and the provided compute resources. Furthermore, a big thank you to everyone in the AIxCell team I learned a lot by working together on this rather large software project. I would also like to thank my internal supervisor Prof. Herbert Jaeger for his advice and feedback. Lastly, I would like to thank my girlfriend Camilla and my parents for their support during each stage of the project.

Abstract

Biomedical analyses of cellular images is a time consuming and costly process. For many analyses tasks, semantic segmentation is the first step. Deep Learning has shown great promise in automating this task and is used in existing tools for cellular image analyses. These tools are being used by biomedical experts and thereby facilitate cellular research. However, they rely on default models for the segmentation task and do not tune hyperparameters on a given dataset. This can result in suboptimal performance. In this thesis we propose an automated machine learning (AutoML) system to tune the hyperparameters of a DL-pipeline for semantic segmentation of cellular images.

The system is based on meta-learning and leverages meta data to efficiently find a suitable pipeline for a new dataset. Meta data here refers to DL-pipelines evaluated on different datasets. To quantify datasets, different meta-features are extracted. These meta-features are concatenated with hyperparameter configurations and used as input to a machine learning model. This model is trained on the meta base and learns to predict how well hyperparameter configurations will perform on different datasets. After training, this model can then predict a ranking of hyperparameter configurations for a new dataset. Based on the predicted ranking successive halving is used to determine the best hyperparameter configuration for a dataset. In addition to the AutoML system, the effects of generic and domain specific transfer learning on model performance and convergence times are evaluated.

For transfer learning the results indicate no significant effect of pre-training on performance and convergence times. On one of the four evaluation datasets generic pre-training lead to worse results compared to all other conditions. Otherwise, there was no difference between generic and domain-specific pre-training.
The results of the meta-learning experiments suggest that classical meta-features are most important. Meta-features aiming to quantify the optical appearance of images were not used in the best meta-learning pipeline. A random forest model outperformed LamdaMART and other evaluated meta-models. To evaluate the overall system, it was tested on four cellular segmentation datasets for which human experts had already developed DL-pipelines. Overall, the developed system performs slightly worse compared to the existing DL-pipelines. On all datasets, the validation set IoU score of the AutoML system was 0.04-0.084 worse compared to the existing pipelines.

1 Introduction

In 2021 alone 79 zetabytes of data were created, captured, copied, and consumed worldwide[1]. This amount is expected to increase significantly in the coming years, more than doubling until 2025. With the overall amount of data increasing, methods to automatically extract valuable information from data are also becoming increasingly important. One powerful tool for this task is machine learning [1]. Machine learning (ML) refers to algorithms that learn from historical data, to predict outputs for new data [1]. First, it enables the use of large data and can be used to extract information that humans are not able to. Secondly, it can be used to automate tasks that usually require humans and thereby save a lot of valuable time.
However, in order to leverage ML, an ML-expert is needed to configure an ML-pipeline for a specific task. As stated in the "no free lunch" theorem [2], there is no single ML algorithm that performs best on all tasks. Therefore, to achieve optimal performance, individual ML systems have to be designed for each task. The growing field of automated machine learning (AutoML) aims to automate this construction of ML-pipelines for new tasks [3]. Most research within AutoML focuses on developing general methods that work well for a variety of tasks and datasets.
While general methods are desirable, since they have the largest applicability, they also increase the difficulty of developing high performing AutoML systems. To automatically construct general deep learning systems for large datasets with instances of high dimensionality is especially challenging. Therefore, limiting the application domain can be beneficial since this limits the number of possible solutions. Moreover, restricting the domain increases the possibility of using meta-learning. Meta-learning or 'learning-to-learn' refers to leveraging previous experience to learn how to perform a new task. Limiting the application domain leads to more similar datasets, thereby the experience gained from evaluating pipelines on them is more valuable. One domain where automatically constructing a deep learning pipeline can be especially useful is cellular image analysis.

Understanding how cells work in healthy or diseased states allows researchers in animal, plant and medical sciences to develop vaccines, more effective medicines and more resilient plants. In general, biomedical research into cellular functions will increase our understanding of living things [4]. One common way of investigating cells is to analyze microscopy images. This analysis then gives insights into the effects of chemical and genetical perturbations on phenotypes of cell cultures.
Advances in microscopy technology have enabled more high-thorough-put experiments to be performed [5]. In this way, more and more images of cell cultures are generated. For cell researchers this means that they can study biological systems at larger scales and it allows to perform more experiments.
This segmentation is a time consuming process that requires human experts and thereby limits cellular-research. Deep learning has shown great promise in automating this task. Human experts only need to annotate some images and the deep learning model then performs the task for other images.
The first step in cellular image analyses is often to segment regions of interest (ROIs) in an image. This segmentation is a time consuming process that requires human experts and thereby limits cellular-research. Deep learning has shown great promise in automating this task [6]. Human experts only need to annotate some images and the deep learning model then performs the task for other images. Currently, it is common practice for human experts to design DL-pipelines for different datasets. There are systems such as Italisk [7] and CellProfiler [8] that offer algorithms that can learn the segmentation task without the need for a human expert. However, they rely on default workflows and do not optimize hyperparameters. Tuning the hyperparameters of deep learning pipelines is required

[1] https://www.statista.com/statistics/871513/worldwide-data-created/

for optimal performance [9]. Therefore, an AutoML system for hyperparameter tuning is desirable in this context.

1.1 The AIxCell Research Project

To enable biomedical experts to utilise the advances in DL for their cell analysis task, the "AIxCell" Research Project[2] is developing an AutoML- and DL-based cell culture image analysis tool. Most recent AutoML systems are designed for tabular data, only a few systems have been developed for image data and these are mostly proprietary [3]. This research project aims to develop an AutoML system for automating biomedical image analyses. The research project is funded by the German Federal Ministry of Economics (BMWi) and a consortium of companies, university hospitals and the Fraunhofer Institute for Production Technology (Fraunhofer IPT, Aachen). The presented thesis was done in the context of the AIxCell research project and supervised by Fraunhofer IPT.
In the first stage of the project annotated datasets and corresponding analyses tasks were provided by the project partners. DL solutions were then developed for these use-cases. In addition, a plug-in deep learning library (PIDLL) that allows for easy configuration and training of DL-pipelines was developed. The PIDLL allows to configure four different steps of the DL-pipeline: Pre-processing, Feature engineering, Deep-Learning Model and Post-processing. For cellular image analyses the first three steps influence how the segmentation mask is learned, Post-processing is then a final step that allows to perform different analysis tasks based on the segmentation masks. This final step can be manually configured by the user, the presented AutoML system focuses on configuring the first three steps to determine an optimal pipeline for segmentation.
This thesis is part of the second stage of the project, developing the AutoML system to automatically construct DL-pipelines. This system relies on the PIDLL and was developed based on insights gained during the first stage of the project. Namely, the AutoML system searches for optimal pipelines in a search space configured based on experience gained during the first stage. Moreover, the PIDLL is used to configure and train pipelines. The final tool also has a user interface for image annotation and is deployed on the cloud. An overview of the developed cell culture analyses tool can be seen in Figure 1.

1.2 Research Questions

This thesis is about developing the AutoML system for the AIxCell tool. Even though the system was developed for cellular image segmentation, the methodology and research is relevant for other domain specific AutoML systems.
In general, predicting an optimal DL-pipeline for biomedical images is challenging due to diverse images, multiple image modalities and approaching new tasks [10]. Even when the image modality is limited to microscopic cellular images, it is still a challenging due to:

1. Long model evaluation times: Because the optimal pipeline needs to be found within a given compute budget, this restricts the system to only fully evaluate a few pipelines on a new dataset.

2. Diverse datasets: Even when limiting the domain, images and segmentation tasks can vary significantly between datasets.

3. A small meta-base: The long evaluation times impose a limit on the amount of DL-pipelines that can be evaluated to collect meta-data.

[2] https://www.forschung-fom.de/forschung/projekte-und-vorhaben/d/s/AIxCell

An additional challenge, which is however not specific to AutoML for images, is the quantification of tasks and their similarity. To tackle these questions, this thesis investigates transfer learning to reduce evaluation times and combat small datasets, meta-learning and optimal meta-features for image datasets and a small meta-base, and finally multi-fidelity methods to efficiently evaluate configurations.
To summarize, this thesis investigates the following main research question (MRQ) divided into four sub research questions (SRQ):

MRQ: How can meta-learning based approaches be used to optimize performance of domain-specific AutoML systems for image data, at the example of cellular image analyses?

SRQ1: Can pre-training on a domain-specific dataset decrease model training time and increase performance, is this better compared to pretraining on a generic dataset?

SRQ2: What are good meta-features to represent image datasets, in a domain-specific setting?

SRQ3: Given a small meta-base, what are optimal ML-models and feature engineering strategies to predict a ranking of DL-pipelines for a new task?

SRQ4: Can successive halving be used to efficiently evaluate a ranking of configurations on a dataset, within a given compute budget?

1.3 Thesis Outline

In the first chapter, an introduction to the topic and a description of the AIxCell system has been given. Moreover, the research questions have been formulated. The second chapter details cellular image analyses and describes how computer vision is used to tackle this problem. Deep learning for image segmentation is discussed and hyperparameters specific to cellular image analyses are presented. Then, an overview over the field of AutoML is given, the focus being on meta-learning and multi-fidelity methods. Chapter two gives a description of radiomics, a potential tool to quantify the optical appearance of image datasets. In chapter three related work is presented and the current state-of-the-art is defined. In chapter four the different components of the overall AutoML system are described. Furthermore, experiments to investigate and optimize each component are described. All datasets used in the presented study are discussed and the evaluation methodology of the final system is given. Chapter five presents the results of all experiments, these results are further discussed in chapter six. Finally, in chapter seven the main contributions of this thesis are highlighted and potential future work is discussed.

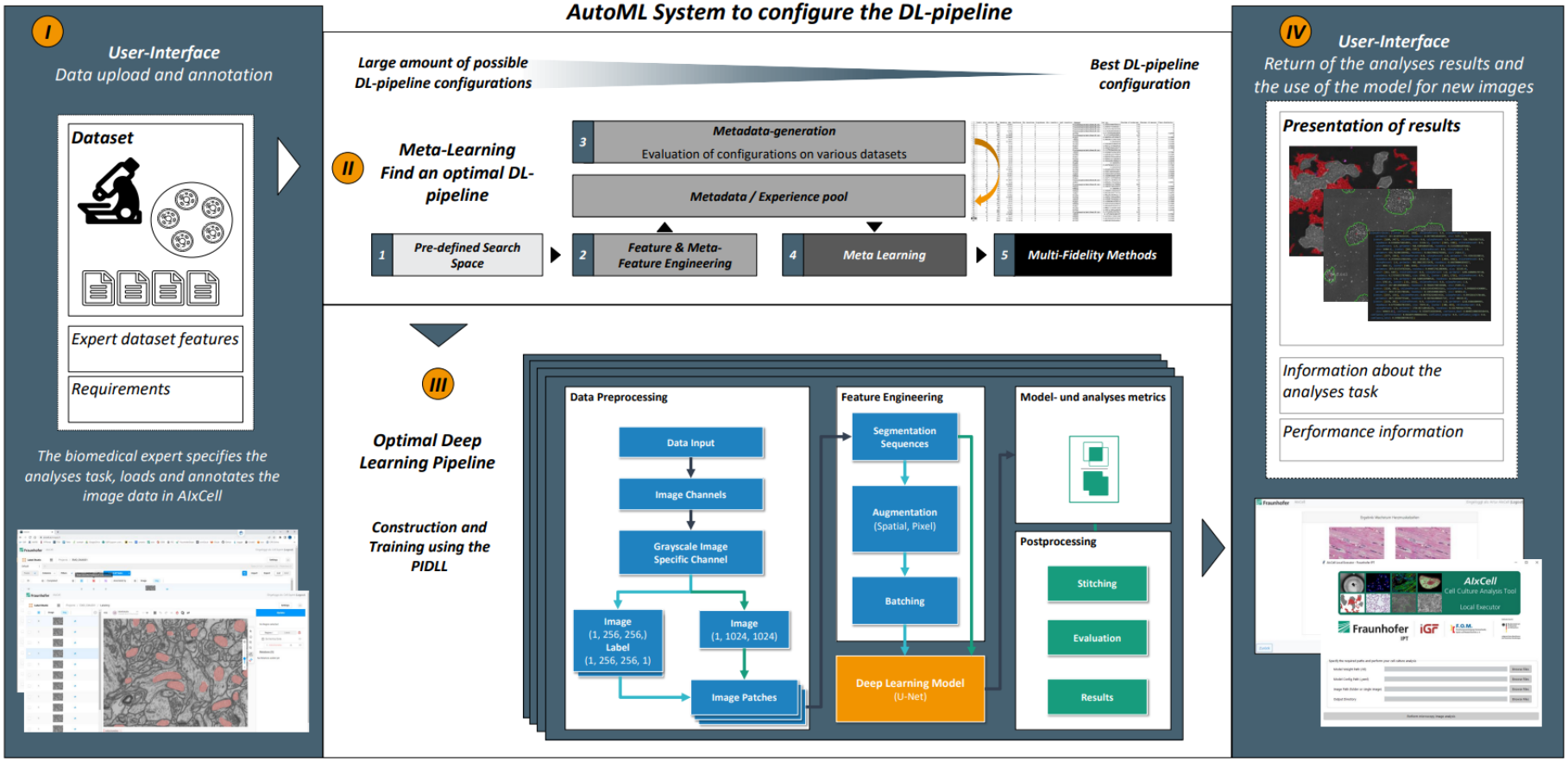
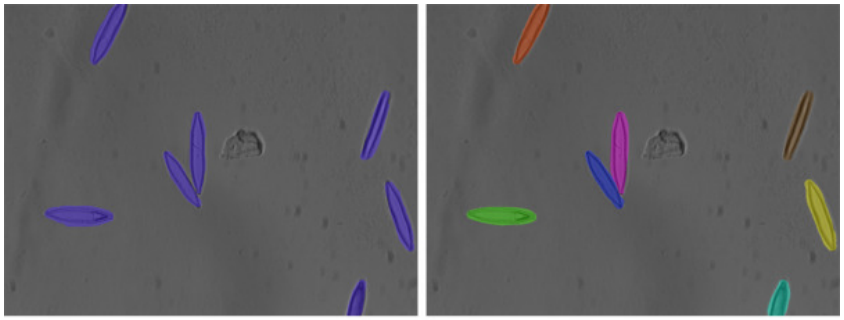


Figure 1: Overview of the AIxCell system. In the first step the user interface allows to upload and annotate a dataset. Moreover, the user specifies the analyses task. Secondly, the AutoML system configures an optimal DL-pipeline using a meta-learning based approach to sequentially limit the overall search space. Thirdly that pipeline is then constructed and trained using the PIDLL. Finally, some example images and the trained model are returned to the user. This model can then be used to perform the analyses task on new images.

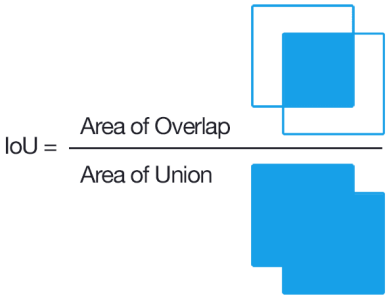
2 Theoretical Foundations

Here the theoretical background to the presented research is discussed. In Section 2.1, image segmentation and approaches for this task are discussed. In Section 2.2 transfer learning is described. In Section 2.3, the field of AutoML, with a focus on meta-learning and multi-fidelity methods, is presented. Finally Section 2.4 describes radiomics as a potential tool to quantify image data.

2.1 Image Segmentation



(a) Semantic segmentation mask. (b) Instance segmentation mask.



(a) Illustration of instance and semantic segmentation, image taken from [11]

(b) Illustration of the IoU score.

Figure 2

There are two types of segmentation, instance and semantic segmentation [11]. Instance segmentation refers to segmenting all instances of an object class separately. In semantic segmentation the task is to assign each instance of an object class the same label. This difference is illustrated in Figure 2a. Both tasks require different approaches, in instance segmentation the challenge is to separate overlapping instances and in semantic segmentation the focus is on segmenting different ROI's. In this thesis, the focus is on semantic segmentation.
One key question in image segmentation is how to evaluate the performance of different approaches. The IoU score is a classic metric to assess the performance of a segmentation algorithm. It is calculated by dividing the intersection between the predicted and true mask by the union of these regions. This is best understood visually and shown in Figure 2b. In this way, the IoU score ranges from 0-1, with zero indicating that the object has not been detected at all and one indicating a perfect match between the prediction and ground truth.

2.1.1 Classic Computer Vision

For a long time research focused on classic computer vision techniques for image segmentation. One simple way of segmenting regions of interest is using a threshold on the image. In gray-scale images, each pixel has a value from 0-255. If the ROI's fall into different scales on this range a threshold can be effectively used for segmentation. The crucial task here is to find the optimal threshold value. Otsu thresholding is a technique to automatically find this value for binary segmentation tasks. This is done by minimizing the following equation to find an optimal value T [12]:

$$\sigma_w^2(T) = \sigma_o^2(T) * p_o^2(T) * \sigma_b^2(T) * p_b^2(T)$$

Here σ_o^2 and σ_b^2 refer to the variance in the foreground and background pixels respectively and p_o^2 and p_b^2 refer to the probability of pixels belonging to the fore and background. The variances and probability's are calculated for each image, with the probabilities being the ratio of pixels belonging to foreground or background over all pixels in an image. All these depend on the threshold value T. While this technique works well for binary segmentation, given a clear threshold exists, in many scenarios there is no clear threshold.
Another popular classic computer vision approach for image segmentation is the watershed algorithm [13]. Here the image is treated as a topographic map, classically defined by the pixel intensity values. The algorithm then starts at the basins of this map and simulates them filling with water. In this way the ridges of the topographic map are identified. Therefore, the algorithm identifies regions that separate objects as the places where two water basins meet (at the ridges). The separate water basins then correspond to identified ROI's.
These are only two classical techniques. There are many more such as k-means clustering [14], active contour models [15] or conditional and Markov random fields [16]. To a degree, all these classical techniques focus on identifying or extracting useful features in an image and using these identified features for the segmentation task. Earlier techniques such as the Otsu threshold use features present in the image already, e.g. the pixel intensities. More advanced techniques such as watershed transform or active contour models apply a predefined operation to an image for feature extraction.

2.1.2 Deep Learning

In 2012, the field of computer vision changed when AlexNet [17], a deep neural network, significantly outperformed competing algorithms at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [18]. As the name states, this challenge is about image classification and not about image segmentation. Image classification and segmentation are different tasks, with segmentation being harder to solve by automated algorithms. The reason for this is not fully understood, however, there are some potential reasons. Segmentation is pixel-wise classification, while in classification many pixels are used to classify a single object. Thereby, in classification more task specific information is available. Moreover, contrary to classification, segmentation requires precise spatial predictions. Nonetheless, both tasks entail pixel data and research then focused on using deep learning for image segmentation [19, 20]. Today deep learning models significantly outperform classical vision methods for image segmentation tasks.
As stated in the previous section, classical computer vision research focused on extracting features from images and then using these features to perform some task. Instead of manually defining features to be extracted from an image, in deep learning the idea is to have a neural network learn representations that are important for a specific task [21].
These representations are learned from labeled data, for example in semantic segmentation images where a human expert has already segmented the ROI's. The field of supervised machine learning refers to models that learn from annotated data $D = (x_i, y_i); 1 \leq i \leq n$. The goal of these models is to learn a mapping from an input x_i to an output y_i. More specifically the goal is to accurately predict outputs for data that is not part of the set of data used to train the algorithm. To this end the dataset D is commonly split into 3 subsets:

- The *training set* used to train the model
- The *validation set* used to tune model hyper-parameters to optimize performance
- The *test set* used to evaluate the final model

One particularly powerful subset of machine learning models are neural networks, black-box models that learn input to output mappings.
Here a multi layer perceptron (MLP) will be briefly described to illustrate these models. A MLP consist of layers of connected neurons, each neuron is represented by a numerical value, commonly called its weight. After each layer the resulting output is passed through an activation function. This ensures that a neural network can learn non-linear input to output mappings. When an input is presented to this network it is sequentially multiplied with the weights in each layer. Mathematically the output of one layer is calculated the following way:

$$x^k = \sigma(W^k * x^{k-1} + b^k)$$

Here x^k is the output of the layer k, if $k = 0$ x is the input to the network. W^k represents the weights, b^k the bias of layer k and σ is some non-linear activation function. So if a MLP has k layers the output of the network is calculated by:

$$\hat{y} = \sum_{k=1}^{k} \sigma(W^k * x^{k-1} + b^k)$$

This output is then compared to the desired output and the weights W are updated depending on the similarity of produced output to the desired one. How this similarity is computed depends on the loss function, one classical example is the mean squared error: $MSE = 1/n \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$. Here n is the number of examples in the dataset, y_i is the desired output and $\hat{y}_i$ is the predicted output.
The goal of the network is to minimize the loss function. This is done by computing the gradients of all weights W in the network w.r.t. the loss of a given input x. In other words, how much did each weight contribute to the error and how should they be adjusted to minimize the error. The algorithm to efficiently compute these gradients is called back-propagation. At each iteration, when an input has been fed through the network and the output has been computed, all weights of the networks are updated based on these gradients. The overall training procedure is called gradient descent.
This automated training also explains why neural networks are black-box models. It is impossible to determine why a local minima of W in the loss function leads to good performance. Thereby, it is impossible to untangle the learned input-to-output mapping and the decisions of a neural network cannot be explained. It is however known that different representations of the data are learned at each layer and that the network learns to perform tasks that way [21].

2.1.3 Convolutional Neural Networks

Based on the MLP many variants of neural networks have been developed [21]. Among others, these include convolutional neural networks specifically designed for pixel data. The main components of convolutional neural networks are convolution layers and pooling (subsampling) layers. At each convolutional layer, several small windows (e.g. size 3x3) slide over the input image, at each step computing a weighted sum of the covered area according to the weights of the window components, resulting in an output image (possibly reduced in size) in which one of those features is highlighted. This process is illustrated in Figure 3. These windows are called filters or kernels, since it filters a feature from a complete image. The output of a convolutional layer is therefore called a feature map. The values of these kernels are the weights of the convolutional neural network and are learned during the training process. Usually images contain three channels (RGB) and feature maps later in the network contain even more channels. For each layer in the output feature map (see Figure 3 on the right) different kernel weights are used. The weights of the kernel (see Figure 3 in the middle) used to compute an output layer stay the same at every location of the image. In this way, the weights in a

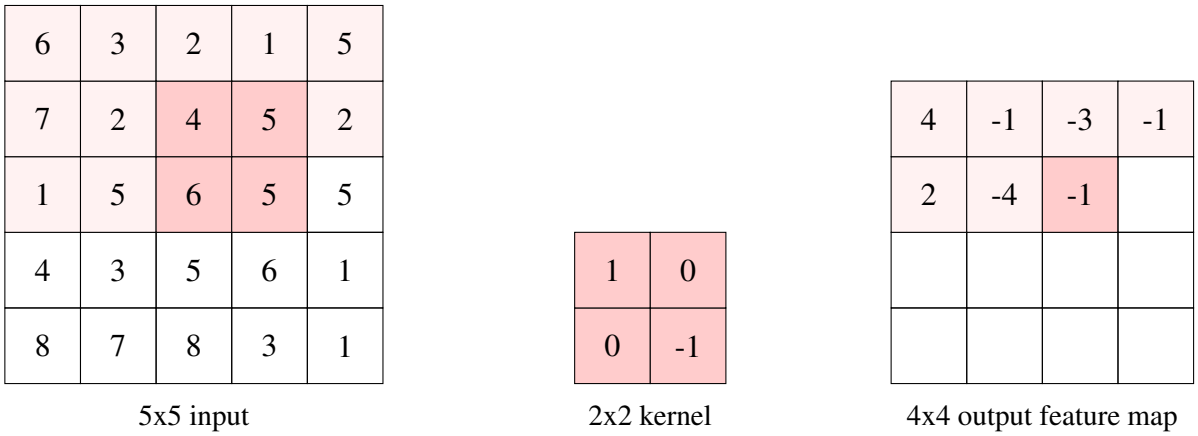


Figure 3: How a kernel slides over the input in a convolutional layer. The weighted sum of the values covered by the kernel with the weights of the kernel is computed and put into the corresponding pixel of the output feature map.

convolutional neural network are shared. This has two benefits, first it reduces the overall number of trainable parameters, second it leads to the same feature detector being used in all parts of the image [21].
A convolutional neural network consists of stacks of these convolution layers followed by a pooling layer. In the pooling layer again a window slides over the image, however, there are no trainable weights in the window. Instead the pooling layer always applies the same operation, the two most common ones being max and average pooling. In max pooling the largest value in a window is taken and in average pooling the average of all values in a window is taken. Usually pooling layer are used with window dimensions and strides that lead to a significant reduction of the feature map.
By repeatedly applying convolutions and pooling some representation necessary to perform a given task is learned. In classification tasks, a fully connected layer that takes the final representation as input and outputs a probability distribution over the classes, is used to decide on a class based on the learned representation. For image segmentation encoder-decoder networks are used.

2.1.4 U-Net

The idea behind encoder-decoder networks is to encode the image to some lower dimension and then to decode this lower dimension to the original size. Thereby, the network learns a representation that is important for the segmentation task and then decodes that representation to segment the ROI's. A crucial component to make this work is to pass information about the original image and higher level encoding to the decoding stage of the network [20].

In U-Net an image is first encoded into a representation of the same height and width using convolution layers. That representation is then compressed using max-pooling. This procedure is repeated until some final compressed representation of the input is reached. In the decoding stage of the network that representation is up-sampled using up-convolution layers. At each stage of this up-sampling process the corresponding representation on the encoding stage is also copied as input to the decoding layer. In this way, during the decoding stage the network has access to same size representations of the original image. This also applies to the segmentation mask. Here the original input is also used as input to the final decoding layer. The intuition behind this process is that, since the network is doing pixel wise classification, information about the individual pixels are needed at each decoding step [20]. In Figure 4, the described network-architecture is illustrated, here one can also see the U-shape of the network that led to its name.

There are more variants of the described U-Net encoder-decoder architecture and also other architectures have been developed for segmentation [22]. Nonetheless, especially for biomedical images, U-Net has been shown to perform well on a variety of datasets.

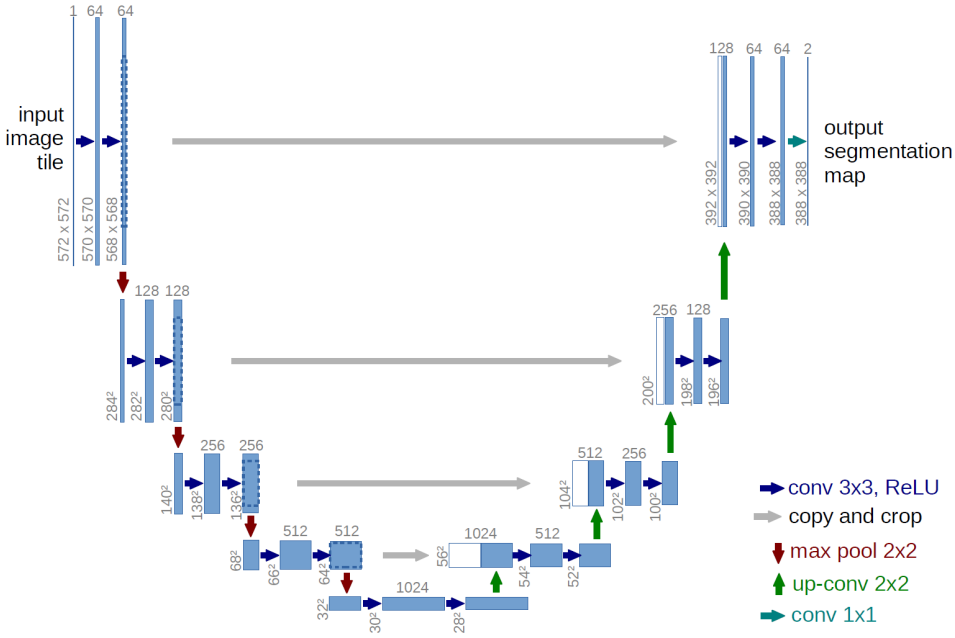


Figure 4: The U-Net architecture, image taken from [20]. The gray numbers next to each feature map indicate the sizes of the convolution layers. The vertical numbers indicate the width and height, the horizontal number the depth of the layer. Operations applied between the layers are illustrated to the right.

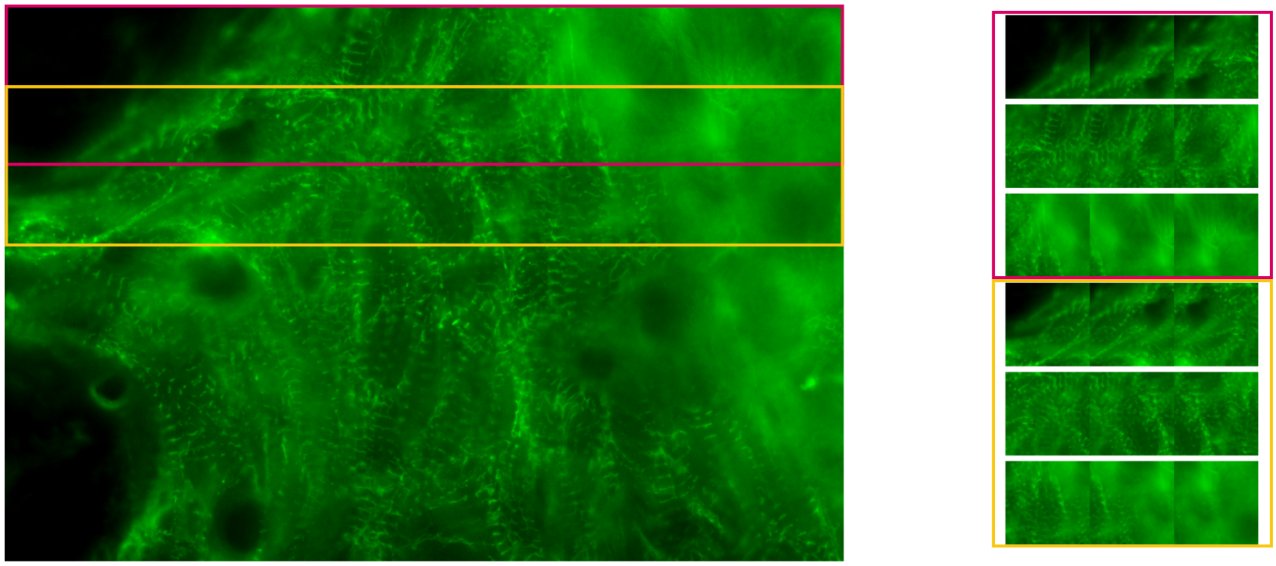


Figure 5: Example of patching an image of postnatal cardiomyocytes. The original image on the left (1388x918) is split into patches of 256x256, shown on the right, with a stride of 128. The row illustrated in red corresponds to 9 patches enclosed in red on the left, the same applies to the row enclosed in yellow.

2.1.5 Hyperparameters

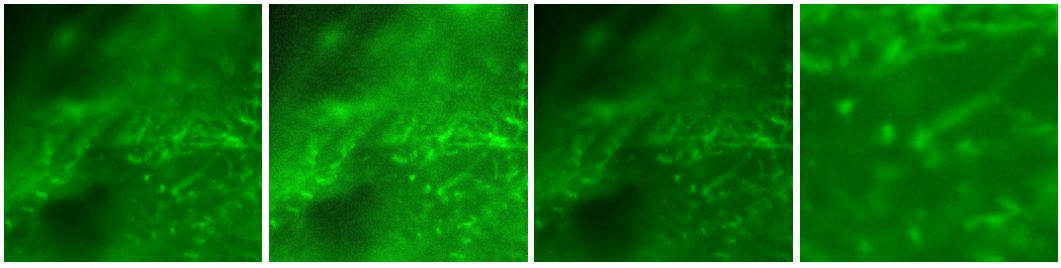
The performance of deep neural networks, such as U-Net, depends on a variety of hyperparameters that have to be optimized. Moreover, microscopic images also require some parameters that need to be decided on. Here hyperparameters included in the presented research will be described.
The first group of hyperparameters falls into the category of image pre-processing, operations applied to the image before they are fed into the neural network.

- **Patching** Microscopic images of cell cultures are usually of very high resolution. Due to memory constraints on common machines it is impossible to input an entire image into a neural network. Simply resizing the images is not an option, because a lot of valuable information would be lost in that way. Therefore images need to be divided into smaller patches, the size of these patches, *patch size*, is an important parameter. The process of patching is illustrated in Figure 5.

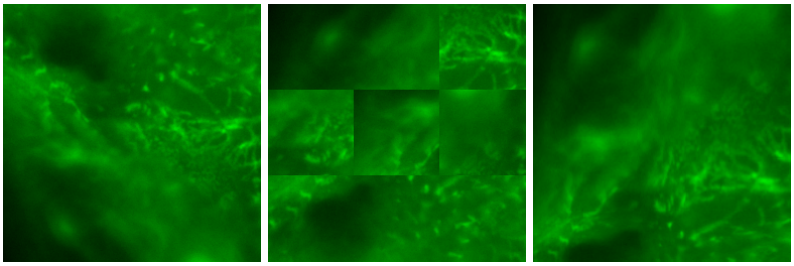
- **Class-based sampling** refers to the process of balancing class instances by simply duplicating them (over-sampling) or by removing instances (under-sampling). This is a common approach to combat class-imbalance, the minority class is over-sampled and the majority class is under-sampled. In segmentation tasks the class distribution can be calculated in two ways. Pixel-wise, for each class the number of pixels belonging to that class over the total amount of pixels or patch-wise, the number of patches containing a class over the total number of patches. The re-sampling is then done on the patch level, doubling patches containing minority classes and removing patches that only contain a majority class.

- **Batching** refers to splitting the training set into ”batches”, a set number of training instances used to calculate the network update for one training step. The *batch size* refers to how many images are included in a single batch.

- **Image augmentations** refer to operations applied to the images during the training process. This is done to induce diversity in the training set, by slightly changing images to create new ones. Image augmentations have been shown to significantly increase the performance of deep neural networks [23]. This is especially true for small datasets and when the augmentations are specifically chosen for a given dataset. Image augmentations can be divided into pixel level and spatial transforms. Some examples of pixel level transforms are brightness, Gaussian blurring and colour augmentations. For spatial transforms examples are flipping, cropping or grid distortions. Some of these transforms are illustrated in Figure 6.



(a) Original image (b) Brightness/Blur (c) Contrast (d) Cropping



(e) Flip (f) Grid shuffle (g) Grid distortion

Figure 6: Illustration of various image augmentations. For transforms that alter image shape or structure the same transform is applied to the masks.

The second group of hyperparameters directly influences the learning process of the neural network.

- **Learning rate** determines how much the weights of the network are updated at each training step. A large learning rate will lead to the network learning faster, at the potential cost of arriving at a sub-optimal solution.

- **Backbone** refers to the specific network architecture of each down and up sampling block in U-Net. Over the years a variety of different backbones for CNN's have been designed [24]. One of the most successful ones is ResNet, here so-called residual connections are included in the network [25]. Next to choosing a backbone architecture, the depth of these blocks can also be adopted. Depth here refers to the number of convolutional layers included in a block.

2.2 Transfer Learning

One major challenge in deep learning is the availability of annotated data. For semantic segmentation of cellular images a biomedical expert needs to invest a significant amount of time to annotate images. This often leads to relatively small datasets. A popular approach to this problem is to initialize the weights of a network by training them on another dataset. The idea behind this is that data and tasks

are often similar, so representaion learned for one task are also useful to perform another. In this way knowledge about one task can be transferred to a new task [26]. Formally transfer learning can be defined using the concepts of *domain* and *task*. A domain refers to a feature space X and a probability distribution $P(X)$ over this space. A task consists of a label space Y and a predictive function $P(y|x)$, where $x \in X$ and $y \in Y$. In transfer learning the task is then to learn a function $f_t(x)$ for a target domain and task (D_t, T_t) utilizing knowledge from function $f_s(x)$ learned on the source domain and task (D_s, T_s) [27].

One task where transfer learning is routinely used is image classification. Here the intuition is that many features occur in multiple objects, so detecting them is helpful to classify many objects. Such features range from very general features, such as edges, to more specific features, for instance eyes. The most common dataset for pre-training is the ImageNet dataset. It includes millions of annotated images of diverse real life objects [28]. Since this dataset is so large and diverse, a neural network that is trained on it learns a diverse set of feature representations. When the same network is then used for another classification task, say facial recognition, the hope is that many of the already learned features transfer to the new task. Pre-training neural networks on ImageNet has led to significant performance improvements on a variety of benchmarks [26]. Aside from performance improvements, transfer learning has also been shown to reduce the training times of neural networks [26, 29]. Therefore, in many computer vision tasks it is common practice to use pre-trained models.

Transfer learning has led to great improvements in a variety of areas, however, it is not clear how helpful transfer learning is when pre-training and training datasets are significantly different. This is especially relevant for different image domains. While ImageNet contains photographs scraped from the web, other datasets contain very different images. Biomedical images for instance are acquired in a different way than photographic images, e.g. MRI scans or retinal images, and can therefore be considered as a separate image domain. For biomedical images it is common practice to use large models pre-trained on ImageNet, but a recent analyses shows that they offer little benefit compared to non pre-trained simple, light-weight models [29].

The reason for this could be that the feature representations learned during pre-training are not relevant in another image domain. For example, feature representations about the textures of eyes are probably not helpful to identify cell nuclei. Instead of pre-training on large general dataset, such as ImageNet, another approach is to pre-train on a dataset in the same domain.

Karimi et al. conducted experiments on the effects of transfer learning for medical image segmentation [30]. They used a variety of 3D voxel datasets for organoid segmentation. The results show that while in general transfer learning leads to faster convergence times and often increases model performance, this is mediated by a variety of factors. One factor is the size and quality of the target dataset as smaller and lower quality datasets benefit significantly more from transfer learning. Another factor is the similarity of source and target datasets. Pre-training models on datasets of the same organ, so on a very similar dataset, outperforms pre-training on other organoid datasets. Finally in the case of large and high quality target datasets, transfer learning did not improve segmentation accuracy. The authors argue that this could be due to an optimal solution being reached already and the error only coming from annotation mistakes [30].

One observation that holds across a variety of studies is that transfer learning decreases model training times [26, 29, 30]. This effect is, however, more significant in classification tasks compared to segmentation tasks. The reason for this is not known. Nonetheless, it could be that, due to the aforementioned differences between segmentation and classification tasks, transferring pre-learned features is less applicable in segmentation tasks.

2.3 Automated Machine Learning

Automatic machine learning (AutoML) refers to the field in machine learning that deals with the automatic configuration of ML-pipelines for a given task. Usually this task is performed by human experts in a tedious trial-and-error process. AutoML aims to replace this process with automatic, data-driven algorithms, that optimize ML-pipelines for a user given metric. For tabular data, some AutoML systems have beaten human experts in ML competitions [3].
An overview of the components in an AutoML system can be found in Figure 7. The input to the system are a dataset and a task, the system then performs data pre-processing selection, algorithm selection and hyperparameter optimization. The output of the system is an ML-pipeline for the given task. This ML-pipeline is then constructed from some machine learning library and trained on the dataset. An important design choice of AutoML systems is the search space $\boldsymbol{A}$, that contains the different components of the ML-pipeline that can be combined. This space needs to be carefully designed and is crucial to the performance of the AutoML system [3]. Another part of some AutoML systems is the meta-database, containing ML-pipelines from the search space evaluated on other datasets. If such a meta-database exists, meta-learning can be used to leverage this previous experiences. Meta-learning is further discussed in section 2.3.3.
The combination of algorithm selection and hyperparameter optimization (HPO) in the search space is referred to as the combined algorithm selection and hyperparameter optimization problem (CASH) [31]. Data pre-processing is usually not considered as a part of an AutoML system. Nonetheless, especially for image data, pre-processing steps such as augmentations are vital to the performance of the system. Since data pre-processing does not change the algorithm, it can also be seen as part of HPO process.

2.3.1 Hyperparameter Optimization and Algorithm Selection

Hyperparameter optimization (HPO) refers to the process of selecting the best performing hyperparameters for an ML algorithm. Hyperparameters can exist as real values (e.g., learning rate), integer values (e.g., batch size), binary values (e.g., should a specific data augmentation be used), or categorical values (e.g., choice of optimizer). Tuning hyperparameters is important since it can often increase the performance of ML-pipelines, compared to leaving them at a default value [9]. Algorithm selection refers to what ML algorithm should be used for a specific task. It is possible to combine HPO and algorithm selection by including the algorithm as a categorical hyperparameter. This is known as the combined algorithm selection and hyperparameter optimization problem (CASH) [31].

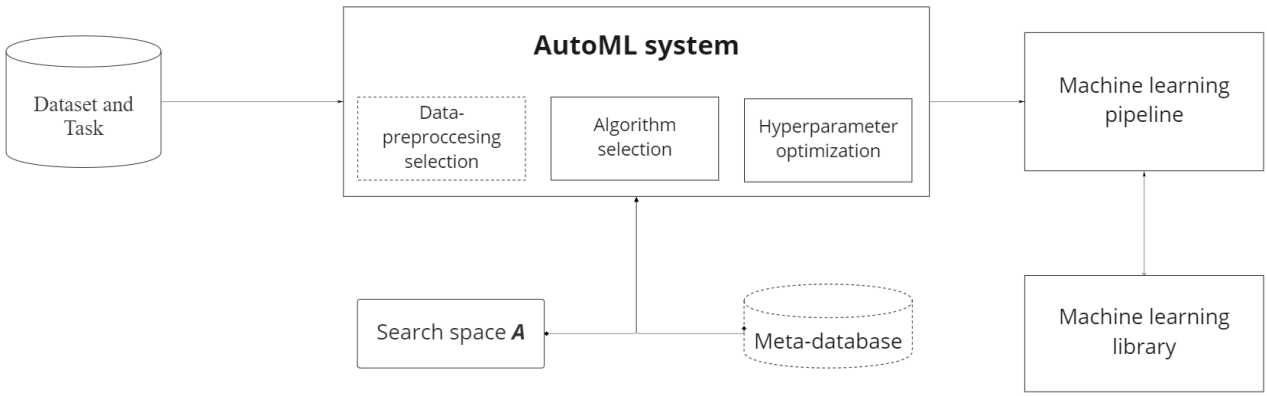


Figure 7: Overview of the components of an AutoML system

Optimizing hyperparameters is a combinatorial optimization problem, each combination of hyperparameters (HP's) is associated with a specific performance value. It is particularly challenging due to the diverse set of data types, as some HP's are integers while others are floats or categorical. There are even conditional HP's, e.g. HP's only applicable to a specific ML algorithm. This highlights the importance of defining a good search space.
Methods used in HPO include random search, grid search, population-based methods (such as genetic algorithm, evolutionary algorithms, and particle swarm optimization), gradient-based optimization, surrogate-based optimization and meta-learning [3, 32]. Gradient-based optimization uses information about the gradient of the loss function with respect to the HP's to optimize the HP's [33]. In surrogate optimization, a surrogate regression model is used to model the performance of HP configurations as a Gaussian process. Some optimization method, e.g. Monte Carlo approach, is then used to identify the HP configuration likely to perform best. One of the most popular tools for HPO is SMAC (sequential model-based algorithm configuration) which performs surrogate optimization. Here a random forest is used as a surrogate model and Bayesian optimization determines the next HP configuration to sample [34]. In this way HP configurations are sequentially evaluated for a predefined number of steps or until a stopping criteria is reached.

2.3.2 Neural Architecture Search

When optimizing neural networks, the network architecture can be included into the search space. This is called neural architecture search (NAS) and currently a hot topic of research. State-of-the-art neural networks often rely on complex architectures, these architectures are designed by human experts in a time-consuming and error prone process.
NAS consists of three main dimensions: search space design, search strategy and performance evaluations [3]. **Search space** design refers to defining a space of architectural components that will be searched for an optimal one. The design choices for neural architectures are vast, so the search space needs to be sufficiently restricted. A common approach to this is to incorporate expert knowledge, however, this might induce human bias and exclude viable novel architectures. **Search strategy** refers to how this space is then searched for an optimal architecture. Many techniques from HPO can also be applied here. **Performance evaluation** strategies describe how the performance of a candidate architectures are evaluated. The standard way this is done is by training the architecture on a training set and then evaluating it on a validation set. However, this is computationally expensive and it is infeasible to evaluate many candidates. Current research therefore focuses on techniques to approximate the performance of configurations without fully training the neural network. Some of these are further described in section 2.3.5.
NAS systems have achieved impressive performance on a variety of datasets [3, 35]. Nonetheless, there are often default architectures that have been shown to generally perform well on a variety of datasets. Therefore, in the presented study the focus is on optimizing other parameters of the DL-pipeline.

2.3.3 Meta-Learning

In AutoML, meta-learning or 'learning to learn', refers to using previous experience to learn how to perform a new task. The construction of a ML-pipeline is considered somewhat of an art without many guiding principles. Therefore, when human experts construct a ML-pipeline for a new task, they rely on their experience of using ML to solve similar tasks. Similarly, in the field of AutoML, meta-models leverage prior experience, in the form of meta-data, to learn how to solve new tasks.

This meta-data consists of tuples *P(i, j)* where *i* is some dataset and *j* is the pipeline configuration. For each tuple *P(i,j)* there are some evaluation scores such as accuracy and run-time, showing how well configuration *j* did on dataset *i*. There also information about the task in the meta-data, each tuple *P(i,j)* belongs to a specific task *t*. Then *(P(i,j,t)* gives information about the performance of configuration *j* on dataset *i* to perform task *t*. However, in many meta-learning based systems, such as the presented research, the task is always the same and therefore we can omit the task and the meta-base only consists of tuples *P(i,j)*. The goal in meta-learning is then to use a *meta-model* to predict the performance of tuples *P(**i**, j)*, where ***i*** is some new dataset [3]. It should be noted that task and dataset are often used interchangeably in the literature. Here, task is only used to describe a machine learning task such as regression, classification or detection.

A key question in meta-learning is how to represent the datasets so that they contain information relevant to the performance of configurations. To this end, features describing datasets need to be extracted. These are called meta-features. These meta-features can be *handcrafted* simple (e.g size of dataset, number of classes, number of features) , statistical (e.g class distribution, correlation, covariance), information-theoretic (e.g. information gain), model-based, and landmarkers [3]. Landmarkers are the performances of simple models, e.g. 1-NN or random trees, evaluated on a dataset. Studies on the Open-ML database [36], have shown that the optimal set of meta-features often depends on the application [37].

Another approach is to learn representations of datasets. The objective here is to learn representations that capture information relevant to quantize and assign similarity scores to datasets. This can been done by using meta-models to learn a function $f : M \rightarrow M'$ from existing meta-data P, here M is a set of handcrafted meta-features. The meta-features M' can then help to accurately predict performances for new tasks. One example of M' is a binary encoding of the pairwise performance of configurations, so for two given configuration which one is predicted to perform better on the new dataset [38]. Another way to learn M' is deep metric learning. As an example, if two datasets have the same feature dimensions (e.g. same size images), a Siamese neural network can be used. Here the data is inputed to two twin networks and the difference between predicted performance and observed performance is used as a loss function [39]. Since the weights between both networks are tied in a Siamese network, two very similar tasks are mapped to the same regions in the latent meta-feature space [3].

There are variety of ways to leverage meta-data to find an optimal configuration for a new dataset. One approach is to *warm-start* HPO with configurations known to perform well on similar datasets [40] . In order to do this some form of dataset similarity needs to be defined, this can simply be some distance measure between two dataset meta-feature vectors. In *warm-starting* the best configurations of the t most similar datasets are then first evaluated on the new dataset. In this way, the surrogate model is initialized and the optimization process is guided towards a promising area in the search space [40, 3]. Especially for larger datasets, and therefore longer evaluation times, warm-starting the optimization process leads to significant performance improvements. This approach is used in the first version of the auto-sklearn system, which has won many (early) AutoML challenges [41].

Another approach to leveraging the meta-data is to use a *meta-model* to construct a ranking of configurations that are likely to perform well on a new dataset. Here the goal of the *meta-model* is to learn a function $f : (M_i, C_j) \rightarrow R(i, j)$. M_i are the dataset meta-features, C_j the configuration parameters and $R(i, j)$ is the rank of configuration j on dataset i relative to all other configurations C. The meta-model learns this function by training on the described meta-base. This training is called the *offline* phase of a meta-learning system. In order to then find a good configuration for a new dataset, all configurations C_j in the search space can be concatenated with the dataset meta-features M_i and the meta-model outputs a ranking of the configurations. This prediction for a new dataset is called the *online* phase. An overview of this process is given in Figure 8.

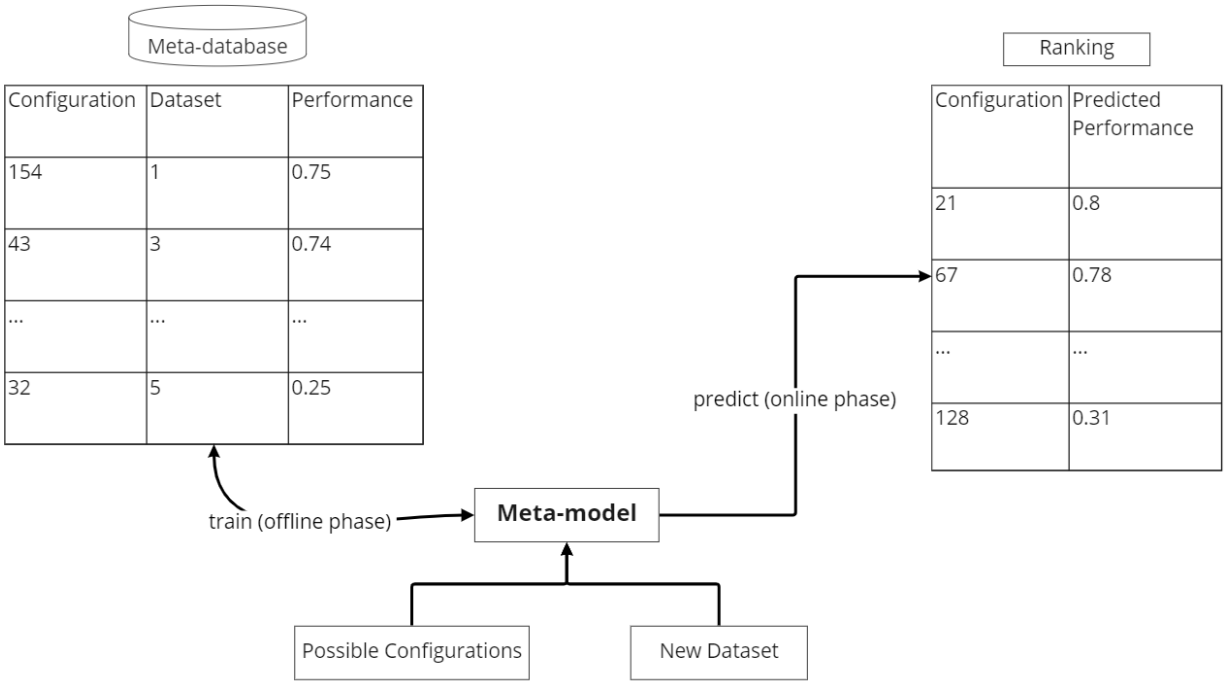


Figure 8: Overview of an AutoML meta-learning system

There are three different ways in which machine learning can be used to construct a ranking. In pointwise ranking, a regression model is used to predict the relevance score of an input. In pairwise ranking the problem is transformed into binary classification; for two given configuration predict the better one. Iterating over configurations in this way leads to final ranking. Finally, the listwise method optimizes for a ranking evaluation metric directly, in this way the order of all configurations is considered in the loss function [42].

2.3.4 Ranking AutoML Systems

The first system proposed to create a ranking of configurations for a new dataset was not based on a machine learning model. Nonetheless, it is often used as a baseline for other more sophisticated systems. In *average ranking* first some dataset similarity measure is used to find the k most similar datasets. Then all configurations are ranked based on their average performance on these k most similar datasets [43].

One more recent system constructs an optimal bagging workflow for a given new tabular dataset [44]. Among other parameters, the search space considered consists of the number of learners in the ensemble, the pruning method used or the dynamic integration method. Due to the structure of a bagging workflow this search space only results in 63 valid configurations per dataset. To construct a meta-base 140 classification dataset from OpenML [36] were used, on each dataset the performance metrics of the 63 configurations were obtained. This resulted in a meta-base with 8820 datapoints. Each dataset was represented by a vector of 158 meta-features, these were extracted using a tool for automatic meta-feature extraction [45]. Thereby, most classic *handcrafted* meta-features are included. The ranking was constructed using the pointwise approach, the meta-model (XGBoost) was trained to predict accuracy scores given a concatenation of dataset meta-features and the given configuration. This meta-model was evaluated in a leave-one-dataset out cross-validation scheme.

The results showed that this learning-to-rank approach significantly outperformed average ranking.

Moreover, their was no significant difference between the performance of the optimal workflow and the best one in the predicted top five.
Another recent ranking based AutoML system is RankML [46]. Here, the search space consists of various available components with scikit-learn [47], a classical machine learning library. These were presented acyclic graphs, in the order that machine learning pipelines are usually designed. First data pre-processing, then feature preprocessing, then feature engineering and finally predictive models. To construct the meta-base ML-pipelines were randomly constructed for each task, regression or classification, and dataset. In this way, 142.006 pipelines for classification and 171.482 pipelines for regression were evaluated. Overall 149 classification and 79 regression datasets were included in the meta-base. Due to the the representations of pipelines as acyclic graphs, the overall size of pipeline vectors is kept small while still representing the large search space. The ranking was constructed using the pairwise approach, the meta-model used was again XGBoost.
The resulting system achieved performances similar to then state-of-the-art AutoML systems on a variety of AutoML benchmarks, at a fraction of time. So RankML showed that it is possible to predict optimal ML-pipelines for a new task, given a sufficiently large meta-base. This is especially beneficial when the goal is to find an optimal configuration in a short amount of time.

2.3.5 Multi-Fidelity Methods

A key challenge in AutoML is that it is computationally expensive to evaluate an ML-pipeline on a dataset. This is especially true for deep learning models and large dimensional datasets, such as images. Training a single ML-pipeline on a single dataset can easily take several hours, and for some tasks even days [48].
A common approach that human experts use to combat this problem is to evaluate configurations on a subset of the data or some otherwise reduced version of it. Multi-fidelity methods refer to algorithms that formalize these manual heuristics, using so called low fidelities to approximate the actual performance of configurations. Examples of these low fidelities are subsets of the data, short training times, down-sampled images or using a subset of the available features. While there are a variety of multi-fidelity methods, in AutoML there are two predominant approaches to this problem. The first is to extrapolate learning curves to approximate final performance. The second focuses on bandit-based algorithms to to determine the best configuration out of a finite set.

In learning-curve based prediction for early stopping some number of configurations are trained for a specific time and a performance evaluation is recorded at every step. Based on these learning-curves a decision is then made which configurations are likely to perform well, these are then trained further [3]. This is similar to how deep learning practitioners evaluate models, experienced practitioners often decide whether to keep training a model based on the learning curve. In order to automate this process, a model needs to predict the final performance of configurations based on partially evaluated learning curves. Configurations that are predicted to perform significantly worse then the best predicted configuration are stopped.

Therefore, the model needs to give a probabilistic estimate of the the continuation of a learning curve. This has been done by modelling the learning curve as a weighted combination of parametric models from various scientific fields [49]. These parametric models were selected due to similar characteristics as learning curves, increasing rapidly in the beginning and then slowly converging to a horizontal line. So each learning curve is modelled by a function:

$$f_{comb}(t|\varepsilon) = \sum_{k=1}^{k=11} w_k f_k(t|\theta_k)$$

Where $\varepsilon = (w_1, ..., w_{11}, \theta_1, ..., \theta_{11}, \sigma^2)$ represents the combined parameter vector, w is the weight of function $f(t|\theta)$ with parameters θ and σ^2 is the noise variance. These weights and parameters are then sampled via Markov chain Monte Carlo to minimize the loss of fitting the partially observed learning curve. This yields a predictive distribution, which allows to stop training configurations based on their probability of outperforming the best predicted model [3, 49].

This predictive early stopping method can be combined with SMAC, by keeping track of the current best configuration and stopping training if it is unlikely that a configuration outperforms it. On average including early stopping decreases the time needed for SMAC to find an optimal configuring by a factor of two [49], for the MNIST [50] and CIFAR-10 [51] datasets.

A limitation of the method presented above is that information is not shared across different configurations. The combination of parametric models is only fitted to one learning curve and there is no way to incorporate knowledge about the learning curves of other configurations. This can be achieved by using a Bayesian neural network to predict the parameters and weights of the parametric function combination [52]. In this way an informative prior, knowledge about the shape of learning curves, is incorporated into the Bayesian neural network and information is shared across different configurations.

In the multi-armed bandit problem a limited amount of resources must be allocated between competing choices to maximize the overall expected gain. The properties of each choice are not fully known initially, but they might become better understood as more resources are allocated towards them [53, 54]. Selecting the best ML-pipeline out of a finite set for a given task within a given compute budget is such a multi-armed bandit problem. At first it is unknown how well each configuration will perform, but, as the configurations are evaluated on larger fidelity's of the dataset, more is known about the performance of individual configurations.

One popular bandit based multi-fidelity method is successive halving. The idea behind it is quite simple, but nonetheless has been shown to perform well and can also be combined with other methods. While in learning rate extrapolation only shorter run-times are used as fidelity's, in successive halving the budget can be anything.

In successive halving all configurations are first evaluated on some initial evaluation budget, then the worst performing half is dropped and the evaluation budget is doubled. This procedure is then repeated until only one configuration is left [55, 3]. This is illustrated in Figure 9. One downside of successive halving is that is suffers from the budget vs number of configurations trade-off. For a given total budget the user has to decide whether to try many configurations and only allocate little budget to each configuration, or to include fewer configurations and allocate more budget to each. When to little budget is allocated per configuration good configurations might be dropped, and when to much budget is allocated resources are wasted to evaluate bad configurations.

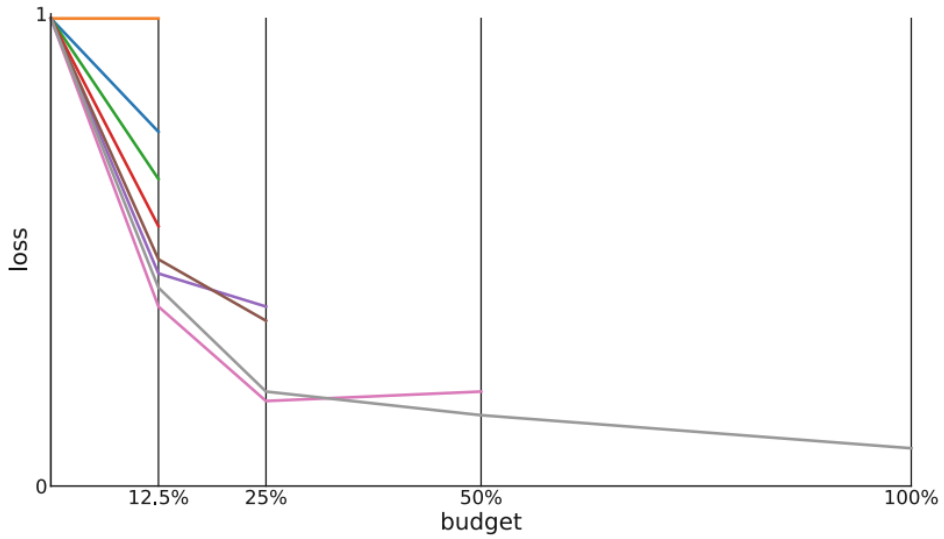


Figure 9: Illustration of the successive halving procedure. Image taken from [3].

2.4 Quantizing Optical Appearance

As explained in the previous section, designing suitable meta-features to represent datasets is crucial for meta-learning based systems. For AutoML systems for images, information about how the images ”look like” could also be important to find well working configurations.
In medical image analyses, radiomics are data-characterization algorithms aiming to extract a standardized set of features from images. These features fall into five different classes: First-order statistics, shape descriptors and textural descriptors. Textural descriptors is then further divided into gray level co-occurrence matrix, gray level run length matrix, and gray level size zone matrix [56]. Radiomics feature extraction was developed to convert images into meaningful vectors that can then be used for analyses. This quantization of images has shown great promises in clinical-decision support systems to improve diagnostic, prognostic, and predictive accuracy [57]. In general, radiomics feature extraction has shown promises in classifying cancer in a variety of image modalities [58, 59, 57].
A typical radiomics workflow consists of image segmentation, preprocessing and feature extraction. This workflow is often designed with a specific application in mind so that only application relevant feature are extracted. Therefore, a radiomics feature extraction scheme cannot be used across applications and has to be redesigned [60]. AutoML has been used to automate this process, aiming to streamline radiomics research, facilitate radiomics reproducibility, and simplify its application [61].
In general, radiomics have shown promise in medical image analyses and it can therefore be assumed that they are a good tool to quantify images into meaningful vectors.

3 Related Work

This section provides an overview over AutoML systems for image data. To this end, first existing AutoML systems and their performances are presented. Secondly, some technologies that are potentially used in these systems are discussed. Thirdly, systems that automate the development of DL-pipelines are discussed. Finally some existing systems for cellular image analyses are referenced and the motivation behind the presented research is given.

3.1 AutoML Systems for Images

There are many AutoML systems developed for tabular data. For this reason, a variety of techniques have been developed to find optimal ML-pipelines on tabular datasets. An AutoML system for images is in many ways similar to systems for tabular data, nonetheless there are some differences. Mainly due to the high dimensionality of images, it takes a long time to evaluate pipeline configurations. Additionally, images always have a similar structure, so techniques such as transfer learning and few-shot algorithms are potentially more applicable.
While there is little published research on AutoML for image data, there are some publicly available systems. Google AutoML[3], H20 hydrogen torch [4], Microsoft Azure AutoML [5], Amazon Sagemaker [6] and BigML's OptiML [7] all offer automatic configuration of DL-models for image analyses. The user therefore does not need to configure a model themselves, but can just upload their dataset, pay a fee, and get a trained model in return. Since these companies are competing for the best AutoML image analyses tool, there are no published papers describing how exactly these systems configure an optimal deep learning model. For instance, researchers at Amazon have published a paper describing AutoGluon, their tabular AutoML tool [62]. They state that their system also works for image data, but that the underlying technology for this task is significantly different.
While there is not much information about the underlying technologies, the performance of these systems can still be compared. A good overview of the performance of state-of-the-art neural networks with default hyperparameters (ResNet [25]), open source (auto-sklearn [41] and AutoKeras [63]) and commercial (Google AutoML Vision) AutoML tools was provided by Yang et al. [64]. They released a new benchmark dataset, MedMNIST, consisting of 10 different preprocessed medical image datasets. These cover all key data modalities in medical images and the tasks include binary/multi-class, multi-label classification and ordinal regression. Moreover, the size of dataset ranges from 100 to 100.000 images. Overall, the results indicate no clear winner, no system achieved superior performance on all datasets. Nonetheless, Google AutoML vision performed best overall followed by AutoKeras and default ResNets. Auto-sklearn performed worst, which is not surprising since it relies on classical machine learning and modern CNN architectures are not included in the search space.

3.2 Generalist Deep Learning Models

As stated, for many AutoML systems for image data no published research describing the underlying technology is given. Still, there are some technologies that are potentially used in these systems. Here,

[3] https://cloud.google.com/automl
[4] https://h2o.ai/blog/comprehensive-guide-to-image-classification-using-h2o-hydrogen-torch/
[5] https://azure.microsoft.com/en-us/services/machine-learning/automatedml
[6] https://cloud.google.com/automl
[7] https://bigml.com/whatsnew/optiml

two techniques that aim at making DL models more general, so applicable to multiple datasets/tasks are discussed.

3.2.1 Few shot learning

Few shot learning is a field related to the presented research. Here, the aim is to enable networks to learn new task with only a few data-points of that task. For example, the new task could be to detect cats in an image. The model should then learn this from a few annotated images of cats. This is a hard problem, because, usually neural networks require many examples per class. In Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks (MAML) [65] this is done using a specific training technique. First, some tasks are sampled, then for each task the gradient step w.r.t. k examples of the task is calculated. The update of the overall network is then the average of the gradient steps over the sampled tasks. In this way, it is possible to learn a new task with a few examples since the gradient step is also calculated w.r.t. other tasks. So the model leverages information from other tasks when learning a new task. This is only a high level description of this technique, for a more detailed description please see the original paper [65].
The connection between few-shot learning and the presented research is that both aim at leveraging previous tasks to learn a new task. Different to the presented system, few shot learning focuses on directly learning a new task, while in the presented research the focus is on *how* to best learn a new task. For the presented research, a few shot meta-learning approach could be to treat each dataset as a task and to then training a agnostic model using the described training procedure. In this way, the model could be quickly adopted for a new dataset. However, this would require a large variety of datasets and there is less common information across datasets than there is across different classes in image recognition. Nonetheless, few shot meta-learning could potentially enable quick adaptations of neural networks for new datasets.
While few shot meta-learning focuses on fast adaptions of neural networks, there are also models that are trained on diverse datasets and work on new datasets without re-training. These models are able to perform well on variety of datasets in a specific domain.

3.2.2 Cellpose

Cellpose is a generalist deep learning model for instance segmentation of cellular images [66]. The model was trained on a large dataset composed of cellular images from a variety of microscopy modalities and fluorescent markers. Overall, the dataset contains over 70.000 segmented objects and has been designed to contain all kinds of object shapes. Due to this training on a variety dataset, the model does not require re-training and can be used directly on novel images. The network architecture is similar to U-Net [20] with a backbone based on ResNet [25]. Running the system on the CellImageLibrary dataset [67] showed that it significantly outperforms StarDist [68] and Mask-RCNN [69], two other state-of-the-art generalist segmentation models.
The main attraction of Cellpose is that it works well on a variety of cellular images and does not require retraining. This means that biomedical experts do not need to go through the laborious process of annotating a new dataset. While this model is an amazing tool and is currently being used in practice, there are two limitations. First, the model only performs instance segmentation and cannot be used for semantic segmentation. Secondly, it will be outperformed by models trained on large specific datasets. The more different the dataset, the larger the performance difference will be.
This section presented a technique for quick adaptations of neural networks and a system relying on transfer learning. These are related to AutoML systems for images, since both offer solutions to the

problems of long training times and re-usability of similar data. The approach of Cellpose can be seen as an alternative to the approach presented in this thesis, using a generalist model instead of optimizing a specific pipeline.

3.3 Automating Deep Learning Pipelines

There are also systems that focus on finding an optimal DL-pipeline for a new dataset. This optimal DL-pipeline includes the neural architecture and other hyperparameters, such as the learning rate. Most of these systems focus on tabular data, but there are some that also work on images. Here two systems automating the two most commonly used deep learning libraries (Tensorflow and Pytorch) are briefly presented.

3.3.1 Auto-Pytorch

Auto-Pytorch jointly optimizes the network architecture and the training hyperparameters to enable fully automated deep learning [70]. It has been developed for tabular data, nonetheless its optimization approach has also been used on the NAS-Bench-201 search space [71] and has achieved good performance on benchmark image classification datasets. The authors introduced the BOHB optimization algorithm, combining Bayesian optimization and hyperband. Furthermore, they warm-start the Bayesian optimization using meta-learning. Meta-learning is done by first evaluating some configurations, that performed well on other datasets, on the new dataset. This is similar to the approach used in auto-sklearn, described in section 2.3.3.
Overall, their approach relies on an efficient selection of both what configurations to evaluate and on what budget they should be evaluated. By combing these two, a guided and highly efficient search over configurations is achieved. Nonetheless, this approach still relies on training many configurations from scratch on a new dataset. This leads to a long overall search process, which is not desirable for high dimensional image datasets.

3.3.2 Auto-Keras

This AutoML tool is specifically developed to perform NAS, it searches for the best neural architecture given a dataset and a task. It combines Bayesian optimization with a novel method called neural morphism [63]. This is the process of modifying a given neural network architecture by applying discrete operations such as inserting a layer, adding a skip connection between two layers, etc. What morphism operation to apply and evaluate next is determined by Bayesian optimization. The benefit of this network morphism is that previously trained weights can be reused, reducing the evaluation time for different configurations. Moreover, the functionality of the network is preserved throughout the optimization process.
Auto-Keras achieves state-of-the-art performance on common image datasets, such as MNIST [50] and CIFAR-10 [51]. While Auto-Keras presents and interesting method for NAS, it is not immediately usable for biomedical images since these require preprocessing. Moreover, as stated above the presented research does not include NAS and focuses on other hyperparameters.

3.4 Automatic Cellular Image Analyses

Next to the already presented Cellpose model, a variety of other tools for the analyses of cellular image have been developed. Italisk [7] contains pre-defined workflows for image segmentation, object

classification, counting and tracking. With its impressive user interface and good performance of the default workflows it is used by many researchers. Nevertheless, Italisk does not leverage AutoML to tune hyperparameters or perform feature selection. So for optimal performance the user must still configure the workflow manually. CellProfiler [8] is an image analysis software designed for high throughput screening. CellProfiler is able to automatically extract cellular features in images, such as cell size, shape, pixel intensity, texture, and colocalisation. Furthermore, CellProfiler offers functionality to interactively explore and analyse the multidimensional data extracted from the images using plots such as histogram, scatter plot, density plot and parallel coordinate plot. The data visualisation feature allows biomedical experts to intuitively and quickly draw hypotheses or insights regarding their cell culture. CellProfiler does not offer any ML or DL models for segmentation, but researchers can add their own implementations due to the software being open-source.
There are also products that are only available to paying costumers, one of these is Olympus Live Cell imaging [8]. In this system, the user can choose between three different pre-configured workflows for segmentation. All of these are based on U-Net, but they are specifically designing for different types of tasks. These tasks are loosely described based on characteristics such as object shapes or signal-to-noise ratios. The user can then choose an option based on their use-case.
In conclusion, there are variety of tools for cellular image analyses available to researchers. These offer great value, since they allow researchers to easily explore their data and perform some analyses task automatically. However, due to the workflows performing these analyses tasks not being task-specifically optimized, they might not perform optimally. To the best of our knowledge, there are no systems that incorporate AutoML to determine an optimal DL-pipeline for an analyses task.

[8] https://www.olympus-lifescience.com/en/solutions/live-cell-imaging/

4 Methods

In this section, all components of the AutoML system are presented. Each component is described and experiments to optimize the components are presented. First, the requirements to the system and the approach used are discussed. Second, the transfer learning methodology is presented. Third, the search space for pipeline configurations is given. Fourth, the default settings and a method to automatically configure class-based sampling are discussed. Fifth, all datasets in the meta-base and the construction of the meta-base are outlined. Sixth, the different meta-feature extraction techniques are discussed. Seventh, meta-learning metrics, models and experiments are described. Eighth, successive halving is briefly presented. Finally, the evaluation methodology of the best combination of all components, the final AutoML system, is presented

4.1 System Requirements and Approach

As described in the Introduction, the AutoML system has to find the best DL-pipeline configuration that learns how to segment the given ROIs in the images. Moreover, due to compute constraints this has to be done within a reasonable time limit. The input to the system is a dataset consisting of microscopic cellular images and their corresponding masks. The challenge for the system is that training a single pipeline takes a considerable amount of time and that the search space for good pipeline configurations is vast. So the system has to find a good pipeline configuration without evaluating many configurations on the input dataset. The AIxCell AutoML system tackles this task in a five-step process. First, a search space for DL-pipelines is designed, this is done based on previous experience and the search space is biased towards good configurations. Next to defining the search space, other hyperparameters are set to good default values. Secondly, random pipeline configurations are sampled from the search space and evaluated on different datasets. This then forms the Metabase, previous experience that the system leverages to find optimal configurations for new datasets. Thirdly, meta-features are extracted from the datasets and different feature engineering strategies are evaluated. Fourthly, meta-learning reduces the search space to a few configurations that are predicted to perform well. Finally, successive halving is used to find the optimal pipeline based on the meta-learning selection. This process of condensing the search space to a single DL-pipeline is illustrated in Figure 10. Prior to developing the AutoML system, transfer learning is investigated. Here the focus is on if transfer learning can reduce model evaluation times, which could improve the overall system. More specifically the transfer learning experiments explored the effects of domain-specific and generic pretraining.

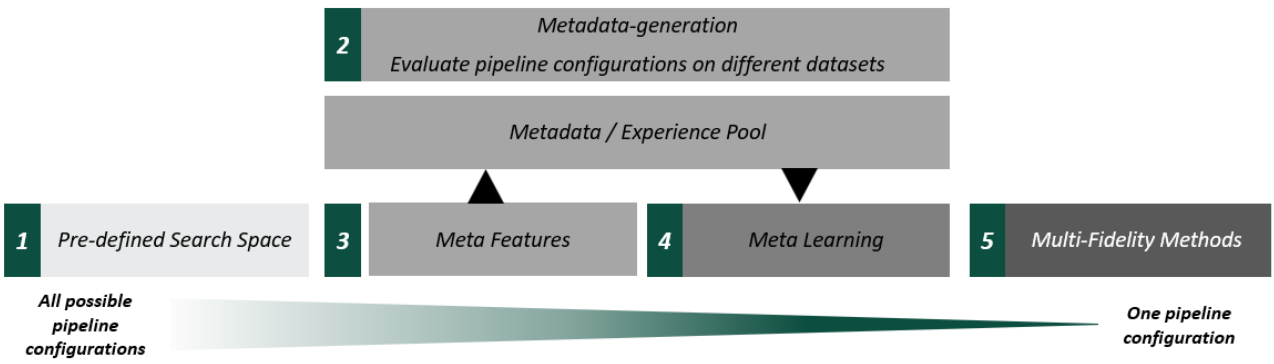


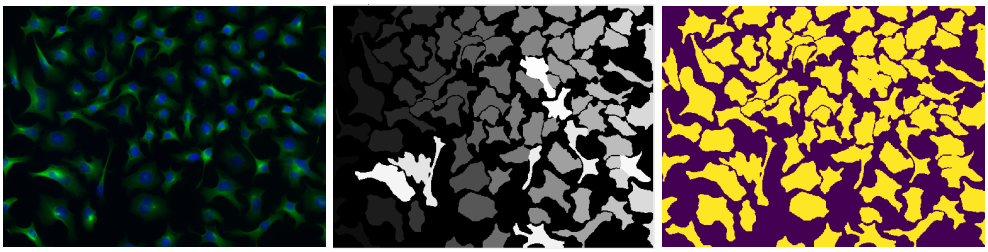
Figure 10: The five-step process used to find an optimal DL-pipeline from unlimited search space, within a given compute limit.

4.2 Transfer Learning

To compare domain-specific and generic transfer learning for semantic segmentation of cellular images, the effects of pre-training on two cellular image datasets and ImageNet are evaluated. Here the datasets and transfer learning methodology and experimental setup will be described. The same model is used in all conditions and is also used in the overall AutoML system. It is described in section 4.4 for the transfer learning experiments. ResNet-18 is used as a backbone.

4.2.1 Pre-training Datasets

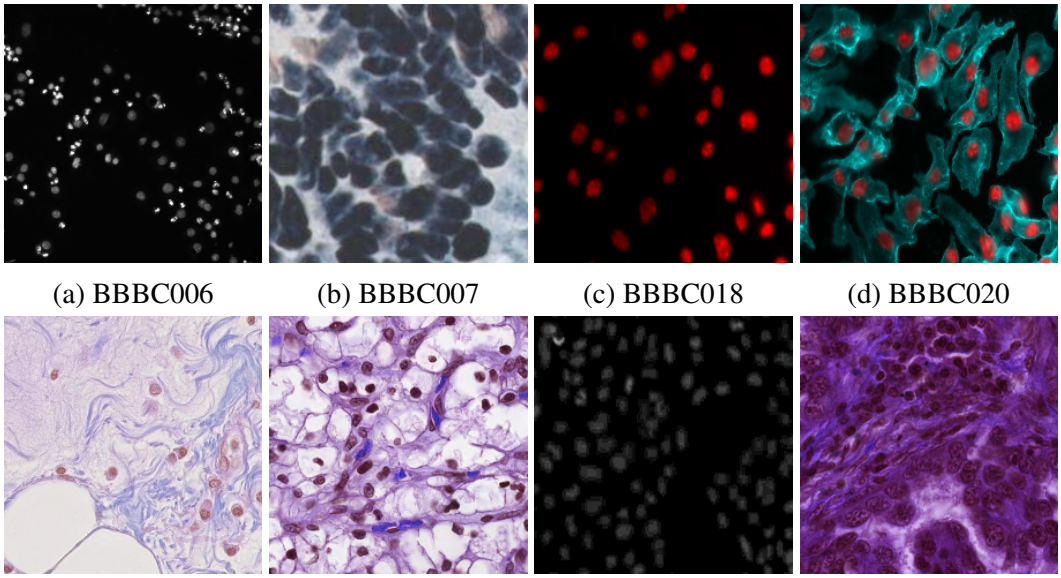
Two different domain-specific datasets and one generic dataset are used to pre-train the model. The first domain-specific one is curated to train the Cellpose model. It contains 608 images representing a variety of microscopy modalities and fluorescent markers. Additionally, it contains photographic images of various objects, such as sea shells or bananas. Overall, it contains a large variety of segmented objects. The cellpose dataset [66] was developed for instance segmentation. In order to use it for semantic segmentation, all segmented objects were assigned the same class and the masks were thereby binarized. An example of this conversion can be seen in Figure 11. This works well for most image mask pairs in the cellpose dataset. However, for images that contain many objects and not much background, it can be problematic since a large percentage of the image then corresponds to the object class. This can potentially be difficult to learn for the model, due to only minor changes in the loss function for predicting the entire image as class one or accurately segmenting the objects.



(a) Original image (b) Instance mask (c) Binary mask

Figure 11: The process of converting the instance segmentation masks into binary masks for semantic segmentation.

The second dataset used for pre-training is a collection of various datasets for nuclei segmentation. These images were collected using different microscopy modalities and thereby cover a large part of the cellular image domain.
The combined dataset is made up of a large part of the 2018 Data Science Bowl from kaggle (1276 images) [72], nuclei in histopathology images [73] (50 images), nuclei segmentation challenge (120 images) [74], nuclei in immunohistochemistry (50 images) [75], BBBC006 (32 images), BBBC007 (32 images), BBBC0018 (168 images), BBBC0020 (25 images) all from the Broad Bank Institute [76] and nuclei in U2OS cells (97 images) [77]. Some example images from this dataset can be seen in Figure 12. In total the dataset contains 1850 images with binary masks, representing the segmented nuclei. Since most images in this dataset come from the kaggle data science bowl, this dataset will be refered to as the kaggle dataset.
ImageNet [28] is used as the generic dataset for pre-training. Specifically pre-trained ResNet blocks were used for the encoder part of the network. The decoder part of the network is not pre-trained in the generic condition.



(a) BBBC006 (b) BBBC007 (c) BBBC018 (d) BBBC020

(e) Tissue Nuclei (f) Nuclei Benchmark (g) Kaggle Challenge (h) Histopathology

Figure 12: Example images for each dataset included in the *kaggle* dataset

4.2.2 Transfer Learning Procedure

To initialize the weights of the model, it was pre-trained for 50 epochs on both datasets. In this way, there are two models of identical structure, each one with its weights trained on a different domain-specific dataset. The learning rate was set to 0.0005, the adam optimizer and binary-cross-entropy loss were used.

The datasets that are used to evaluate the effects of pre-training sometimes required different input and output shapes. Therefore, the first and/or last layer have to be exchanged and its weights are randomly initialized. To preserve the pre-trained weights, all other layers are frozen and only the new layers are trained until convergence. The new layers were assumed to have reached a converged state when the validation IoU score did not change by more then 0.03 for three consecutive epochs. After this initial training all layers were unfrozen and trained on the new dataset, using a learning rate of 0.0001. The learning rate is set at such a low value to avoid ”destruction” of the pre-trained weights.

A similar procedure is used for the model with weights pre-trained on the ImageNet dataset. First, the pre-trained layers are frozen and all other layers were trained until convergence. Then, all layers are unfrozen and the model is fine tuned on the new dataset.

4.2.3 Experiments

Four differently initialized models were compared: pre-trained on cellpose, pre-trained on kaggle, encoder weights pre-trained on ImageNet and a non pre-trained model. The models were evaluated on four of the datasets within AIxCell: RUB, UKA, UKK and UKB. A description of these datasets is given in section 4.5. Images in these datasets are of large dimension, therefore they were divided into patches of size 224x224. The images used for evaluation were of the same dimensions as the images used for pre-training. The evaluation datasets were split into training, and test set using a split of [0.7, 0.3]. Performance of the models is evaluated using the validation set IoU score, the performance metric is logged every epoch.

Patch size	64	128	256	
Batch size	4	8	16	32
Learning rate	0.0005	0.001	0.01	
Backbone	ResNet-18	ResNet-50		
Augmentation: Flip	0	0.5		
Augmentation: Brightness and Gaussian Blur	0	0.5		
Augmentation: Grid distortion	0	0.5		

Table 1: The Search space considered in AIxCell. For the augmentations the value refers to the probability of applying an augmentation during the training process. Overall, there are 576 possible configurations.

4.3 Search Space

As stated in the theoretical foundations section, deciding on what hyperparameters to optimize is crucial to the performance of AutoML systems. For AIxCell we based that decision on the experience gained while specific DL-pipelines were developed for the use cases. Hyperparameters that were found to cause large variability of performance metrics were: patch size, batch size, learning rate, image augmentations, and the backbone used by the network. A description of these hyperparameters was given in section 2.1.5. Another factor that was found to have a large impact on performance, is the neural architecture used. However, on all use-cases, U-Net showed good performance and on three out of four it indeed performed best. Therefore, it was decided to only use the U-Net architecture and thereby restrict the problem to HPO and to exclude NAS.

Next to deciding on what hyperparameters to optimize, the possible values of the chosen hyperparameters need to be defined. Technically, even though there are only five hyperparameters, without further restriction the search space would be infinite. This restriction needs to be done in a way that includes optimal configurations, while also inducing enough variance into the search space. The possible values were again chosen based on the previous student project, see Table 1 for all possible values. Overall, the search space contains 576 possible combinations. Image augmentations were randomly applied during the training process, so before an image was fed to the model there was a chance of an augmentation to be applied. For each augmentation setting that probability to zero, so never applying that augmentation, and setting that probability to 0.5, so applying the augmentation 50 percent of the time, are included in the search space. Image augmentation can also be combined in that way, in the most extreme case all three can be applied to the same image.

Overall 576 possible configurations are included in the search space. By choosing potentially important hyperparameters and overall good values for them the search space is biased towards good solutions. In this space of well-performing configurations, the task of the AIxCell AutoML system is to find the optimal one.

4.4 Default Settings

While some hyperparameters are being optimized in AIxCell, others are set at default values. In this section, the chosen default parameters are presented. Moreover, a simple rule-based method to configure class-based (re-)sampling is discussed.

4.4.1 Default Parameters

A U-Net like model is used, see section 2.1.4. The model is implemented in Keras and is based on the segmentation models library [9]. In the encoder part of the network, there are five down-sampling layers that each contain four convolutional blocks. The convolutional blocks consist of a convolution layer followed by batch normalization and that output is then passed through a ReLU activation function. Convolutional layers contain residual connections. Each convolutional block is a ResNet block and contains either 18 or 50 layers. The decoder part of the network also has five convolutional blocks that follow the same structure as the encoder blocks. The activation function of the final layer is sigmoid for binary segmentation and softmax for multi-class problems. If ResNet-18 is used as the model backbone there are a total of 14.330.789 trainable parameters, for ResNet-50 there are 32.513.701 trainable parameters.
Other default parameters are: The patch stride was set to half of the patch size, adam optimizer on default settings, and categorical focal loss [78] (adopted for multi-class according to [79]) with a gamma of 2.

4.4.2 Automatic class-based sampling

In the first stage of the AIxCell project, the configuration of class-based sampling turned out to be crucial for optimal performance on some imbalanced datasets. In the presented system first the class distribution is calculated pixel-wise, the re-sampling is then done patch-wise. As explained above, the class distribution is calculated by dividing the number of pixels belonging to a class by the total number of pixels. Therefore, the class distribution is represented by an array with as many elements as there are classes. In the automatic class based sampling method it is first checked whether a dataset is indeed imbalanced. To determine if a dataset is imbalanced first a threshold is calculated:

$$threshold = \frac{\sum_{i=1}^{i=n} x_i}{n} + \frac{\sum_{i=1}^{i=n} x_i}{n} * 0.75$$

Here x is the array representing the class distribution and n is the number of classes. If any of the classes occur on a larger percentage of pixels than this threshold, the dataset is deemed imbalanced. As an example, consider a dataset with two classes and a class distribution of [0.8, 0.2] 80% of the pixels belong to class one, and 20% belong to class two. This dataset would not be considered imbalanced since the threshold ($\frac{1}{2} + \frac{1}{2} * 0.75 = 0.875$) is smaller than the largest class distribution value (0.8). Therefore, it can be said that this is a fairly conservative measure to calculate class imbalance. Nonetheless, one needs to consider that we are calculating class imbalance based on pixel-wise occurrence and in this setting only more extreme imbalances are problematic.
If a dataset is imbalanced the class-based sampling factors are calculated in the following way. First an acceptable low and high value of class occurrence are calculated: $a_{low} = \mu(x) - \mu(x) * 0.5$ and $a_{high} = \mu(x) + \mu(x) * 0.2$. If a class occurrence percentage (x[i]) is below or above this threshold a re-sampling factor is calculated: $factor = a_{low}\ or\ a_{high}/x[i]$. The factor is capped at 2 for up-sampling, but not capped for down-sampling. This is done to limit the number of duplicated images in the dataset.
The re-sampling based on the calculated factors is then done patch-wise, first, it determined which classes occur in each patch. Then, patches are sampled based on the calculated sampling factor. To illustrate the entire class-based sampling progress consider a three-class problem, the pixel-wise distribution is [0.1,0.1,0.7]. Since 0.7 is above the calculated threshold of 0.525, the dataset is deemed

[9] https://github.com/qubvel/segmentation_models

imbalanced. The class-based sampling factors are calculated as [1.5, 1.5, 0.514]. Now, after the AutoML system determined the optimal patch size, the images are divided into patches. Then the class distribution on a patch level is calculated. Say that classes one and two are more in the center of the images and in absolute numbers the patches containing pixels of the classes is [7, 10, 34]. So only 7 out of 50 patches contain the first class. Now after class-based sampling the absolute distribution could be [10, 15, 22]. Note that classes can co-occur on some patches and the new absolute distribution therefore can be different than the exact multiplication of sampling factors and initial absolute values. In the given example the last class co-occurs with the other classes on some patches and is therefore sometimes up-sampled and sometimes down-sampled.
It would be better to do both the initial calculation and the re-sampling on a patch level. However, since patch size is an optimizable hyperparameter and the process of dividing a dataset into patches is computationally expensive, the patch-wise distribution is approximated using the pixel-wise distribution.

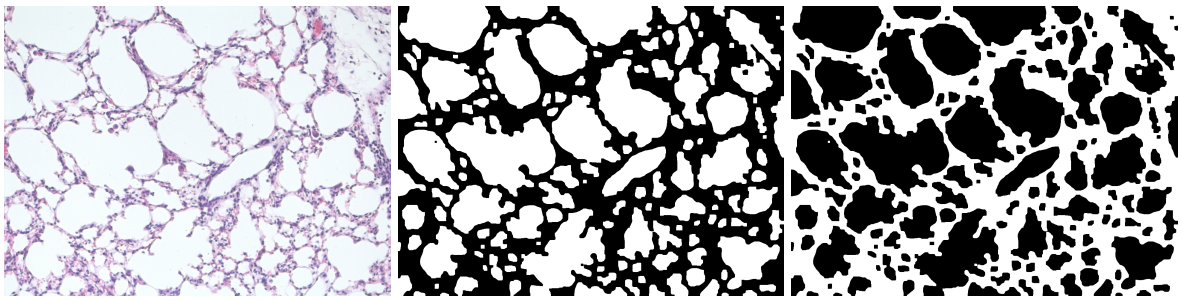
4.5 Meta-Base

The core of the AIxCell AutoML system is meta-learning, here the system leverages data of the performances of configurations on other datasets to predict an optimal pipeline for a new dataset. To this end, a meta-base was constructed. This meta-base consists of pipeline configurations, sampled without replacement from the search space, trained on eight different datasets. The entry into the meta-base is then the pipeline configuration, the dataset, and the validation set IoU score. In this way, an experience pool is constructed for the meta-model to learn from.

4.5.1 Datasets

Six of the used datasets were part of the AIxCell project, four were provided by the project partners, and two were annotated with the Fraunhofer IPT. Additionally, two public datasets were also used to create the meta-base. In this section, all datasets are described. To this end, the datasets meta-features (further discussed in section 4.6) and example images are given. For the all example images white represents the class and black the background.

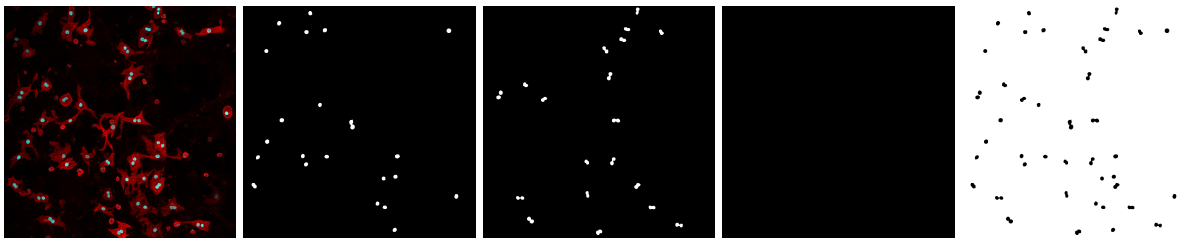
RUB contains 72 RGB images with mean image dimensions 968x1292 of lunge tissue obtained using brightfield microscopy. The object of interest is tissue and the magnification factor is unknown. There are two classes, one being the inverse of the other. The other handcrafted meta-features are: average mean area: 8440.43, average std area: 25047.10, average min area: 2.0, average max area: 372536.0. An example image is shown in Figure 13.



(a) Image (b) First Class (c) Second Class

Figure 13: Example image from the RUB dataset.

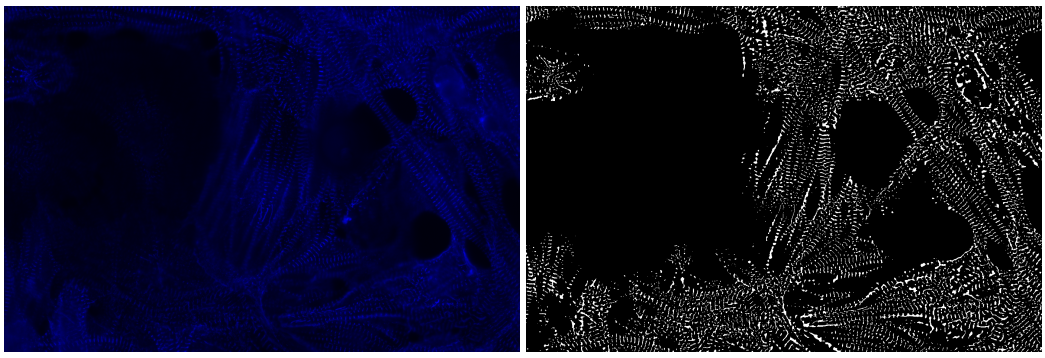
UKB contains 32 images with mean image dimensions 1024x1024 of cardiomyocytes obtained using fluorescent microscopy. The object of interest is nuclei and the magnifcation factor is 20. There are four classes, one of the classes only occurs in some of the images. The area-based handcrafted meta-features are: average mean area: 7140.11, average std area: 41909.09, average min area: 105.0, average max area: 262031.37. An example is shown in Figure 14, note that the third class only occurs on a few images and is not present in this example image.



(a) Image (b) First Class (c) Second Class (d) Third Class (e) Fourth Class

Figure 14: Example image from the UKB dataset.

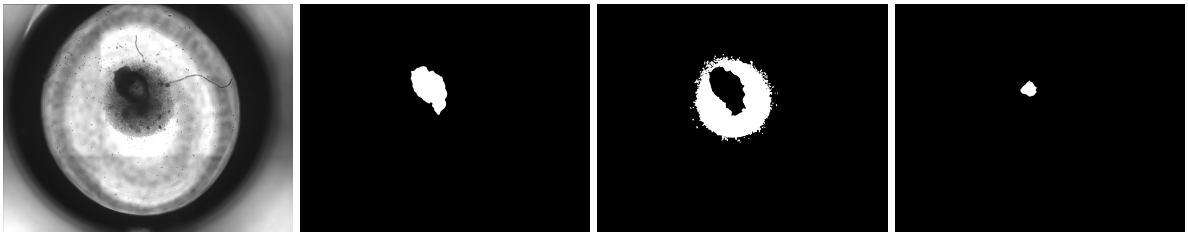
UKK contains 229 images with mean image dimensions of 957x1414 of sarcomeres obtained using fluorescent microscopy, there is only one class. The object of interest is sarcomeres and the magnification factor is unknown. The other handcrafted meta-features are: average mean area: 10.81, average std area: 34.60, average min area: 0.0, average max area: 6994.0. An example image is shown in Figure 15.



(a) Image (b) First Class

Figure 15: Example image from the UKK dataset.

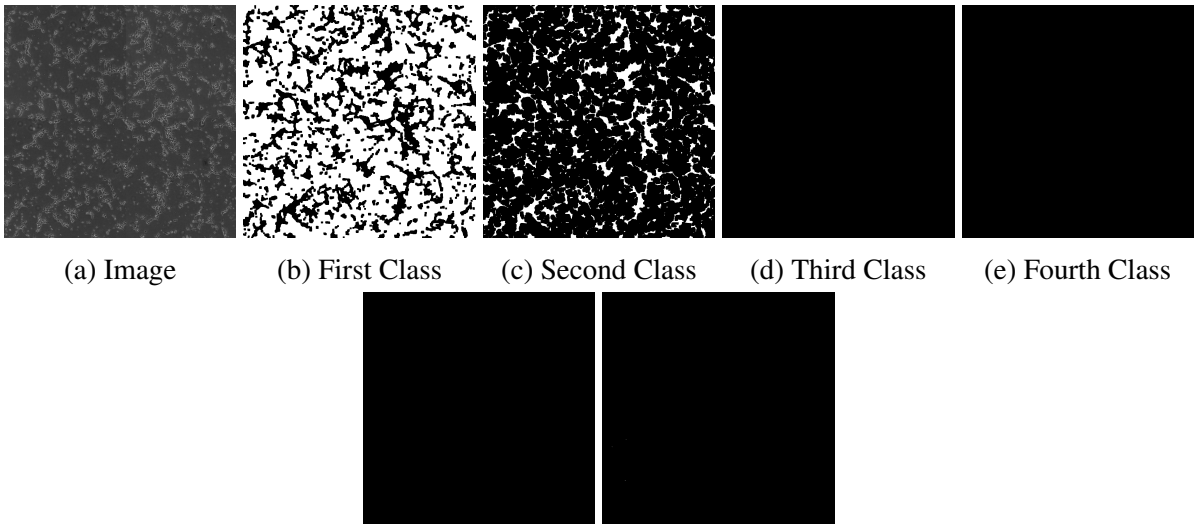
UKA contains 71 images with mean dimension 2657x3395 of induced pluripotent stem cells obtained by Phase contrast microscopy. The object of interest is embroid bodies and the magnification factor is four. There are four classes, notably, the classes are quite imbalanced and one of the classes only occurs in a few images. The handcrafted meta-features are: average mean area: 76988.24, average std area: 439257.09, average min area: 0.5, average max area: 3290732.62. An example image can be seen in Figure 16.



(a) Image (b) First Class (c) Second Class (d) Third Class

Figure 16: Example image from the UKA dataset.

iPSC contains 40 images with mean dimension 2666x2666 of induced pluripotent stem cells obtained by Phase contrast microscopy. The object of interest is stem cells and the magnification factor is 4. There are six classes, some of these classes are underrepresented and only occur in a few of the images. In general, while there are six classes in this dataset an image contains at most four classes. The other handcrafted meta-features are: average mean area: 12026.07, average std area: 91226.61, average min area: 1017.0, average max area: 2360599.75. An example image can be seen in Figure 17, note that only two classes are present in this image and the other images are therefore black.

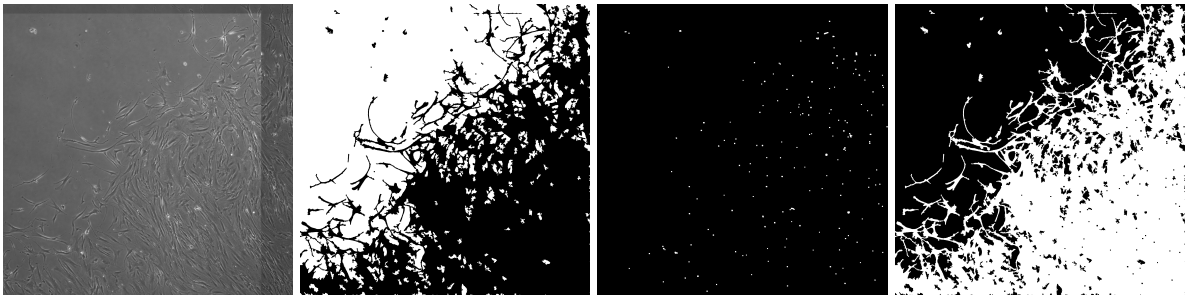


(a) Image (b) First Class (c) Second Class (d) Third Class (e) Fourth Class

(f) Fifth Class (g) Sixth Class

Figure 17: Example image from the iPSC dataset.

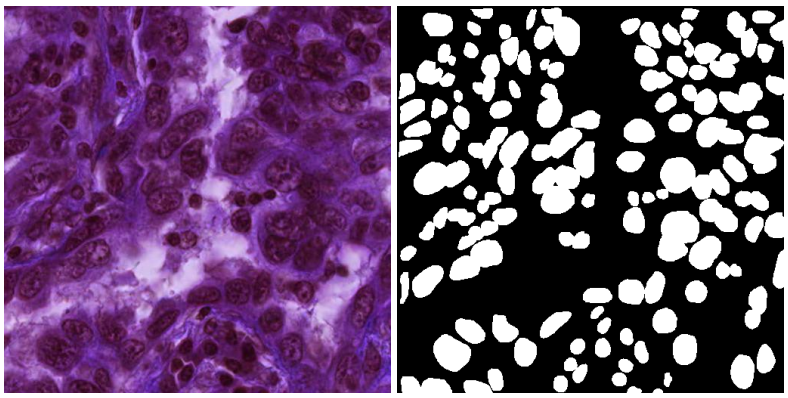
MSC contains 202 images with mean dimension 2052x2052 of stem cells obtained using Phase contrast microscopy, there are three classes. The object of interest is stem cells and the magnification factor is 4. The other handcrafted meta-features are: average mean area: 6851.93, average std area: 82203.03, average min area: 0.66, average max area: 2787744.0. An example image can be seen in Figure 18.



(a) Image (b) First Class (c) Second Class (d) Third Class

Figure 18: Example image from the MSC dataset.

NuceleiSegBench [74] contains 120 images with mean dimension 1000x1000 of tissue images obtained using fluorescent microscopy, there is one class. The object of interest is nuclei and the magnification factor is unknown. The other handcrafted meta-features are: average mean area: 409.63, average std area: 569.87, average min area: 0.0, average max area: 18186.0. An example image can be seen in Figure 19.



(a) Image (b) First Class

Figure 19: Example image from the Nuclei segmentation benchmark dataset.

BBBC020 [76] contains 20 images with mean dimension 1040x1388 of macrophages obtained using fluorescent microscopy, there are two classes. The objects of interest are the cell surface and nuclei, the magnification factor is unknown. The other handcrafted meta-features are: average mean area: 8440.43, average std area: 25047.10, average min area: 2.0, average max area: 372536.0. An example image can be seen in Figure 20.

(a) Image (b) First Class (c) Second Class

Figure 20: Example image from the BBBC020 dataset.

4.5.2 Meta data collection

To create the meta-base, for each dataset pipeline configurations were sampled without replacement from the search space. These pipeline configurations were then trained on the respective datasets. By sampling configurations without replace for each dataset, it is ensured that each configurations is only trained once on each dataset. Since there are eight datasets and 576 possible configurations, there are a total of 4608 possible datapoints. To evaluate how well pipeline configurations perform after training they were evaluated on a validation set. During each training run the datasets were randomly divided into 80% training and 20% validation images. In this way, the train validation split is different each time. Configurations were evaluated on two different GPU's, a P-100 and an A-100. Both are recent and powerful, however, the A-100 is faster. This training on different hardware limits the comparability of configuration training times. Training configurations for a set amount of epochs that ensures that configurations always converge to optimal performance is expensive and potentially unnecessary. To limit training times and still ensure convergence, early stopping was used. Configurations were trained until the validation set IoU score changed no more than 0.05 for eight consecutive epochs. A relatively large performance delta of 0.05 was chosen because on some datasets there was a large variability in validation IoU score between different epochs. A potential downside of these early stopping parameters is that some configurations might not reach a performance maximum. Nonetheless, these parameters are a good trade-off between ensuring convergence and not training configurations longer than necessary. In some way, using early stopping to create the meta-base can be seen as using a low-fidelity approximation.

4.6 Meta-Feature extraction

The goal of the Meta-Features is to quantify datasets in a way that represents the characteristics important to the performance of pipeline configurations on different datasets. To this end, three different types of features were extracted. These are meta-features that DL experts classically deem important, meta-features from the expert user, and meta-features about the optical appearance of images.

4.6.1 Handcrafted Meta-Features

These refer to dataset features that were deemed import and therefore extracted from datasets. These are: The number of instances, the number of classes, the class distribution, the mean image shape and the image format (RGB or grayscale). Mean image shape refers to the mean of the average width and height of the images. Class distribution is calculated pixel wise, for each class the number of

pixels of that class over the total amount of pixels. Therefore, the class distribution is represented by a list of length number of classes. Since there needs to be an equal number of meta-features for each dataset that list was compressed to a scalar value, aiming to represent class imbalance. This scalar is the average distance between class distribution values. In this way, the scalar value represents how imbalanced a dataset is.

In addition, information about the ROIs in the images was extracted. To this end, the mean, standard deviation, minimum, and maxima of ROI sizes were extracted. These were calculated for each class and averaged for each image, then averaged over all images in the dataset. This results in an additional four dataset meta-features.

Overall, nine handcrafted meta-features were extracted for each dataset.

4.6.2 User given Meta-Features

The biomedical expert that uses the system can also provide useful information about a dataset. This includes the microscopy technique, object of interest and the magnification level. The eight datasets in the meta-base contain three different microscopy techniques, five different objects of interest and four magnification levels. The microscopy techniques and objects of interest were represented as binary vectors and the magnification level as a scalar value. Moreover, an additional feature representing if the magnification level is known for a dataset was included. After the binary encoding there are ten user given meta-features for each dataset.

4.6.3 Optical Meta-Features

To extract optical information of the image dataset, radiomics features were extracted. For a description of radiomics, please see section 2.4. The python implementation pyradiomics [56] was used to extract 95 radiomics features per class. For each ROI a set of 95 features is extracted. These feature vectors are then averaged and one 95-dimensional radiomics vector is obtained for each image. Averaging all image level vectors leads to one radiomics vector for each dataset. Since these radiomics features have different scales, feature-wise normalization was performed. This was done in two ways, once over all image radiomics vectors and once over all dataset radiomics vectors. Feature-wise normalization was used since instance-wise normalization (normalizing each radiomics vector) would still lead to different feature scales.

In addition to feature-wise normalization, some dimension reduction techniques were applied. T-SNE [80] is used to visualize the data and PCA [81] is used to compress the vector to four dimensions that still capture 94.45% of the total variance in the dataset.

To quantify dataset similarity the cosine similarity measure was used. Cosine similarity can be calculated between same dimensional vectors by dividing their dot product with their multiplied vector norms:

$$S_c(A,B) = \frac{A.B}{\|A\| * \|B\|}$$

For each dataset the cosine similarity to all other datasets was calculated and these values were added as meta-features.

In total there are four (PCA), eight (cosine similarities) or 95 (all) radiomics meta-features for each dataset.

4.6.4 Feature-engineering

All meta-features described above, are constant for each dataset and all the within-dataset variance comes from the varying hyperparameters, see Table 1. To create more meaningful meta-features, dataset meta-features and hyperparameter settings were combined. The following meta-features were created based on handcrafted dataset features:

- number of images = (mean image shape / patch size)^2
- batches per epoch = (number of images * number of instances) / batch size
- augmentation batches = sum(image augmentations) * 2 * batches per epoch
- learning rate batches = learning rate * batches per epoch
- backbone classes = backbone * number of classes

Here sum(image augmentations) refers to how many of the three augmentations are potentially applied and backbone is set to 0.5 for ResNet-18 or 2 for ResNet-50.

The radiomics meta-features were also combined with the hyperparameters, to capture potential interaction effects between the two. Either the first four principal components or the eight similarity scores were multiplied with the learning rate, backbone and number of classes. In this way, an additional four or eight meta-features were created.

4.6.5 Experiments

To evaluate the potential of radiomics to represent images belonging to different datasets, t-sne is used to visualize radiomics vectors in two dimensions. Moreover, the cosine similarities of radiomics vectors were evaluated in a qualitative way. The effect of feature-wise normalization on radiomics vectors was evaluated using both t-sne and cosine similarity scores.

The goal of these different meta-features is to allow the meta-model to accurately rank different configurations. Therefore, a variety of different meta-features combinations were tried for each meta-model. In Figure 21 the 11 different meta-feature representations that were evaluated are illustrated. The performance of these different dataset representations was evaluated using the metrics described in section 4.7.1, moreover, the evaluation strategy is the same as for the meta-models.

Meta-Feature representation	Hyperparameter settings	Handcrafted Meta-Features	Engineered Handcrafted	User-given	4 PCA's Radiomics Vectors	Radiomics cosine similarity	Engineered radiomics PCA	Engineered radiomics similarity	95-dim Radiomics Vectors
1	X	X							
2	X	X	X						
3	X	X	X			X			
4	X		X						
5	X	X					X		
6	X	X				X			
7	X	X	X				X		
8	X	X	X					X	
9	X	X		X					
10	X	X	X	X	X	X	X	X	
11	X								X

Figure 21: Evaluated Meta-Feature combinations to represent datasets. Fields marked with an **X** indicate that a meta-feature representation is included in the evaluated representation.

4.7 Meta-Learning

The main part of the AIxCell system is meta-learning. Here, a machine learning model is trained on the configurations of the meta-base and then predicts a ranking of configurations for a new dataset. The input to the ML-model is the hyperparameter vector concatenated with a meta-feature vector. The output was represented in two different forms first the validation set IoU score and second an ordinal division of configurations. Here, configurations were assigned to ten bins based on their validation set IoU performance, the worst 10% of configurations were assigned the label one the second worst 10% the label two, and so on. The top 10% of configurations were assigned the label ten. This was done to more clearly separate configurations since for many datasets there is not much variance between different configurations.

4.7.1 Metrics

A key part of developing the meta-learning configurator was to find appropriate metrics. Here the metrics used to evaluate the performance of meta-models will be described.
For the regression models the root mean squared error:

$$RMSE = \sqrt{1/n \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

was used to assess how accurately the model predicts validation set IoU scores.
Pearson's r is a measure of linear correlation between two variables. It is the ratio between the covariance of two variables and the product of their standard deviations, when these are estimated using a sample $r_{X,Y}$ is calculated by:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

The possible values for Pearson'r r range from -1, indicating a perfect negative correlation and 1 indicating a perfect positive correlation. A value of 0 indicates that two variables are not related.
Kendall rank correlation Tau is a measure to determine the correlation between two rankings, it is calculated based on the number of concordant and discordant pairs. Any pair of observations (x_i, y_i) and (x_j, y_j) are concordant if the sort order of (x_i, x_j) and (y_i, y_j) agrees, that is either $x_i < x_j$ and $y_i < y_j$ or $x_i > x_j$ and $y_i > y_j$. Otherwise, a pair of observations is said to be discordant. Kendall τ is then defined as:

$$\tau = \frac{n_{concordant} - n_{discordant}}{n_{total}}$$

The possible values for Kendall's tau range from -1, two rankings being inverse, to 1 meaning that two rankings are identical.
The task of the meta-model is to limit the search space for good pipelines so that successive halving can be used to find the optimal one. Therefore, two measures to evaluate how well the model predicts the top 10% of configurations were also included. First, the difference between a random selection and using the meta-model to limit the search space was assessed using the z-score. Here the z-score is calculated as:

$$z = \frac{\mu(\hat{y}) - \mu(y_{all})}{\sigma(y_{all})}$$

where $\hat{y}$ are the IoU scores of the predicted top 10% configurations and y_{all} are the IoU scores of all configurations. The larger the z-score, the better the selection from the meta-model is compared to a random selection.
Second, the difference between the predicted top 10% and the actual top 10% is evaluated as:

$$diff_{iou} = \mu(y) - \mu(\hat{y})$$

where y are the IoU scores of the actual top 10% of configurations.
All evaluation metrics were computed for each dataset and then averaged to evaluate the performance of a model and/or feature engineering strategy.

4.7.2 Meta-models

Loss Functions The pointwise and pairwise ranking approaches were compared. These approaches differ in the loss functions used. Let $x = x_1, ..., x_n$ be the instances to be ranked and $R = r(1), ..., r(n)$ be their corresponding relevance scores. These are either raw validation set IoU scores or a binning based on them. If $r(i) > r(j)$ then x_i should be ranked higher than x_j. For pointwise ranking the loss function is then:

$$L^r(f; x, R) = \sum_{i=1}^{i=n} \sqrt{(f(x_i) - r(i))^2}$$

In pairwise ranking the loss function is:

$$L^p(f; x, R) = \sum_{i=1}^{i=n-1} \sum_{j=1, r(j)<r(i)}^{j=n} \phi(f(x_i) - f(x_j))$$

where ϕ is the hinge function: $\phi(z) = max(0, 1-z)$. Note that even though the model is performing binary classification it still learns a relevance score, so the output of $f(x)$ is some scalar value. In words, the loss for one instance is: For all instances that have a lower true relevance score, how well does the model recognize these lower instances. This is often referred to as a maximum-margin loss, the larger the difference between two scores the larger the loss.

Regression Models Five different models were evaluated to create the ranking using the pointwise approach. These model range from simple to more complex: Linear regression, decision tree, support vector machine (SVM), random forest and XGBoost. For the SVM the data was normalized, replacing every sample with its corresponding z-score. The other models do not require normalization.

LamdaMART is a state-of-the-art algorithm for learning to rank that has achieved top performance on a variety of experimental datasets [82]. It combines LamdaRANK and MART (multiple additive regression trees). The gradient-boosted decision trees from MART are extended with a cost function derived from LambdaRank to order any ranking situation. So instead of predicting pseudo-residuals based on a regression metric, as in regression trees, the trees predict some ranking loss function. Here, the pairwise ranking loss function described above is used.

4.7.3 Experiments

Meta-models and meta-features were evaluated using leave-one-dataset out cross-validation. This means that a meta-model is trained on the meta-data of seven datasets and then tested on the meta-data of one dataset. The meta-data of each dataset is the testing set once and the performance of a meta-model and meta-feature combination is then the average of all test datasets. Moreover, due to the meta-base being small in size it was observed that the performance of meta-models varied a fair bit due to randomness in the training process. Therefore, all meta-models where model training includes a random process (SVM, decision tree, random forest) were evaluated over 30 random seeds. For XGBoost the training process is deterministic on default settings, nonetheless, there are two hyperparameters that lead to a randomized behavior. Subsample by tree refers to how much of the training data is used to construct each tree and Colsample by tree refers to what percentage of features is used to construct each tree. To estimate the potential over-fitting of the default XGBoost, XGBoost with subsample=0.9 and colsample=0.9 was also evaluated over 30 random seeds.
The following meta-models were evaluated for every meta-feature representation (see section 4.6.5):

- Linear Regression
- Decision Tree
- SVM
- Random Forest
- XGBoost Regressor
- LamdaMART
- LamdaMART with randomized training

Each model was also evaluated given the two relevance score representations discussed above. In total, 168 (2x7x12) different meta-learning configurations were evaluated.

4.8 Successive Halving

To evaluate the top k configurations of the ranking predicted by the meta-model, successive halving was used. Next to this k configurations the input also includes a time limit, this is the total budget. As described in section 2.3.5, the worst half of configurations is dropped after each step until

only one configuration remains. Therefore, first, the number of steps required to end up at a single configuration, given k configurations, is calculated by:

$$n\ steps = floor(\log(k)/\log(2)) + 1$$

Then the budget per step is calculated, budget step = total budget / n steps. For the first step, each configuration is assigned a budget of budget step / k, and for the second step the budget for each configuration is doubled since k is divided by two. In this way, the algorithm proceeds until only one configuration remains. The performance of configurations is the validation set IoU score and configurations are stopped when their time budget runs out.

4.9 System Evaluation Experiments

The performance of the AIxCell system is evaluated in two ways, on two new open-source datasets and on some of the datasets used to curate the meta-base. For the evaluation the best 50 predicted configurations are used to start the successive halving process. With 50 configurations there are six steps in successive halving, therefore 27.38% of the total budget is used to train the best configuration. The time budgets for each of the datasets were adjusted based on their dimensions and total number of instances.

The first open-source dataset is Fluocells [83], with 283 images of neuronal cells, of mean dimension 1600x1200, obtained using Fluorescent microscopy. The task is binary segmentation, separating neuronal cells and background. An example image and mask are shown in Figure 22. The dataset was split into 0.8 training, 0.1 validation and 0.1 testing. Due to the large number of instances and relatively large image dimensions of the Fluocells dataset, the AIxCell system was given a compute budget of 6 hours. The best configuration is therefore trained for 1.64 hours.

The second public dataset used is the Electron Microscopy Dataset from EPFL [84]. For this dataset a pre-defined split into training and testing is available. The training set consists of 165 images of mitochondria, with dimension 1024x768, obtained using Electron microscopy. Again, there is one class and here the task is to segment mitochondria. An example image can be seen in Figure 23. The testing set also contains 165 images of the same dimensions. To evaluate performance the training set was split into 0.8 training, 0.1 validation and 0.1 testing. Note that the testing set is never used here due to the existing test set, nonetheless the split is set at a default and therefore not affected by this.

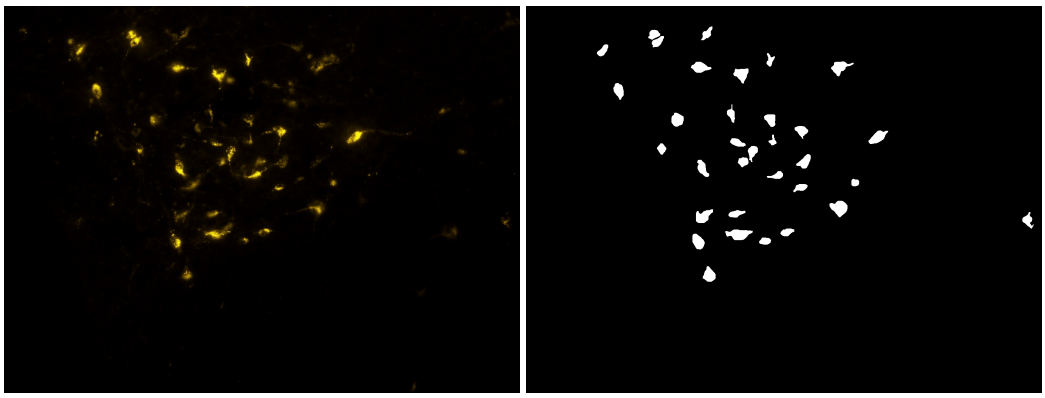


Figure 22: Image and Mask from the Fluocells dataset

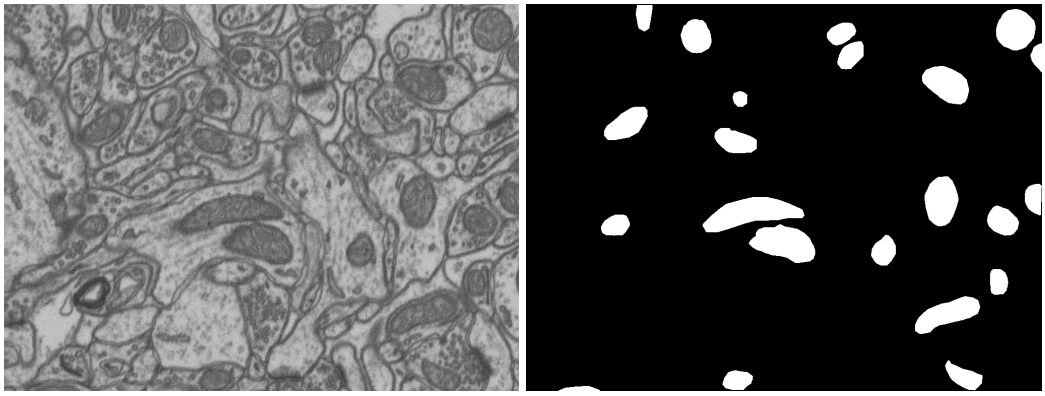


Figure 23: Image and Mask from the Electron microscopy dataset from EPFL

Next to evaluating the system on novel datasets it was also evaluated on two of the datasets in the meta-base. Namely, on the UKB and iPSC datasets. To this end, the meta-model was only trained on the meta-data of the seven other datasets. Since for both of these use-cases specific pipelines were developed by students beforehand, this allows for a comparison between the AIxCell system and human experts. For both datasets a train, validation and testing split of 0.8, 0.1, 0.1 is used.
The UKB dataset is relatively small, therefore the AIxCell system was given a compute budget of 4 to train the top configuration for 1.09 hours. The iPSC dataset only contains 40 images, however these are of very large dimension. Moreover, there are 6 classes and it is therefore reasonable to assume that longer training times might be needed. The AIxCell system was given a compute budget of 6 hours for this dataset.

5 Results

In this section, the results of the described experiments for answering the main research question on how meta-learning based approaches can be used to optimize performance of domain-specific AutoML systems for image data, at the example of cellular image analyses are presented. To this end four sub-research questions are investigated. The first one is about the effects of transfer learning 5.1, the second is about optimal meta-features for image data 5.2, the third is about optimal meta-models and feature engineering 5.4, and the final one is about the effectiveness of successive halving 5.5.

5.1 Transfer Learning

For transfer learning, the research question was whether pre-training the model will lead to shorter training times and increased performance. In addition, the effect of domain-specific and generic pre-training was investigated. To this end, the model is pre-trained on two datasets. On a validation set of the cellpose dataset the model achieved an IoU score of 0.6867 and on a validation set of the kaggle dataset an IoU score of 0.7341. These are high IoU scores and therefore it appears that the model converges to a performance maxima for both datasets.
Nonetheless, there are a few noteworthy observations. On the UKA dataset using a model pre-trained on ImageNet or kaggle leads to an increase in train set performance (see Figure 26). This increase is mainly due to the model predicting the third class more accurately, this class only occurs on a few images and thereby only on a very low percentage of the total pixels. So it appears that pre-training can help to identify such minority classes. However, that effect was not present on the validation dataset. On the RUB dataset (see Figure 24), the model pre-trained on ImageNet has a significantly lower validation set IoU score and the validation set IoU takes longer to increase. Therefore, on this dataset it appears that generic pre-training is actually harmful to model performance. For all datasets, the pre-trained models initially have a lower IoU score than the non pre-trained model, the models require slightly longer training to reach a comparable validation set IoU score. This effect is only present in the first one or two epochs and more significant on the UKA and RUB datasets.
In conclusion, the experiments show that, on the evaluated datasets, for semantic-segmentation pre-trained models do not perform better and do not have shorter training times. Moreover, results on the RUB dataset indicate that generic pre-training can be harmful to model performance. On all datasets, pre-training lead to slightly longer convergence times. There also appears to be no difference between generic and domain-specific pre-training.

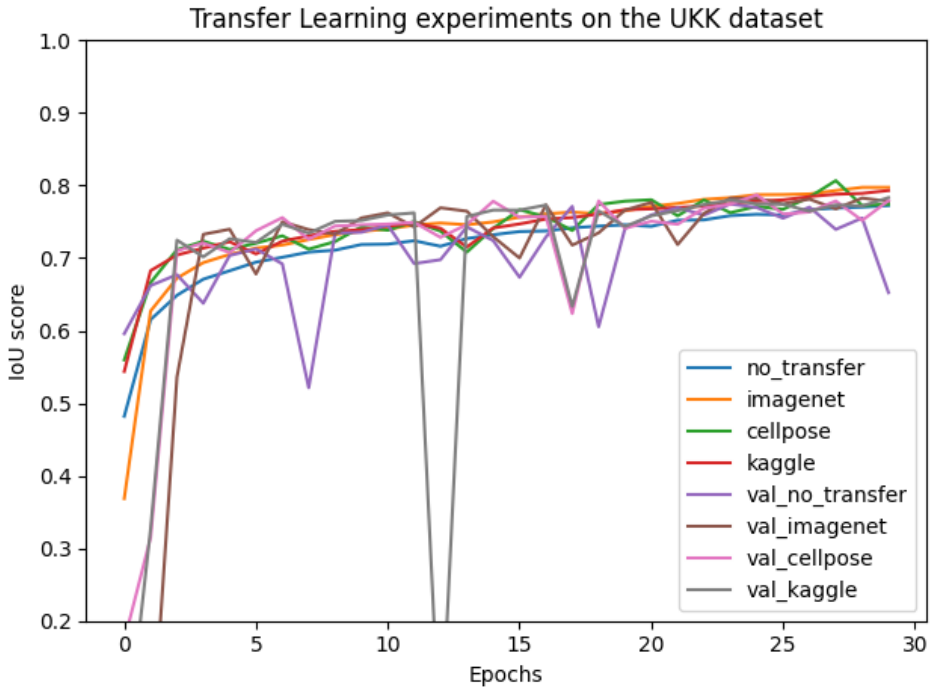


Figure 25: Model learning curves for different pre-training settings on the UKK dataset

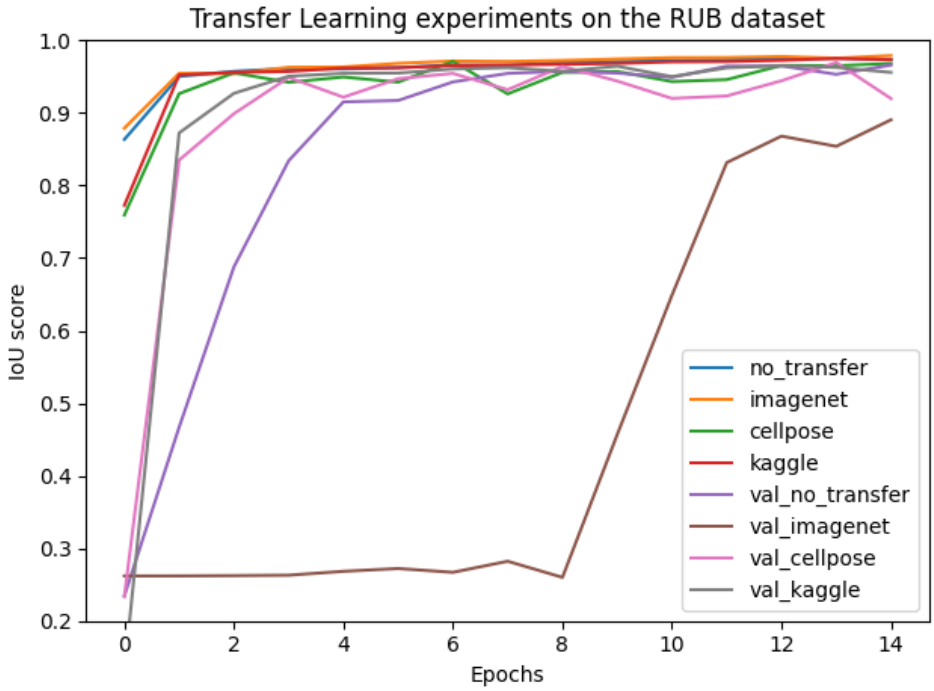


Figure 24: Model learning curves for different pre-training settings on the RUB dataset

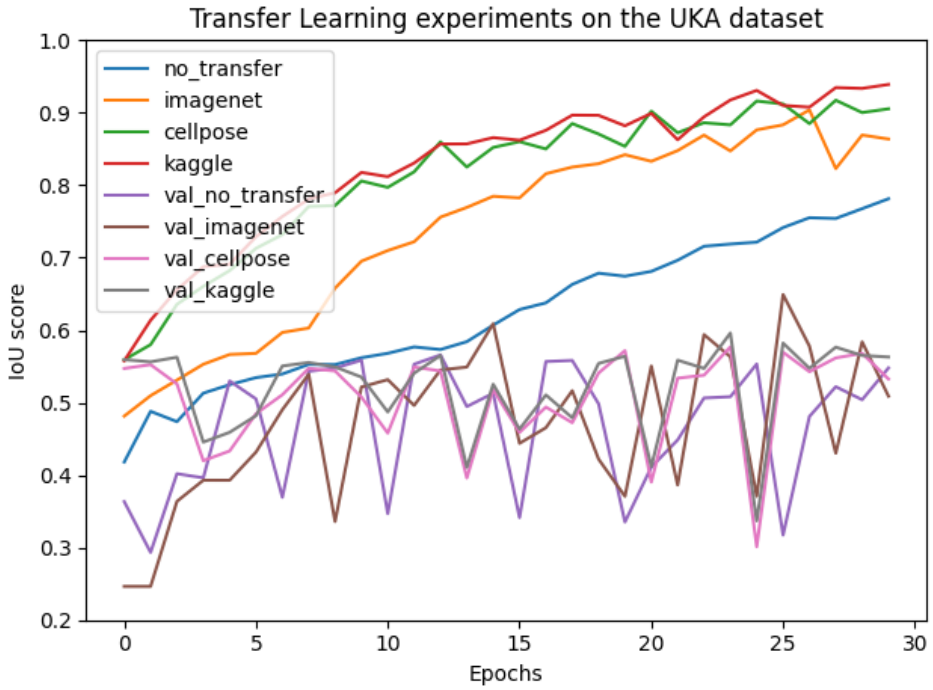


Figure 26: Model learning curves for different pre-training settings on the UKA dataset

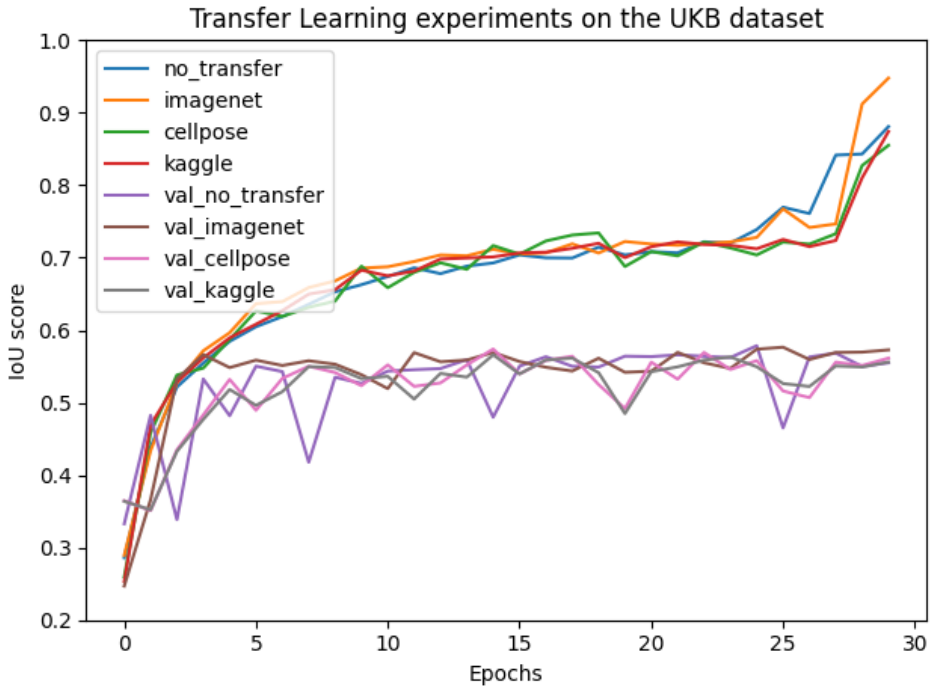
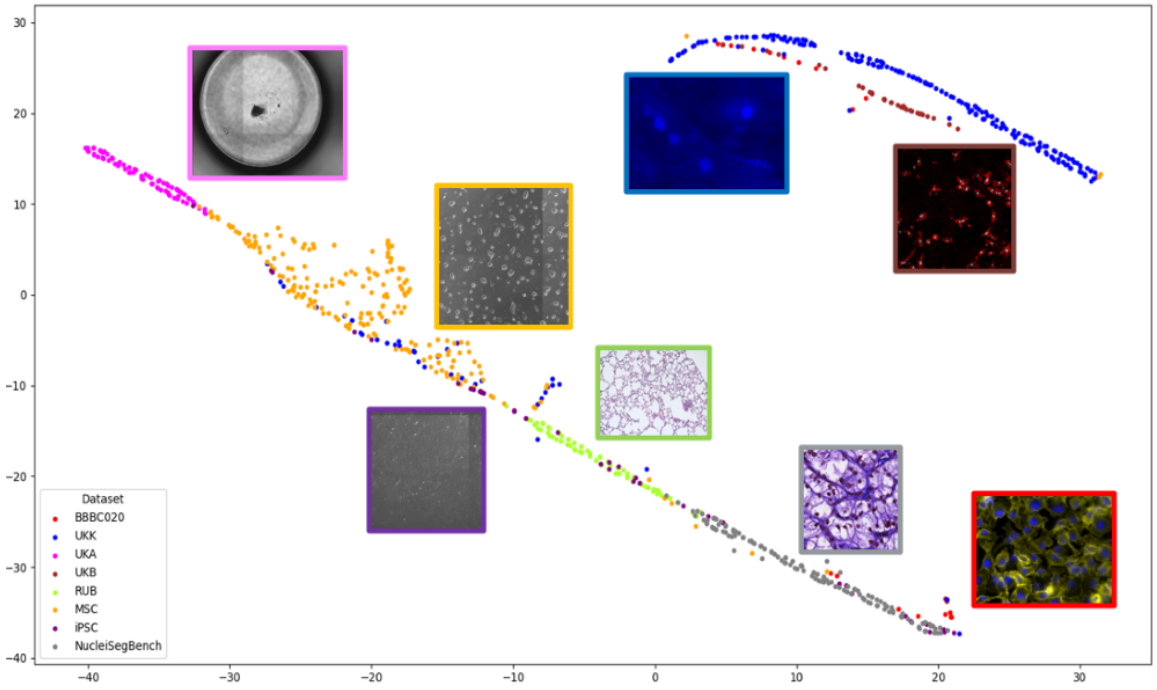
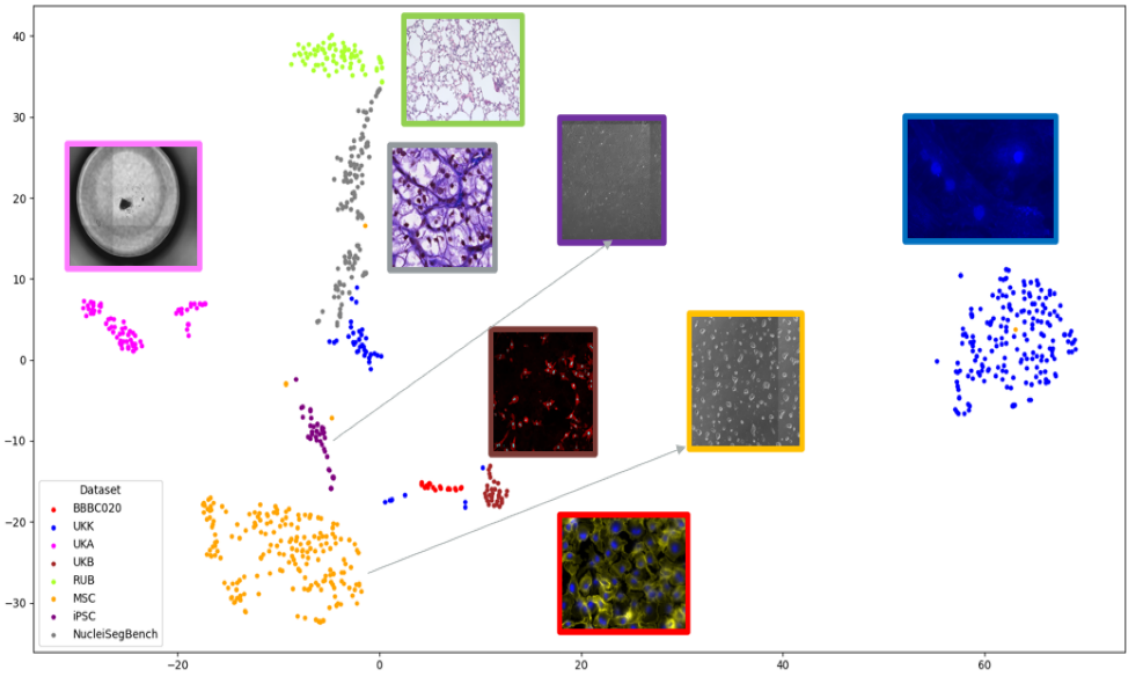


Figure 27: Model learning curves for different pre-training settings on the UKA dataset

5.2 Meta-Features for Optical Appearance

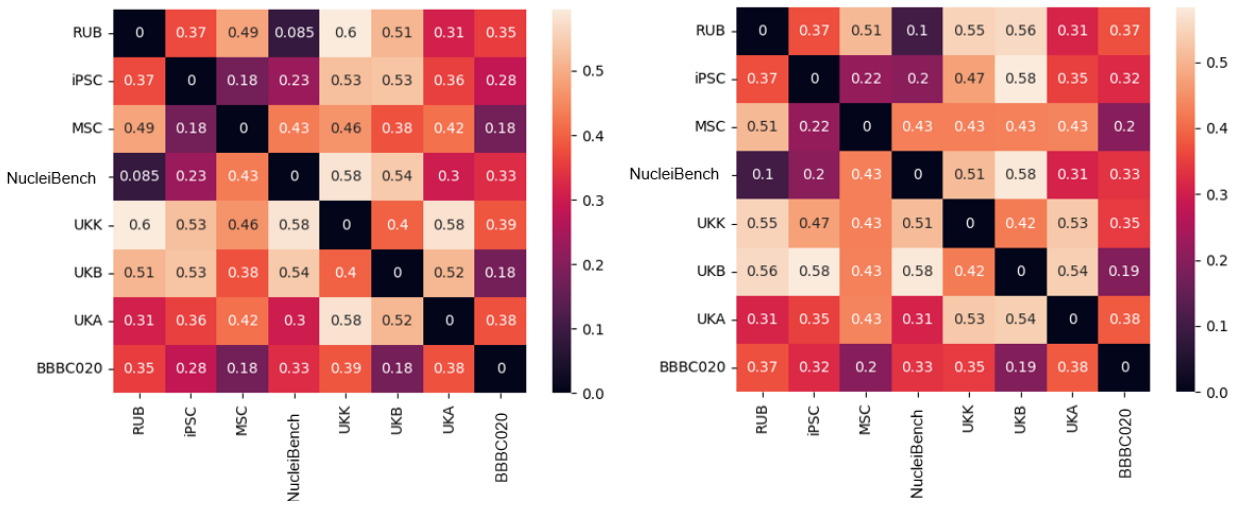


(a) No feature-wise normalization applied.



(b) Feature-wise normalization applied.

Figure 28: Radiomics vectors of all images of the meta-base visualized in two dimensions. The colors represent the dataset the vectors belong to, examples images for each dataset are also shown. Feature-wise normalization clearly leads to a better clustering of images belonging to the same dataset.



(a) Dataset-level radiomics vectors obtained using all images in each dataset.

(b) Dataset-level radiomics vectors obtained using 20 images for each dataset.

Figure 29: Cosine distance scores between the dataset level radiomics vectors. A value of zero indicates vector directions to be identical and a value of one indicates orthogonal vectors. Only using 20 images for each dataset, leads to a good approximation of the true cosine similarity.

The goal of extracting radiomics features from the images in the datasets is to capture optical information. This optical information could then be important for the meta-model to predict good pipeline configurations. To investigate if radiomics vectors are suitable to represent optical information, the originally 95-dimensional vectors are visualized in two dimensions using t-sne. Moreover, cosine similarity scores between all datasets are calculated and qualitatively examined.
The radiomics vectors were visualized and then analyzed. First, t-sne was used to visualize the radiomics vectors of all images in two dimensions. In Figure 28, the benefit of applying feature-wise normalization is clearly visible. When feature-wise normalization is applied, see Figure 28a, the radiomics vectors of images belonging to the same dataset cluster together. The datasets clusters that are closest to each other also look most similar. For instance, the MSC and iPSC dataset both contain images of stem cells and were obtained using phase contrast microscopy, therefore their images look very similar. The radiomics vectors of images in these datasets cluster closely to each other, although there is still a clear separation between the two. Based on the visual investigation of the t-sne dimension reduced radiomics vectors, radiomics vectors are suitable to represent images of different datasets. Therefore, this visual investigation confirms that radiomics vectors are suitable to express the optical features of images in vector form.
The same applies when looking at the cosine distance scores between dataset level radiomics vectors, see Figure 29. Datasets that look similarly visually also have a low cosine distance and datasets that are visually different have a higher cosine distance. RUB and NucSegBench have the lowest cosine distance with 0.085 and when comparing the images of the two they also look very similar. Conversely, between UKK and RUB the cosine distance is the largest. In general, the cosine similarity scores (and t-sne visualization) indicate that, based on optical features, the UKK dataset is the most different from all other datasets. The cosine similarity scores between dataset level radiomics vectors can be well approximated if only 20 images of each dataset are used to determine dataset vectors, see Figure 29 a. The cosine similarity scores differ slightly, however, the maximal difference is 0.04 and

the overall ordering of similarities stays the same. This can be helpful to reduce the computational cost of radiomics feature extraction for large datasets.

5.3 Meta-Base Statistics

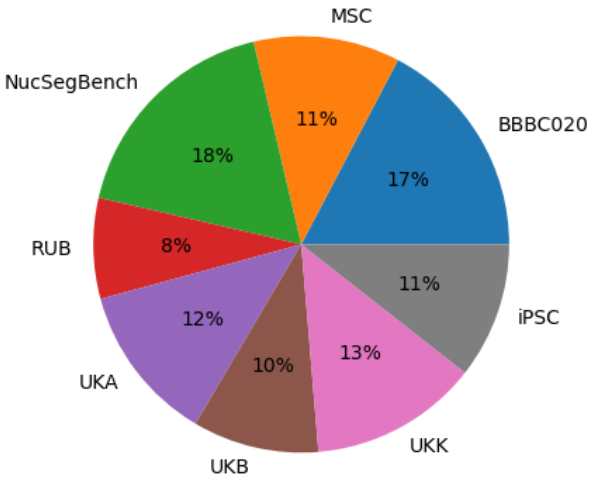


Figure 30: Percentages for how many data points in the metabase belong to which dataset.

Overall, 1.458 different configurations were trained on the different datasets. A distribution of how many configurations were trained on each dataset can be seen in Figure 30. There are 4.608 possible data points, so the metabase covers 31.64% of the search space. Due to the training of configurations sometimes stopping due to technical problems, the training time of some configurations was short. Moreover, due to manual errors when extracting the data from the cloud, the validation IoU scores of some configurations did not make sense (for instance a validation set IoU of zero). To eliminate these configurations outlier removal based on the absolute z-score of configurations was done. First all configurations that had total training times with a z-score larger then three (computed over the dataset train time distribution), were removed. Next, all configurations with a z-score larger than four, based on the validation set IoU score, were removed. This resulted in 35 configurations being removed from the meta-base. After outlier removal, the meta-base consists of 1.423 datapoints and covers 30.8% of the search space.
The distributions of the validation set IoU scores, after outlier removal, can be seen in Figure 31. The scores on most datasets roughly follow a normal distribution, however for RUB and NucleiSegBench the distributions are skewed toward the right. There are a lot of configurations with good scores and only a few that perform significantly worse. The mean scores differ significantly between datasets, iPSC seems to most difficult with a mean score of 0.48 while RUB is easiest with a mean score of 0.95. The average standard deviation in validation set IoU score is 0.039 and the within-dataset standard deviation varies between datasets. The dataset with the lowest standard deviation is RUB with $\sigma = 0.021$ and the dataset with the largest standard deviation is NucSegBench with $\sigma = 0.075$.

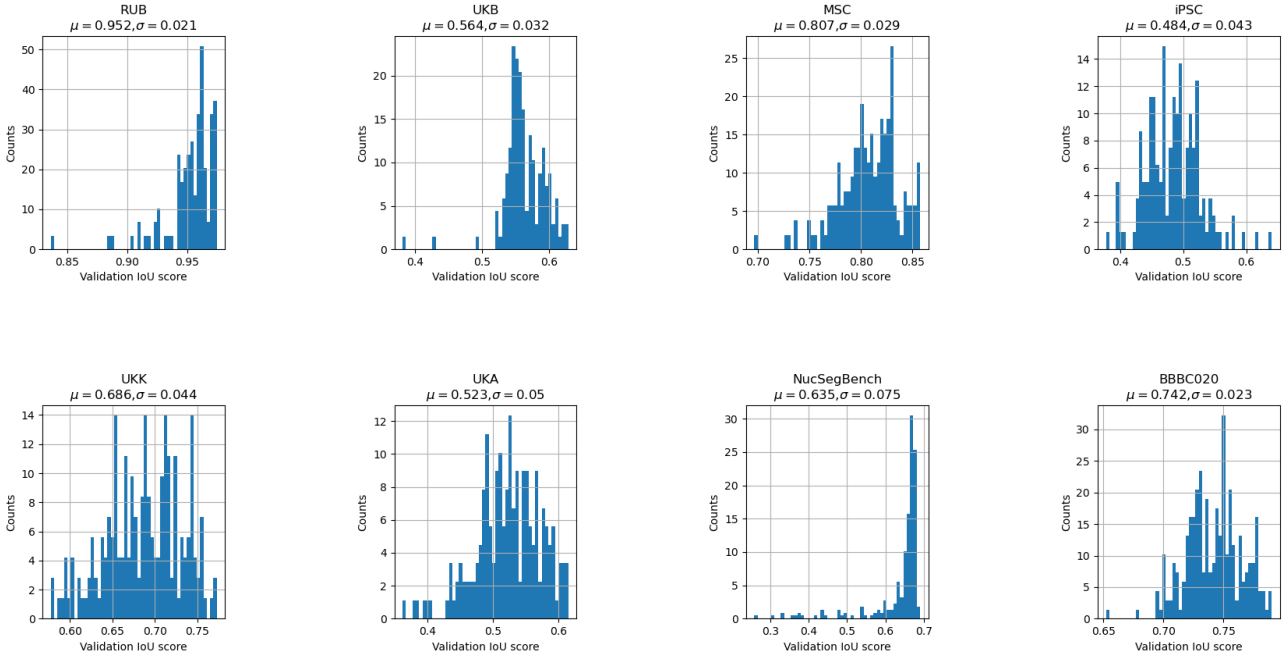


Figure 31: Distribution of validation set IoU scores on each dataset. Scores were divided into 50 bins for visualization.

5.4 Meta-Learning

Meta-Model	Feature Engineering	Z-score	Pearson	Diff IoU	RMSE
Random Forest	Handcrafted, engineered handcrafted	0.643	0.295	0.028	0.138
LamdaMART deterministic	Only handcrafted	0.608	0.299	0.030	
LamdaMART deterministic	Handcrafted, eng handcrafted, radiomics sim	0.606	0.291	0.028	
Random Forest Binned IoU	Only handcrafted	0.601	0.346	0.029	
Random Forest	Handcrafted, eng handcrafted, radiomics sim	0.583	0.308	0.030	0.157

Table 2: The top 5 combinations of meta-features, meta-model, and target representation, measured by z-score. Binned IoU score below the meta-model indicates the binned target representation, otherwise the results refer to raw IoU scores.

An excerpt of the results is shown in Table 2 the best 5 configurations measured by z-score are shown. For the comprehensive results, refer to Appendix A.
The best combination of meta-model, feature engineering, and target representation is: random forest with handcrafted and engineered handcrafted features predicting raw IoU scores. In this way, a selection based on the top 10% predicted configurations is 0.643 standard deviations better than a random selection. This is a decent results considering that the true top 10% of configurations are only 1.41 standard deviations better than random selection. In its deterministic setting, meaning that

all data points and features are used for each tree, LamdaMART performs second best. For target representation using the raw IoU scores is best. Nonetheless, the difference between binning the IoU scores and using raw IoU scores is quite small. Applying random forest on these binned IoU scores also performs well.

The z-score was chosen to evaluate meta-models to reiterate the z-score measures how much better a given selection is compared to a random selection. In this way, it most clearly represents what the meta-model is supposed to achieve. Limit the search space, so that successive halving can find the best configuration by training on the new dataset.

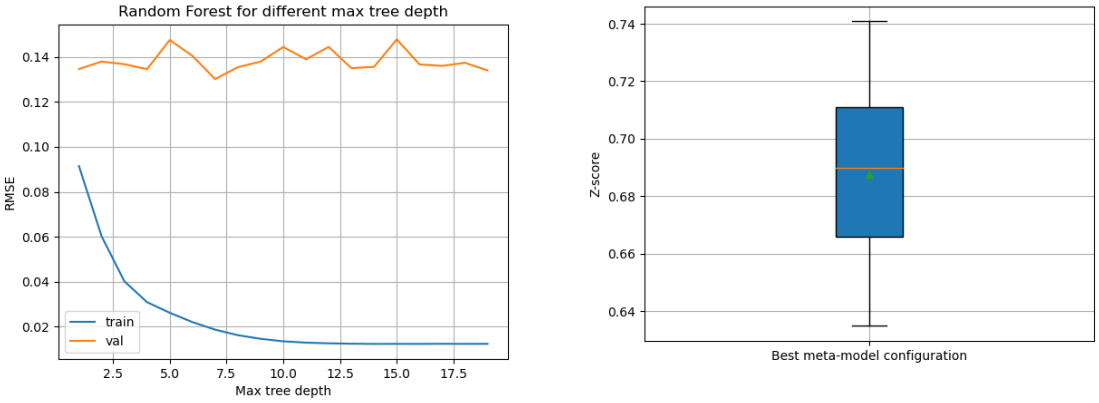
This measure also correlates with the Pearson correlation, difference in IoU and RMSE score. While the largest z-score is not always associated with the best score on these metrics, configurations with large z-scores always have one of the best scores on these metrics as well. The best meta-model configuration, with a z-score of 0.643, indeed has the lowest RMSE, 0.138, is tied for the lowest difference in IoU, 0.028, and also has a good Pearson correlation, 0.295. It is observed that using a binned IoU score target representation leads to better Pearson correlation. Unfortunately, all meta-models score badly when evaluated using Kendall's Tau. The scores for all models are around zero, meaning that according to Kendall's Tau there is no relationship between the predicted and true ranking. However, many meta-datapoints only differ by an IoU score lower then 0.001. Therefore, constructing an accurate ranking is not that important as long as the meta-model accurately assigns configurations to different tiers.

The results clearly show that tree based models perform best, with Random Forest slightly outperforming LamdaMART. As expected the linear regression baseline model performed worst with a top z-score of 0.303 using only the handcrafted meta-features. The SVM did not perform well either with a top z-score of 0.35 for handcrafted and engineered handcrafted features. For gradient boosting trees using the pairwise-ranking loss function did outperform the pointwise loss function. Randomized LamdaMART achieved a z-score of 0.55 while randomized XGBoost only achieved a z-score of 0.48. For both LamdaMART and XGBoost only using a subset of the training data and features for each tree results in worst performance. Overall, the best model is the random forest using the pointwise ranking method.

Including meta-features describing the dataset is clearly necessary, all models performed worst using only the hyperparameters. The handcrafted meta-features are the most important ones. This is not surprising, since these features correlate most with performance. For instance, a dataset with many classes will most likely have a lower IoU score. When examining the trees in the random forest, the first node was indeed almost always split based on the number of classes in the dataset and therefore the number of classes in a dataset leads to the largest decrease in impurity. So the number of classes and other handcrafted meta-features are used by the model to predict most of the IoU score. This is logical since these meta-features stay constant for each dataset and can therefore be used to predict the mean IoU score of a dataset. Since the within-dataset variance is quite small, accurately predicting the mean will already lead to a low RMSE. Nonetheless, these meta-features to predict the mean IoU score are not useful when it comes to predict within-dataset variance. As can be seen in Figure 33, the engineered handcrafted features and hyperparameter settings are most important to accurately predict the mean squared error of configurations on a validation dataset. Again this was to be expected, since these meta-features are the only ones that vary on a dataset level. Note that different meta-features are important to predict different datasets.

While the extracted radiomics features led to visually plausible results to quantify datasets, they are not used by the best meta-model. Nonetheless, the radiomics based (cosine-)similarity scores are used in two of the top five performing configurations. Only using the entire 95-dimensional radiomics vectors and hyperparameter settings as meta-features led to bad results for all meta-models. The same

goes for using only similarity scores or PCA. So it is clear that the handcrafted meta-features are most important, while radiomics features do not seem to add much value.
The user-given features are not used by any of the top performing meta-model configurations. Nonetheless, especially when the target is the binned IoU scores they do seem to add value. For LamdaMART in the randomized and deterministic setting, the combination of handcrafted meta-features and the user-given ones performs best, with a respective z-score of 0.551 and 0.583.



(a) Performance of the best model for maximal tree depth. The RMSE score are averaged over all eight dataset being the validation set once.

(b) Performance variation over 20 random seeds of the best meta model. The yellow line is the median z-score and the green triangle the mean z-score.

Figure 32: Left train and validation RMSE for different max depth. Right boxplot showing the variation of meta-model performance over 20 random seeds.

For the best performing model, it is observed that there was a very large difference between train and test set performance. Therefore, the random forest was evaluated for different numbers of maximal tree depths. In this way, the capacity of the random forest can be controlled. As can be seen in Figure 32a, the RMSE is lowest for a max depth of seven. Overall, there the impact of the max depth is not very significant and there is always a large gap between train and validation RMSE. As described, this can be explained by some meta-features being used to predict the mean IoU score on a dataset and this is not possible on the validation dataset.
Training the best model again with the max depth set to seven, improved the results. Averaged over 20 random seeds this results in an average z-score of **0.688**, an average RMSE of 0.136, an average Pearson correlation of 0.291 and a difference in IoU of 0.262. The performance of this model on each validation dataset is shown in Figure 34. As can be seen the model performs similarly well across all datasets, even though the z-scores are different. This shows that the z-scores always have to be looked at in the context of a specific dataset and are not a good measure to compare model performance between two datasets. A potentially better way to evaluate model performance could be to normalize the z-score based on the top z-score that can be achieved. One observation from the performance plots is that on most datasets the model does well at predicting bad configurations, but when predicting the best configurations the variance increases. This is especially the case for BBBC020, MSC, and UKB, where the meta-model predicts low IoU scores for some of the best true configurations. On MSC one of the true top 10% configurations is predicted to be the worst configuration. On all datasets except UKB, there is some overlap between the predicted top 10% and true top 10% (green points). Overall, the model performs worst on the UKB dataset. The best performance is achieved on the RUB dataset,

almost all points in the true and predicted top 10% overlap. The model used to create the plots has a z-score of 0.702, so slightly better than when the z-scores are averaged over 20 random seeds. As can be seen in Figure 32b the performance of the best meta-model varies over different random seeds. Nevertheless, the model always has a z-score above 0.64 which is a good performance.

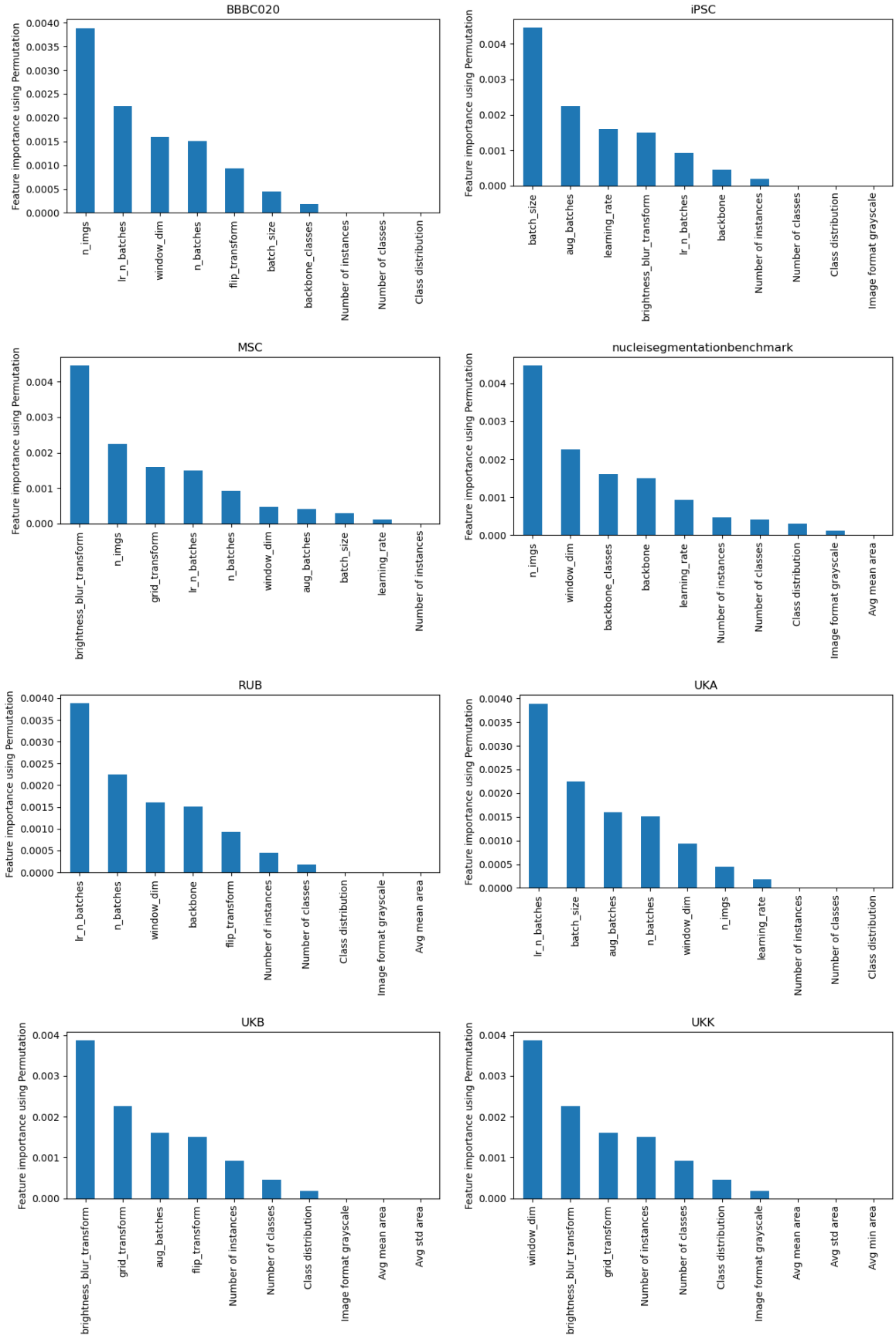


Figure 33: Feature Importance to predict IoU scores on each dataset. Importance is obtained using permutation: The model is trained with each feature being removed and the performance on the validation dataset is obtained. The feature importance is then the difference in validation dataset MSE, between training with and without the feature.

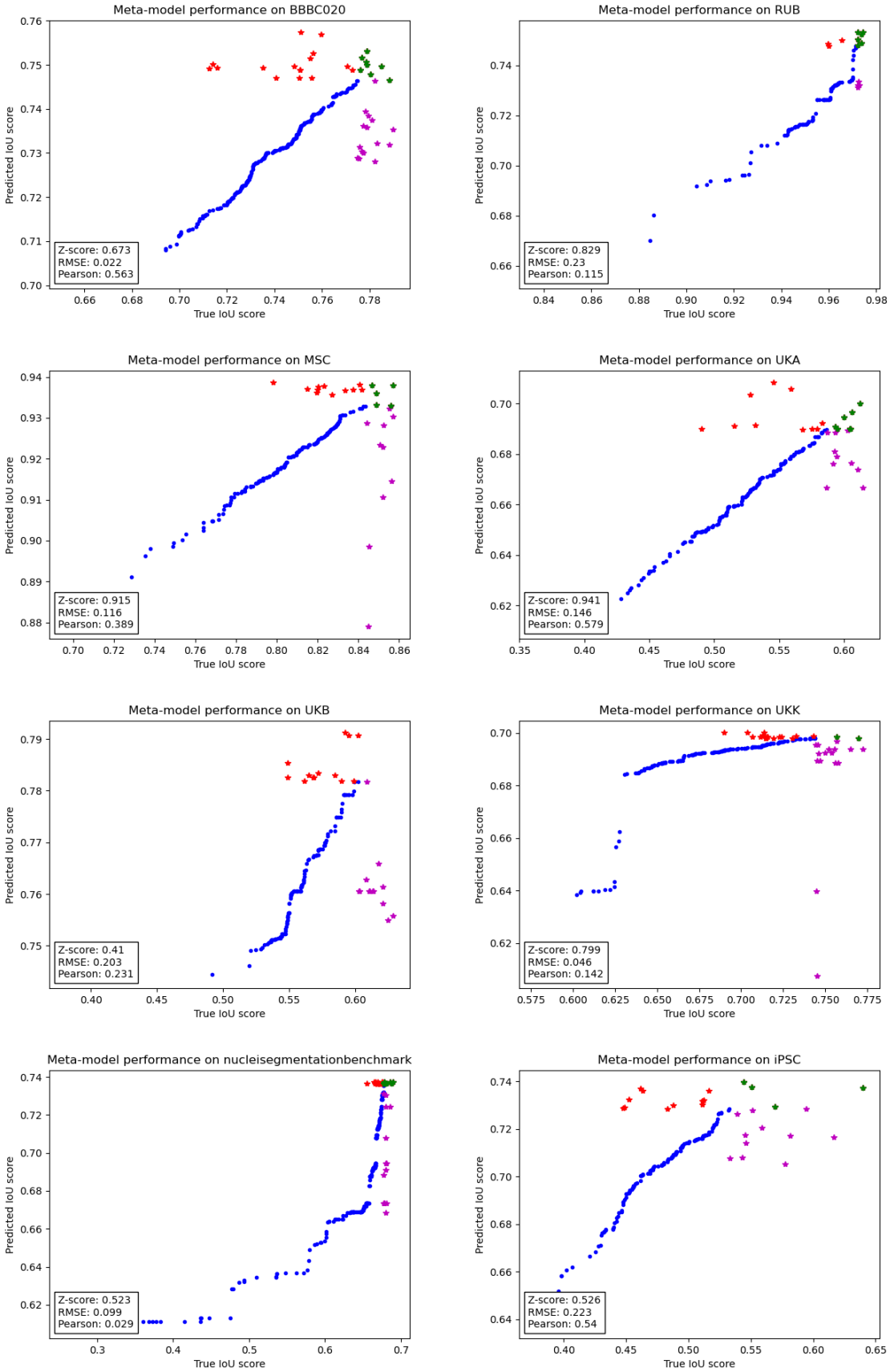


Figure 34: Validation set performance of the best meta-model, feature engineering strategy and target representation (random forest, handcrafted and engineered handcrafted, IoU scores) on all datasets. Points are colored blue by default, red is used for the top 10% of predicted configurations, magenta for the true top 10% and green for configurations that are both in the predicted and true top 10%.

5.5 Evaluation of the AutoML System

In the previous sections, the results of optimizing the different components of the AutoML system are presented. Based on these results the final AutoML system uses the best meta-learning pipeline: Random forest with max tree depth of 7, handcrafted and engineered handcrafted meta-features trained to predict raw IoU scores. Transfer learning and radiomics features are not used since they were found to not have a positive impact.

The workflow of the AIxCell AutoML system is illustrated in Figure 35. For a given dataset first the handcrafted and engineered handcrafted meta-features and the class distribution are extracted. Then in the initial configurator, the pipeline is initiated with the default values and the class-based (re-)sampling values are set by the automatic class-based sampling procedure. Then the meta-learning configurator predicts a ranking of pipeline configurations for the input dataset. Successive halving is then used to evaluate the top 50 pipeline configurations from the ranking, this is done with the compute budget set by the user.

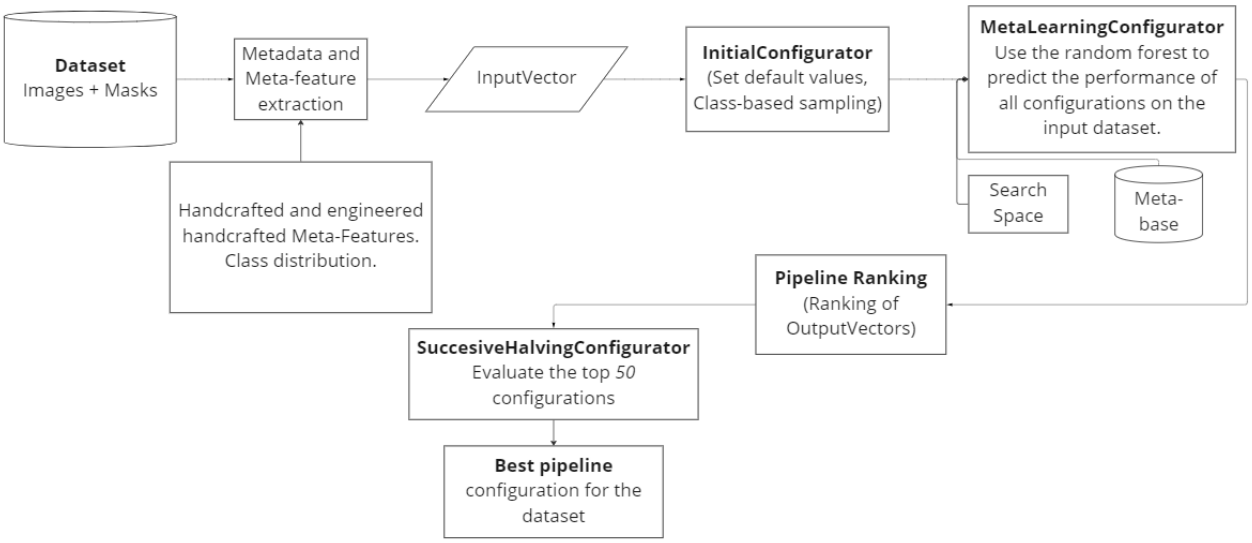


Figure 35: Overview over the final AIxCell AutoMl system

In this section, the results of the AIxCell AutoML system evaluations are presented. To this end, the AutoML system was run on four different datasets. The compute budgets and overall methodology are described in section 4.8. For each evaluated dataset the results are compared to an existing pipeline, constructed by human experts. Moreover, the performance of the best found pipeline configuration on a test set image is visualized.

5.5.1 Performance on the UKB dataset

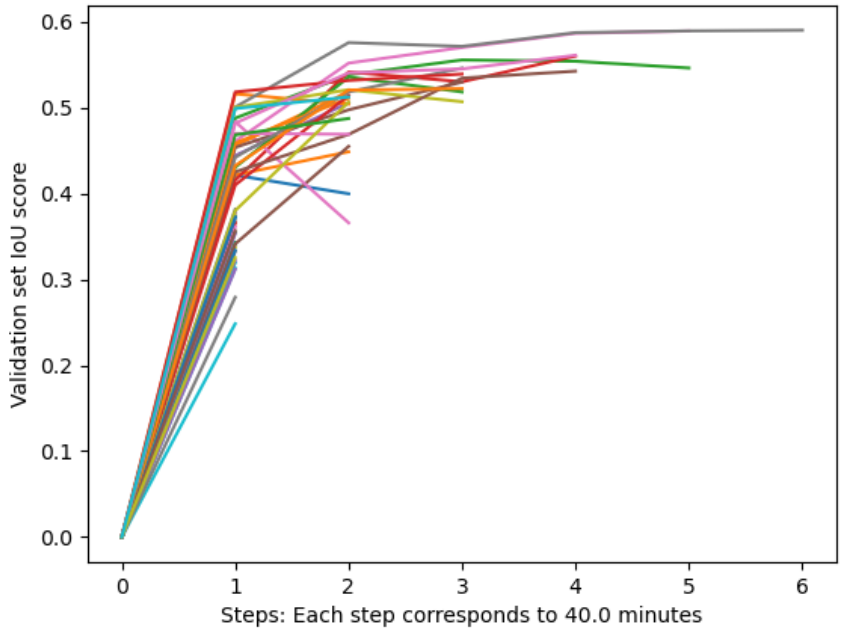
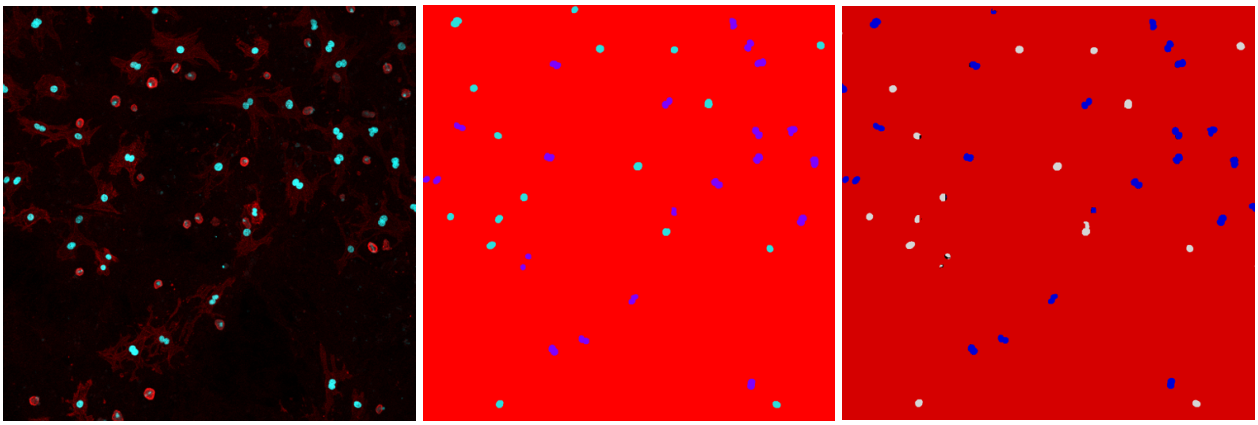


Figure 36: Successive halving on the UKB dataset



(a) Image (b) True mask (c) Predicted mask

Figure 37: Visual result of the best found pipeline configuration on the UKB dataset. Note that the color schemes are slightly different between the true and predicted mask. In the true mask the green color represents mononuclear instances, these are represented by white in the predicted mask. Blue always indicates binuclear instances.

The successive halving plot on the UKB dataset (see Figure 36) shows that the best pipeline already converges after five steps. In general, pipeline configurations converge quickly on this dataset. Moreover, there are two pipeline configurations that perform similarly well and the other top configurations perform worse. The following are the hyperparameter settings of the optimal pipeline configuration:

patch size = 128, learning rate = 0.0005, backbone = ResNet-18, augmentations = Flip and batch size = 4. This configuration was ranked as the 17^{th} best configuration by the meta-model. This pipeline configuration achieves a test set IoU score of ***0.60***. The existing pipeline configured in a previous project within AIxCell achieved a test set IoU score of ***0.64***. So the AIxCell AutoML system performs slightly worse.
The visual results (see Figure 37) show that the pipeline mostly segments the mononuclear and binuclear instances correctly. In some cases, binuclear instances are classified as mononuclear ones. For an example, see the top left of the predicted mask in Figure 37.

5.5.2 Performance on the iPSC dataset

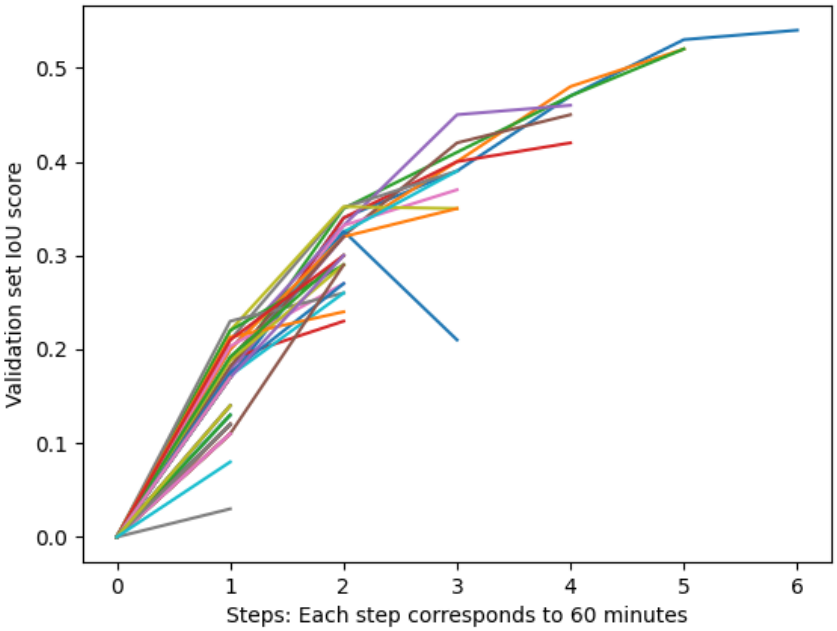
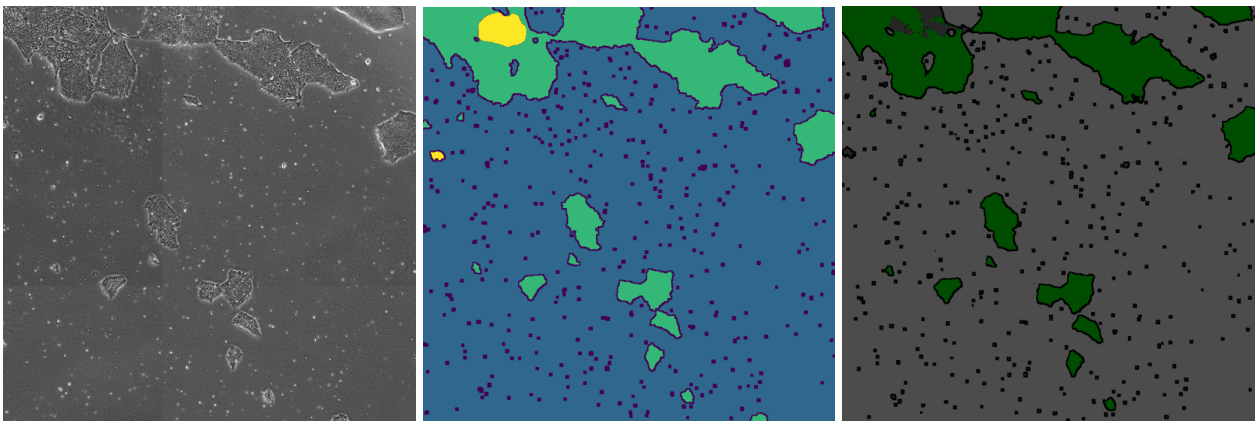


Figure 38: Successive halving on the iPSC dataset

The successive halving plot on the iPSC dataset (see Figure 38) shows that pipelines take significantly longer to converge, compared to the UKB dataset. Nonetheless, it there is no significant increase in performance from the fifth to the sixth step so it can be assumed that the optimal pipeline converged to optimal performance. Most configurations improve at every step, however, for one configuration there is a significant drop from the second to the third step.
The following are the hyperparameter settings of the optimal pipeline configuration: patch size = 256, learning rate = 0.0005, backbone = ResNet-18, augmentations = None and batch size = 8. This configuration was ranked as the 24^{th} best configuration by the meta-model. This pipeline configuration achieves a test set F1-score of ***0.6431***. The existing pipeline configured in a previous project within AIxCell achieved a test set IoU score of ***0.7344***. So the AIxCell AutoML system performs slightly worse. It has to be noted that for the iPSC dataset the test set performance significantly varies depending on what images are included in the test set. This is due to the dataset having six different classes which are unevenly distributed over the 40 images. If images that contain classes that only occur on

a minority of the images are in the test set, performance will be worse due to little training examples being available for these classes.
The visual results (see Figure 39) show that the pipeline detects all three classes in the image. However, especially the third class (yellow in true and dark grey in predicted) is not accurately segmented. So the pipeline detects the class but is unable to accurately segment it.



(a) Image (b) True mask (c) Predicted mask

Figure 39: Visual result of the best found pipeline configuration on the iPSC dataset. Note that the color schemes are slightly different between the true and predicted mask.

5.5.3 Performance on the Fluocells dataset

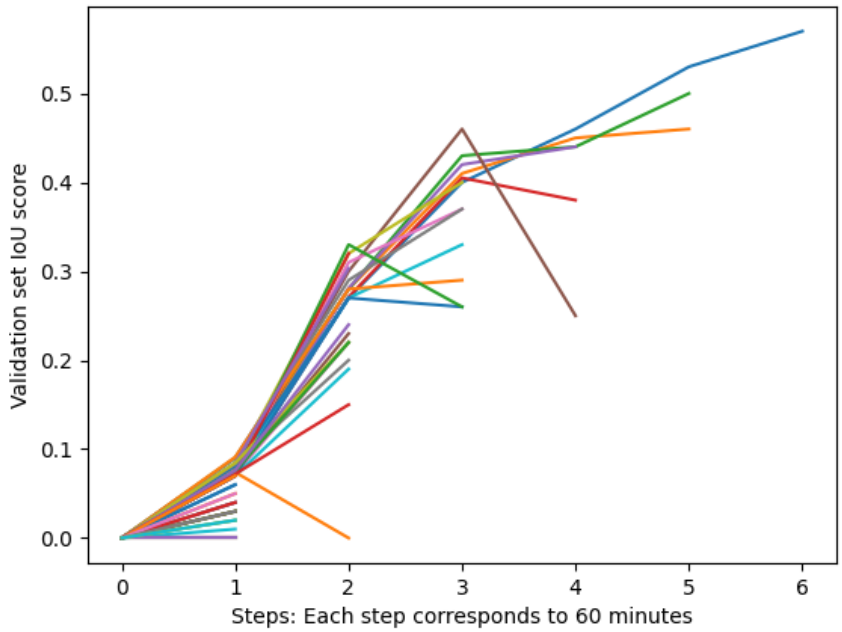


Figure 40: Successive halving on the Fluocells dataset

(a) Image (b) True mask (c) Predicted mask

Figure 41: Visual result of the best found pipeline configuration on the Fluocells dataset.

For the Fluocells dataset the successive halving plot (see Figure 40) shows that pipeline configurations take a long time to converge. Based on the plot it appears that this dataset requires the longest model training times. In the first step, configurations do not perform well as there is no configuration with a validation set IoU score higher then 0.1. In the next steps there are always significant differences between the configurations that are droped and continued. Furthermore, in the third and forth steps there are significant performance drops for some pipelines. The best pipeline configuration still improved from the fifth to the sixth step. It could therefore be that the best pipeline is not fully converged.
The following are the hyperparameter settings of the optimal pipeline configuration: patch size = 256, learning rate = 0.001, backbone = ResNet-50, augmentations = None and batch size = 16. This configuration was ranked as the 9^{th} best configuration by the meta-model. This pipeline configuration

achieves a test set F1-score of ***0.7621***. The best pipeline in a benchmarking of different configurations achieves a test set F1-Score of ***0.8149*** [83].
The visual results on a test set image show that the model accurately segments three neuronal cells (see Figure 41). However, there the model also segments two regions that are not labeled as neuronal cells in the true mask.

5.5.4 Performance on the EPFL electron microscopy dataset

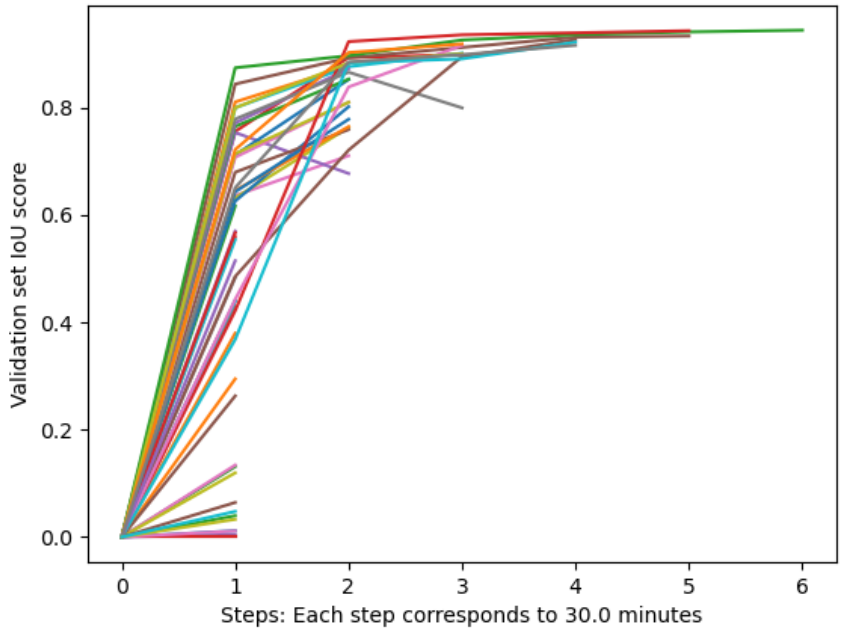
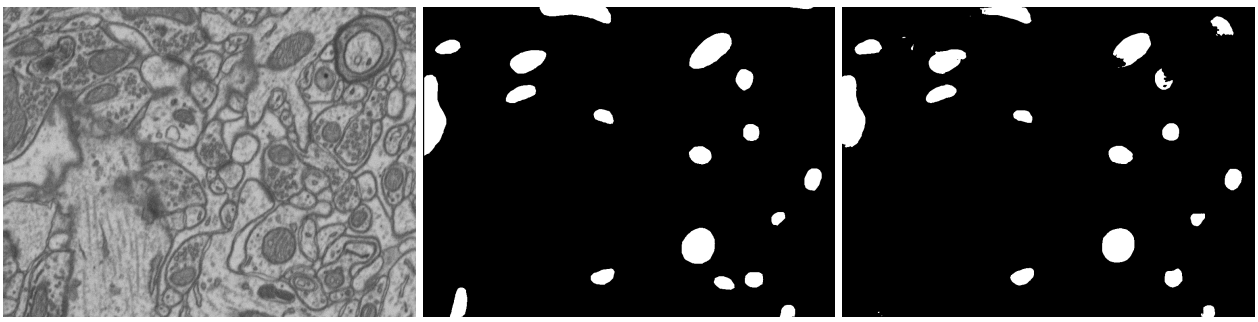


Figure 42: Successive halving on the EPFL electron microscopy dataset



(a) Image (b) True mask (c) Predicted mask

Figure 43: Visual result of the best found pipeline configuration on the EPFL electron microscopy dataset.

Based on the successive halving plot (see Figure 42 the electron microscopy dataset [84] appears to be the dataset that requires the least amount of time for a model to converge. The most notable differences between configurations occurs during the first step, here some configurations have IoU

scores below 0.2, while some others already converged to near optimal performance. During the last three steps the top configurations still improve, however, the performance gains are becoming smaller and smaller. The best configuration does not improve in the final step and it can therefore be assumed that the model converged to optimal performance.
The following are the hyperparameter settings of the optimal pipeline configuration: patch size = 256, learning rate = 0.0005, backbone = ResNet-18, augmentations = None and batch size = 8. This configuration was ranked as the 32^{nd} best configuration by the meta-model. This pipeline configuration achieves a test set IoU-score of ***0.7832***. The best known reported solution achieves a test set IoU score of ***0.8672*** [84].
The visual results from investigating one of the test images, show that the model accurately segments most regions (see Figure 43. Still, there are some regions that are not detected by the model and others that the model segments as a false positive.
Overall, pipeline configurations found by the AIxCell AutoML system are always worse compared to known solutions that were developed by human experts.

6 Discussion

In this section, the results of the experiments are discussed. First in section 6.1, the transfer learning results are discussed and compared to experiments from the literature. Secondly in section 6.2, the results of the meta-learning experiments are discussed. The focus here is on the meta-features and different meta-models. Finally, in section 6.3 the performance of the overall system is evaluated and the strength and weaknesses of the overall system are discussed.

6.1 Transfer Learning

The results of the transfer learning experiments indicate that transfer learning does not speed up model convergence time and does not increase model performance. Moreover, there was no significant difference between domain-specific and generic pre-training.
These results are in contrast to the results of previous studies that were presented in section 2.2. However, a review of the literature showed that the benefits of transfer learning were most prominent in classification tasks and weaker for segmentation tasks [26, 27, 30]. Furthermore, a study on biomedical image segmentation indicated that the effects of pre-training were mediated by source and target dataset similarity and target dataset quality [30]. In the presented research, both in-domain (cellpose and kaggle) and generic (ImageNet) datasets were tested. Nonetheless, the similarity between the in-domain source datasets and the target datasets can still be questioned. Both cellpose and kaggle are about binary segmentation, so the model learns to segment a certain object from the background. For cellpose, this is any object and for kaggle, these are the cell nuclei. In most of the target datasets (except UKK) the model is tasked with a multi-class segmentation problem. The segmentation task is, thereby, significantly different and the results might be due to this difference in tasks. In summary, even though the images in the datasets are from the same domain, the difference in what needs to be segmented could lead to pre-training not being beneficial.

Another potential explanation for the results is the quality of the target datasets. After division into patches, all target datasets contain a large number of images. Moreover, the images in all datasets are of good quality. Good quality here refers to no blurring or significant contrast differences due to image acquisition. Due to this relatively high quality of target datasets, the model might not benefit from pre-training because near optimal performance is reached in any case. An observation that stands in contrast to this conclusion is that on all datasets (except RUB) the model only achieves a peak validation set IoU score below 0.72.
Another study that investigated the effect of domain-specific and generic pre-training for segmentation tasks was performed by Cheng and Lam [85]. In their study, they pre-trained a U-Net model on the XPIE dataset [86], which contains 10,000 segmented natural images originally for salient object detection. They compared this model to using an encoder pre-trained on ImageNet and to a model trained from scratch. The target dataset contained images obtained using lung ultrasound and the task was to segment ribs. Comparably to the results of the presented study, they found that the model achieved similar IoU scores in all conditions. Still, they also performed a visual investigation of their results. Based on this investigation they found that, pre-training on the XPIE dataset outperformed both ImageNet pre-training and training from scratch. No visual investigation of the transfer learning results was performed in the presented research.

Due to the aforementioned reasons, the obtained results are not conclusive to determine the applicability of transfer learning to domain specific AutoDL systems. The results do, however, suggest that, for

semantic segmentation on diverse cellular image datasets, generic and domain-specific pre-training is not beneficial. Future research could focus on determining if further restricting the domain and taking the segmentation task into account leads to any benefits. In this way, not one pre-trained model would be used for all datasets in a domain but a different pre-trained model would be used based on dataset characteristics. This would, however, further increase the complexity of the domain specific AutoML system which might not be desirable.

6.2 Meta Learning

The results indicate that meta-learning is suitable to narrow down the search space to a few configurations that are likely to perform well. Handcrafted meta-features that describe the task in a dataset were shown to be the most important. Moreover, combining the handcrafted meta-features with the hyperparameters increased performance. A random forest model used to predict the raw IoU scores (pointwise approach) was best. Here limiting the maximal tree depth to seven further increased performance.

6.2.1 Meta Features

When visualized in two dimensions the radiomics vectors captured what images belong to which dataset, furthermore, the cosine similarity scores between the dataset level vectors were qualitatively good. Nonetheless, the top two meta-learning configurations did not use any representation of the radiomics scores. The radiomics similarity scores seem to be the most viable representation, as they were used in the third best configuration.
There are a few reasons that could explain why, even though visually plausible, the radiomics scores were not used by the best meta-learning configurations. First, the handcrafted meta-features and hyperparameters are more predictive and therefore most useful to the meta-model. Furthermore, the size of the meta-base is limited which could limit the number of features that can be effectively used by the meta-model [1]. The amount of datasets in the meta-base is also limited, if there would be more datasets this could increase the use of radiomics as a visual similarity measure. Secondly, it is not known how much the performance of a DL-pipeline is actually influenced by the visual features of a dataset. If information about the visual features does not have a large impact, this would decrease the value of the radiomics features.

It should, however, be noted that for many meta-models including the radiomics similarity scores and principal components, in combination with handcrafted and handcrafted engineered features, are the second or third best feature representation. Therefore, radiomics features are still promising for meta-learning based AutoML systems. Future research could investigate different ways to reduce the 95-dimensional vectors and investigate radiomics on a larger, more diverse meta-base.
The user given meta-features did not help the meta-model to estimate the performance of configurations. This is probably due to these features being diverse and there is little overlap between datasets. Especially the objects of interest were diverse, nuclei was the most common and only occurred in three out of eight datasets. Some objects only occur in one dataset which unfortunately makes them useless for predictions outside the training distribution. The microscopy technique seems to be the best user-given meta-feature since it gives information about the visual features in an image and the same technique is used for multiple datasets.

These observations suggest that, in order to effectively use user-given meta-features, it is important to design a small set of common features. This could be done together with a biomedical expert. Moreover, increasing the number of datasets in the meta-base would increase the applicability of user-given meta-features.
The best meta-features were the handcrafted ones; designed to extract information about the dataset and task that was deemed important. This confirms that features that are considered important by human experts are also important for meta-learning systems. Combining these features with the hyperparameter settings further increased performance. This shows the importance of combining 'static' dataset meta-features with hyperparameters to capture within dataset variance.
Overall, the investigated meta-features and feature engineering strategies significantly improved the performance of the meta-model. Some directions for future research have already been suggested. Furthermore, future research could investigate tools to automatically extract meta-features and engineer features. One such tool is MetaBU [87]. MetaBU takes a set of meta-features as input and then learns new meta-features via an Optimal Transport procedure, aligning the manually designed meta-features with the space of distributions on the hyper-parameter configurations [87]. This has been shown to increase the performance of state-of-the-art AutoML systems [41, 88] and could also improve the performance of the presented meta-model.

6.2.2 Meta Models

Evaluating different meta-models showed that more complex models significantly outperformed simple baseline models. Moreover, tree based models such as random forest and LamdaMART performed best. The ranking was most accurately predicted using the pointwise loss function, nonetheless, LamdaMART with the pairwise loss function also performed well. This is different to other learning-to-rank systems where no evaluations on different meta-models was presented and LamdaMART was used by default [46, 44]. Nonetheless, the size of the meta-base used in this study is very limited and larger meta-bases were used in the other studies. Therefore, it is possible that a pointwise approach is best for a small meta-base and that the pairwise approach is best for a larger meta-base.
An investigation of the tree is the random forest showed that the 'static' handcrafted meta-features are used for the splits close to the root and that the hyperparameters are used to split nodes near the leafs. Therefore, initially the model learns a baseline value and then this is refined based on the hyperparameters. Looking at the feature importances when predicting IoU scores on a validation dataset shows that the engineered meta-features are most important. This could be due to these combinations capturing both the baseline score of a dataset and the within-dataset variance.
While the results of the meta-learning experiments are good, they should be looked at with a bit of caution. Depending on what random seed is used, the model performance (measured by z-score) varies significantly. This is due to the nature of the z-score and the size of the meta-base. The z-score of the top 10% is easily influenced by a small number of errors in the predictions, since changing only a few of 10-30 points exerts a large influence. The variance over random seeds, shows that the model is not very robust in predicting the top 10%. Nonetheless, by averaging over random seeds more reliable estimates are obtained and it should be noted that the best model always performs well and there are no random seeds that lead to a large drop in performance.

6.3 The AutoML System

The best pipeline configurations that the AutoML system finds for the evaluation datasets are similar for some hyperparameters and more diverse for others. The only image augmentation that is used

in any of the top pipeline configurations is the flip transform for the UKB dataset. Moreover, the flip transform was the only image augmentation in the top 50 predicted pipeline configurations. Grid distortion or brightness and blurring were never used. An analysis of the metabase showed that these two image augmentations showed that there effect is highly dependent on what dataset they are used on. The brightness-blur transform decreases performance on all datasets, mean decrease of the validation IoU score of 0.031, except for iPSC and MSC where it increases performance, mean increase of 0.014. Similarly using the grid transform decreases performance for all datasets, by a factor of 0.023, except the BBBC020 and the RUB dataset, mean increase of 0.024. Due to the performance of configurations being influenced by all hyperparameters and the metabase only covering 30.8% of all possible configurations this analysis is not conclusive. Nonetheless, it indicates that what image augmentations can be used is dependent on the image dataset.
In order to know what image augmentations can be applied for a dataset, human experts look at the images and then try out different augmentations and evaluate them visually. Over time human experts then learn what augmentations can be applied to what type of image. In principle, the meta-model should also be able to learn this from the metabase. However, there is no optical information in the meta-feature representation used in the final system. Therefore, the model might just learn that overall the grid distortion and brightness+blur transform lead to worse performance. This could explain why no image augmentations are included in the top 50 pipeline configurations predicted by the meta-model.
For the other hyperparameters in the search space, a more diverse range of possible values are used by the best pipeline configurations. A patch size of 64 was not used, this could indicate that including this value in the search space does not add much value. Swapping a patch size of 64 for a larger value could bias the search space towards better configurations. Overall, the search space appears to be suitable to find an optimal pipeline configuration. Nonetheless, no analysis to determine what hyperparameter settings work on which datasets was conducted. As stated above, this is difficult due to the interaction effects between hyperparameter configurations.

Based on the results it appears that successive halving is suitable to evaluate a ranking of pipeline configurations. It appears crucial to use a large enough compute budget to ensure the convergence of the top pipeline configuration. The required time for a model to converge differs between datasets, based on the successive halving plot it appears that the Fluocells and iPSC dataset require significantly longer convergence times. Without prior experiments of training pipelines on a dataset, setting a compute budget that is sufficient, while not wasting unnecessary resources, is challenging. There are two possible solutions to this problem. First, before starting the successive halving process one of the pipelines could be trained until convergence and the compute budget could then be adjusted accordingly. This solution has the clear disadvantage that it induces a significant compute overhead. The second possible solution is to estimate convergence time based on the meta-features. For instance, it appears that the number of classes and instances in a dataset are correlated to convergence time, so simple rules could be designed to estimate the compute budget needed for successive halving.
Another notable observation in the plots of all datasets (see Figures 36 42 40 38), is that during the first step there is a large gap between continued configurations and stopped configurations. It is not clear whether that gap is due a worst performing pipeline or due to a better random initialization of the model weights. It has, however, been shown that models that perform worse early on also tend to converge to a lower value [3, 32, 70]. Therefore this might not be a problem for the applicability of successive halving.
Future research could compare the presented results of using successive halving to other multi-fidelity methods. Specifically predictive early stopping based on model learning curves, discussed in section

2.3.5, could further improve the overall AutoML system.

On all evaluated datasets the AIxCell AutoML system is outperformed by the DL-pipelines constructed by human experts. This was to be expected, since the other constructed pipelines are found within a much larger search space and have been found in a longer search process.
For the iPSC dataset the best found configuration has a test set IoU score of 0.54, this in the top 10% of configurations in the metabase, the distribution is shown in Figure 31. Moreover, looking at the predicted top 10% vs true top 10% plot 34, it can be seen that while this performance is within the true top 10% it is still far from the optimal performance of 0.64. So the AutoML system finds a configurations that falls within the top 10% but does not find the optimal solution. This could be due to the meta-model not including the best configuration in the top 50, or due to successive halving discarding the optimal configuration. A potential reason for why it is not included in the prediction of the meta-model is that the brightness/blur augmentation is not used. As discussed above, this augmentation is beneficial for the iPSC dataset but never included by the meta-model. Moreover, as stated in the results section the performance on the iPSC dataset is dependent on the selection of images for the testing set. It could, therefore, be that the best pipeline configuration in the metabase just has a favorable test set selection.
For the UKB dataset the best found configuration has a test set IoU score of 0.60, this again is in the top 10% of configurations in the metabase (see Figure 31). This is a surprisingly good performance, given that evaluating the meta-learning pipeline on the metabase revealed that for the UKB dataset none of the true top 10% configurations was predicted in the top 10% (see Figure 34). Nonetheless, the best pipeline configuration in the metabase is still better with a validation set IoU score of 0.64. This again might be due to test set selection, although the UKB dataset is more robust to this compared to the iPSC dataset. Overall, the performance on these two datasets that are in the metabase is good and the AutoML system reliably identifies a configuration in the true top 10%.
On the Fluocells [83] and EPFL electron microscopy [84] datasets the AutoML system is significantly outperformed by the published solutions [83, 84]. Nonetheless, an F1-Score of 0.7621 on Fluocells and an IoU-score of 0.7832 on EPFL electron microscopy can be considered good and the visual results confirm that the models are able to perform the segmentation task. Nonetheless, on both datasets the models have a tendency towards false positives. One thing that is missing in the evaluation of the AIxCell AutoML system is a comparison to a baseline solution. To this end, existing systems for automatic semantic segmentation such as Italisk [7] could be evaluated on the discussed datasets. A comparison to such a baseline would allow to more clearly asses the benefits of using AutoML to tune the hyperparameters of a cellular image segmentation system.

7 Conclusion

In this thesis an AutoML system that automatically configures a DL-pipeline for a given cellular image segmentation dataset was developed. The results show that the system reliably finds a pipeline configuration that is in the top 10% of all configurations in the search space. Nonetheless, the AutoML system performs worse then DL-pipelines developed by human experts. These DL-pipelines were not included in the search space of the AutoML system. This highlights the importance of designing a good search space that includes the optimal pipeline configurations.

The main component of the AutoML system is meta-learning, the experiments show that meta-learning can be used to limit the search space to pipeline configurations that are likely to perform well. To this end, classic handcrafted meta-features were found to be most important. Moreover, combinations of hyperparameter settings and these meta-features improved results. This limitation of the overall search space is reliable even though the meta-base only consists of 1.412 datapoints that cover 30.8% of the total search space. Overall, the presented study highlights the potential for meta-learning in domain-specific AutoML systems.

To quantify optical information of datasets, radiomics features were evaluated. These are not used in the best meta learning configuration. Nonetheless, results suggest that it might be important to include such meta-features in order to predict optimal image augmentations. No clear analysis of this observation was performed, it remains for future research to determine this effect. The results of the transfer learning experiments showed no effect of model pre-training. Furthermore, no difference between generic and domain-specific pre-training was found. This is inconsistent with the literature. Still, most existing studies focus on classification and it appears that transfer learning is less applicable to segmentation.

The presented AutoML system enables biomedical experts to utilize deep learning pipelines optimized for given datasets. No human expert is required for hyperparameter tuning. However, to achieve optimal performance a DL-expert is still required, as the presented system is reliably outperformed by DL-pipelines developed by DL-experts. Another downside of the presented system is that the biomedical expert is required to annotate images.
Overall, the presented system still increases the applicability of deep learning to cellular image analysis and enables cellular research. Moreover, the evaluated methodology is promising and can inform research on domain-specific AutoML system for image data.

Bibliography

[1] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[2] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. USA: Cambridge University Press, 2014.

[3] F. Hutter, L. Kotthoff, and J. Vanschoren, eds., *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

[4] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. Watson, *Molecular Biology of the Cell*. Garland, 4th ed., 2002.

[5] M. Boutros, F. Heigwer, and C. Laufer, "Microscopy-based high-content screening," *Cell*, vol. 163, no. 6, pp. 1314–1325, 2015.

[6] J. Xu, D. Zhou, D. Deng, J. Li, C. Chen, X. Liao, C. Guangyong, and P. Heng, "Deep learning in cell image analysis," *Intelligent Computing*, vol. 2022, pp. 1–15, 09 2022.

[7] S. Berg, D. Kutra, T. Kroeger, C. N. Straehle, B. X. Kausler, C. Haubold, M. Schiegg, J. Ales, T. Beier, M. Rudy, K. Eren, J. I. Cervantes, B. Xu, F. Beuttenmueller, A. Wolny, C. Zhang, U. Koethe, F. A. Hamprecht, and A. Kreshuk, "ilastik: interactive machine learning for (bio)image analysis," *Nature Methods*, Sept. 2019.

[8] T. Jones, I. Kang, D. Wheeler, R. Lindquist, A. Papallo, D. Sabatini, P. Golland, and A. Carpenter, "Cellprofiler analyst: Data exploration and analysis software for complex image-based screens," *BMC bioinformatics*, vol. 9, p. 482, 12 2008.

[9] R. S. Olson, W. L. Cava, Z. Mustahsan, A. Varik, and J. H. Moore, "Data-driven advice for applying machine learning to bioinformatics problems," *Biocomputing*, pp. 192–203, 2017.

[10] B. Zhang and P. Cao, "Classification of high dimensional biomedical data based on feature selection using redundant removal," *PLOS ONE*, vol. 14, pp. 1–19, 04 2019.

[11] J. Ruiz-Santaquiteria, G. Bueno, O. Deniz, N. Vallez, and G. Cristobal, "Semantic versus instance segmentation in microscopic algae detection," *Engineering Applications of Artificial Intelligence*, vol. 87, p. 103271, 2020.

[12] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[13] J. Roerdink and A. Meijster, "The watershed transform: Definitions, algorithms and parallelization strategies," *Fundamenta Informaticae*, vol. 41, pp. 187–228, 2000.

[14] N. Dhanachandra, K. Manglem, and Y. J. Chanu, "Image segmentation using k-means clustering algorithm and subtractive clustering algorithm," *Procedia Computer Science*, vol. 54, pp. 764–771, 2015.

[15] M. Kass, A. P. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, pp. 321–331, 2004.

[16] N. Plath, M. Toussaint, and S. Nakajima, "Multi-class image segmentation using conditional random fields and global classification," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, (New York, NY, USA), p. 817–824, Association for Computing Machinery, 2009.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[19] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," *CoRR*, vol. abs/1505.04366, 2015.

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

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

[22] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *CoRR*, vol. abs/2001.05566, 2020.

[23] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation policies from data," *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[24] O. Elharrouss, Y. Akbari, N. Almaadeed, and S. Al-ma'adeed, "Backbones-review: Feature extraction networks for deep learning and deep reinforcement learning approaches," *arXiv: Machine Learning*, 06 2022.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[26] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *CoRR*, vol. abs/1911.02685, 2019.

[27] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[29] M. Raghu, C. Zhang, J. M. Kleinberg, and S. Bengio, "Transfusion: Understanding transfer learning with applications to medical imaging," *CoRR*, vol. abs/1902.07208, 2019.

[30] D. Karimi, S. K. Warfield, and A. Gholipour, "Transfer learning in medical image segmentation: New insights from analysis of the dynamics of model parameters and learned representations," *Artif. Intell. Med.*, vol. 116, jun 2021.

[31] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms," *CoRR*, vol. abs/1208.3719, 2012.

[32] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *CoRR*, vol. abs/1908.00709, 2019.

[33] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyper-parameter values," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, 05 2016.

[34] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization* (C. A. C. Coello, ed.), pp. 507–523, Springer Berlin Heidelberg, 2011.

[35] J. Waring, C. Lindvall, and R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare," *Artificial Intelligence in Medicine*, vol. 104, p. 101822, 02 2020.

[36] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "OpenML: networked science in machine learning," *SIGKDD Explorations*, vol. 15, no. 2, pp. 49–60, 2013.

[37] B. Bilalli, A. Abelló, and T. Aluja-Banet, "On the predictive power of meta-features in openml," *International Journal of Applied Mathematics and Computer Science*, vol. 24, pp. 697–712, 08 2017.

[38] Q. Sun and B. Pfahringer, "Pairwise meta-rules for better meta-learning-based algorithm ranking," *Machine Learning*, vol. 93, 10 2013.

[39] J. Kim, S. Kim, and S. Choi, "Learning to warm-start bayesian hyperparameter optimization," *arXiv: Machine Learning*, 2017.

[40] M. Feurer, J. T. Springenberg, and F. Hutter, "Initializing bayesian hyperparameter optimization via meta-learning," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, p. 1128–1135, AAAI Press, 2015.

[41] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[42] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," *Proceedings of the 24th International Conference on Machine Learning*, 01 2007.

[43] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta, "Metalearning - applications to data mining.," *Cognitive Technologies. ISBN 978-3-540-73262-4. Springer Berlin Heidelberg*, 01 2009.

[44] F. Pinto, V. Cerqueira, C. Soares, and J. Moreira, "autobagging: Learning to rank bagging workflows with metalearning," *arXiv: Machine Learning*, 06 2017.

[45] F. Pinto, C. Soares, and J. Mendes-Moreira, "Towards automatic generation of metafeatures," in *PAKDD*, 2016.

[46] D. Laadan, R. Vainshtein, Y. Curiel, G. Katz, and L. Rokach, "Rankml: a meta learning-based approach for pre-ranking machine learning pipelines," *CoRR*, vol. abs/1911.00108, 2019.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[48] W. Konen, P. Koch, O. Flasch, T. Bartz-Beielstein, M. Echtenbruck, and B. Naujoks, "Tuned data mining: A benchmark study on different tuners," GECCO, Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, 2011.

[49] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *IJCAI*, 2015.

[50] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[51] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," *Technical report, University of Toronto*, 2009.

[52] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with bayesian neural networks," in *ICLR*, 2017.

[53] J. C. Gittins, *Multi-armed Bandit Allocation Indices*. Wiley, Chichester, NY, 1989.

[54] D. A. Berry and B. Fristedt, *Bandit Problems: Sequential Allocation of Experiments*. Springer, October 1985.

[55] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (A. Gretton and C. C. Robert, eds.), vol. 51 of *Proceedings of Machine Learning Research*, (Cadiz, Spain), pp. 240–248, PMLR, 09–11 May 2016.

[56] J. J. van Griethuysen, A. Fedorov, C. Parmar, A. Hosny, N. Aucoin, V. Narayan, R. G. Beets-Tan, J.-C. Fillion-Robin, S. Pieper, and H. J. Aerts, "Computational Radiomics System to Decode the Radiographic Phenotype," *Cancer Research*, vol. 77, pp. e104–e107, 10 2017.

[57] P. Lambin, R. Leijenaar, T. Deist, J. Peerlings, E. de Jong, J. Van Timmeren, S. Sanduleanu, R. Larue, A. Even, A. Jochems, Y. Wijk, H. Woodruff, J. Soest, T. Lustberg, E. Roelofs, W. Elmpt, A. Dekker, F. Mottaghy, J. Wildberger, and S. Walsh, "Radiomics: The bridge between medical imaging and personalized medicine," *Nature Reviews Clinical Oncology*, vol. 14, 10 2017.

[58] S. Gitto, R. Cuocolo, A. Annovazzi, V. Anelli, M. Acquasanta, A. Cincotta, D. Albano, V. Chianca, V. Ferraresi, C. Messina, C. Zoccali, E. Armiraglio, A. Parafioriti, R. Sciuto, A. Luzzati, R. Biagini, M. Imbriaco, and L. Sconfienza, "Ct radiomics-based machine learning classification of atypical cartilaginous tumours and appendicular chondrosarcomas," *EBioMedicine*, vol. 68, p. 103407, 06 2021.

[59] R. Laajili, M. Said, and M. Tagina, "Application of radiomics features selection and classification algorithms for medical imaging decision: Mri radiomics breast cancer cases study," *Informatics in Medicine Unlocked*, vol. 27, p. 100801, 2021.

[60] J. Van Timmeren, D. Cester, S. Tanadini-Lang, H. Alkadhi, and B. Baeßler, "Radiomics in medical imaging—"how-to" guide and critical reflection," *Insights into Imaging*, vol. 11, 12 2020.

[61] M. P. A. Starmans, S. R. van der Voort, T. Phil, M. J. M. Timbergen, M. Vos, G. A. Padmos, W. Kessels, D. Hanff, D. J. Grunhagen, C. Verhoef, S. Sleijfer, M. J. v. d. Bent, M. Smits, R. S. Dwarkasing, C. J. Els, F. Fiduzi, G. J. L. H. van Leenders, A. Blazevic, J. Hofland, T. Brabander, R. A. H. van Gils, G. J. H. Franssen, R. A. Feelders, W. W. de Herder, F. E. Buisman, F. E. J. A. Willemssen, B. G. Koerkamp, L. Angus, A. A. M. van der Veldt, A. Rajicic, A. E. Odink, M. Deen, J. M. C. T., J. Veenland, I. Schoots, M. Renckens, M. Doukas, R. A. de Man, J. N. M. IJzermans, R. L. Miclea, P. B. Vermeulen, E. E. Bron, M. G. Thomeer, J. J. Visser, W. J. Niessen, and S. Klein, "Reproducible radiomics through automated machine learning validated on twelve clinical applications," *arXiv: Machine Learning*, 2021.

[62] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, "AutoGluon-tabular: Robust and accurate automl for structured data," *arXiv: Machine Learning*, 2020.

[63] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1946–1956, ACM, 2019.

[64] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "Medmnist v2: A large-scale lightweight benchmark for 2d and 3d biomedical image classification," *CoRR*, vol. abs/2110.14795, 2021.

[65] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *CoRR*, vol. abs/1703.03400, 2017.

[66] C. Stringer, M. Michaelos, and M. Pachitariu, "Cellpose: a generalist algorithm for cellular segmentation," *Nature Methods*, 2020.

[67] L. H. H. S. B. W. Y. . A. Yu, W., "Cell image library," *Neuroblastoma*, 2022.

[68] U. Schmidt, M. Weigert, C. Broaddus, and G. Myers, "Cell detection with star-convex polygons," in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018* (A. F. Frangi, J. A. Schnabel, C. Davatzikos, C. Alberola-López, and G. Fichtinger, eds.), (Cham), pp. 265–273, Springer International Publishing, 2018.

[69] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[70] L. Zimmer, M. Lindauer, and F. Hutter, "Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl," *CoRR*, vol. abs/2006.13799, 2020.

[71] X. Dong and Y. Yang, "Nas-bench-201: Extending the scope of reproducible neural architecture search," *CoRR*, vol. abs/2001.00326, 2020.

[72] J. Caicedo, A. Goodman, K. Karhohs, B. Cimini, J. Ackerman, M. Haghighi, C. Heng, T. Becker, M. Doan, C. McQuin, M. H. Rohban, S. Singh, and A. Carpenter, "Nucleus segmentation across imaging experiments: the 2018 data science bowl," *Nature Methods*, vol. 16, 12 2019.

[73] P. Naylor, M. Laé, F. Reyal, and T. Walter, "Segmentation of nuclei in histopathology images by deep regression of the distance map," *IEEE Transactions on Medical Imaging*, vol. 38, pp. 1–1, 08 2018.

[74] N. Kumar, R. Verma, S. Sharma, S. Bhargava, A. Vahadane, and A. Sethi, "A dataset and a technique for generalized nuclear segmentation for computational pathology," *IEEE Transactions on Medical Imaging*, vol. 36, pp. 1–1, 03 2017.

[75] J. Shu, *Immunohistochemistry image analysis : protein, nuclei and gland*. PhD thesis, University of Nottingham, UK, 2014.

[76] V. Ljosa, K. Sokolnicki, and A. Carpenter, "Annotated high-throughput microscopy image sets for validation," *Nature methods*, vol. 9, p. 637, 06 2012.

[77] L. P. Coelho, A. Shariff, and R. Murphy, "Nuclear segmentation in microscope cell images: A hand-segmented dataset and comparison of algorithms," *Proceedings / IEEE International Symposium on Biomedical Imaging: from nano to macro. IEEE International Symposium on Biomedical Imaging*, vol. 5193098, pp. 518–521, 06 2009.

[78] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017.

[79] T. Nguyen, T. Özaslan, I. D. Miller, J. Keller, G. Loianno, C. J. Taylor, D. D. Lee, V. Kumar, J. H. Harwood, and J. M. Wozencraft, "U-net for mav-based penstock inspection: an investigation of focal loss in multi-class segmentation for corrosion identification," *CoRR*, vol. abs/1809.06576, 2018.

[80] L. van der Maaten and G. Hinton, "Viualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 11 2008.

[81] I. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, p. 20150202, 04 2016.

[82] C. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, vol. 11, 01 2010.

[83] L. Clissa, R. Morelli, F. Squarcio, T. Hitrec, M. Luppi, L. Rinaldi, M. Cerri, R. Amici, S. Bastianini, C. Berteotti, V. L. Martire, D. Martelli, A. Occhinegro, D. Tupone, G. Zoccoli, and A. Zoccoli, "Fluorescent neuronal cells," *Dataset provided by University of Bologna,* `http://amsacta.unibo.it/6706/`, 2021.

[84] A. Lucchi, Y. Li, and P. Fua, "Learning for structured prediction using approximate subgradient descent with working sets," *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1987–1994, 06 2013.

[85] D. Cheng and E. Y. Lam, "Transfer learning u-net deep learning for lung ultrasound segmentation," *arXiv: Machine Learning*, 2021.

[86] C. Xia, J. Li, X. Chen, A. Zheng, and Y. Zhang, "What is and what is not a salient object? learning salient object detector by ensembling linear exemplar regressors," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4399–4407, 2017.

[87] H. Rakotoarison, L. Milijaona, A. RASOANAIVO, M. Sebag, and M. Schoenauer, "Learning meta-features for autoML," in *International Conference on Learning Representations*, 2022.

[88] N. Fusi, R. Sheth, and M. Elibol, "Probabilistic matrix factorization for automated machine learning," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

Appendices

A Meta Learning Results

Table 3: Results using Decison Tree as the meta-model to predict raw IoU scores

	RMSE	Kendall Tau	Diff IoU	Z-score	Pearson
All meta-features	0.210	0.010	0.041	0.308	0.135
Only hyperparameters	0.189	0.034	0.053	0.015	0.017
Handcrafted and engineered handcrafted	0.183	0.022	0.041	0.346	0.200
Radiomics similarity scores	0.189	0.007	0.045	0.320	0.134
Handcrafted, rad sim, eng handcrafted,radiomics	0.190	0.010	0.042	0.263	0.149
Handcrafted, rad PCA, eng handcrafted,radiomics	0.286	0.004	0.042	0.280	0.121
Only engineered handcrafted	0.194	0.021	0.046	0.171	0.034
Handcrafted and radiomics PCA	0.212	0.025	0.036	0.492	0.145
Handcrafted and radiomics similarity	0.199	0.018	0.039	0.419	0.131
Only Handcrafted	0.202	0.012	0.033	0.542	0.127
Handcrafted, eng handcrafted, radiomics PCA	0.186	-0.001	0.043	0.287	0.152
Handcrafted, eng handcrafted, radiomics sim	0.189	0.003	0.042	0.289	0.161
Handcrafted and user-given	0.197	0.005	0.035	0.483	0.152
Only entire radiomics vectors	0.255	0.002	0.042	0.341	0.116

Table 4: Results using Linear Regressor as the meta-model to predict raw IoU scores

	RMSE	Kendall Tau	Diff IoU	Z-score	Pearson
All meta-features	1.077	-0.048	0.060	-0.043	0.031
Only hyperparameters	0.151	0.017	0.050	0.173	0.057
Handcrafted and engineered handcrafted	4.203	-0.004	0.052	0.118	0.124
Radiomics similarity scores	0.200	0.016	0.059	0.041	0.010
Handcrafted, rad sim, eng handcrafted,radiomics	0.212	0.011	0.063	-0.148	0.068
Handcrafted, rad PCA, eng handcrafted,radiomics	1.336	0.010	0.048	0.183	0.076
Only engineered handcrafted	0.147	0.019	0.042	0.303	0.146
Handcrafted and radiomics PCA	0.577	0.016	0.059	0.041	0.010
Handcrafted and radiomics similarity	0.577	0.016	0.059	0.041	0.010
Only Handcrafted	0.577	0.016	0.059	0.041	0.010
Handcrafted, eng handcrafted, radiomics PCA	4.195	-0.004	0.052	0.118	0.124
Handcrafted, eng handcrafted, radiomics sim	4.202	-0.004	0.052	0.118	0.124
Handcrafted and user-given	0.579	0.016	0.059	0.041	0.010
Only entire radiomics vectors	0.577	0.016	0.059	0.041	0.010

Table 5: Results using SVM as the meta-model to predict raw IoU scores

	RMSE	Kendall Tau	Diff IoU	Z-score	Pearson
All meta-features	0.153	0.036	0.057	-0.048	0.187
Only hyperparameters	0.152	0.039	0.059	-0.205	0.039
Handcrafted and engineered handcrafted	0.314	-0.009	0.036	0.352	0.258
Radiomics similarity scores	0.153	0.044	0.058	-0.195	0.035
Handcrafted, rad sim, eng handcrafted,radiomics	0.153	0.037	0.057	-0.047	0.187
Handcrafted, rad PCA, eng handcrafted,radiomics	0.153	0.038	0.057	-0.041	0.186
Only engineered handcrafted	0.153	0.031	0.057	-0.048	0.187
Handcrafted and radiomics PCA	0.314	-0.015	0.052	-0.060	0.117
Handcrafted and radiomics similarity	0.314	-0.011	0.052	-0.065	0.117
Only Handcrafted	0.314	-0.016	0.052	-0.065	0.116
Handcrafted, eng handcrafted, radiomics PCA	0.314	-0.002	0.036	0.350	0.260
Handcrafted, eng handcrafted, radiomics sim	0.315	-0.003	0.037	0.329	0.259
Handcrafted and user-given	0.315	-0.013	0.052	-0.062	0.117
Only entire radiomics vectors	0.315	-0.012	0.052	-0.062	0.116

Table 6: Results using XGBoost as the meta-model to predict raw IoU scores

	RMSE	Kendall Tau	Diff IoU	Z-score	Pearson
All meta-features	0.157	0.006	0.044	0.224	0.173
Only hyperparameters	0.181	0.025	0.057	-0.062	0.013
Handcrafted and engineered handcrafted	0.136	0.004	0.034	0.479	0.320
Radiomics similarity scores	0.167	0.013	0.050	0.234	0.111
Handcrafted, rad sim, eng handcrafted,radiomics	0.159	0.004	0.047	0.193	0.168
Handcrafted, rad PCA, eng handcrafted,radiomics	0.263	-0.008	0.043	0.235	0.106
Only engineered handcrafted	0.166	0.006	0.052	-0.053	0.067
Handcrafted and radiomics PCA	0.165	0.001	0.039	0.446	0.255
Handcrafted and radiomics similarity	0.194	-0.005	0.037	0.464	0.239
Only Handcrafted	0.165	0.011	0.039	0.452	0.264
Handcrafted, eng handcrafted, radiomics PCA	0.138	0.002	0.034	0.482	0.311
Handcrafted, eng handcrafted, radiomics sim	0.171	-0.001	0.033	0.491	0.308
Handcrafted and user-given	0.156	-0.001	0.037	0.493	0.272
Only entire radiomics vectors	0.262	0.001	0.040	0.430	0.211

Table 7: Results using Random Forest as the meta-model to predict IoU scores

	RMSE	Kendall Tau	Diff IoU	Z-score	Pearson
All meta-features	0.181	0.013	0.041	0.294	0.244
Only hyperparameters	0.175	0.026	0.052	0.024	0.021
Handcrafted and engineered handcrafted	0.138	0.007	**0.028**	**0.643**	0.295
Radiomics similarity scores	0.173	0.000	0.043	0.374	0.173
Handcrafted, rad sim, eng handcrafted,radiomics	0.166	0.021	0.042	0.229	0.227
Handcrafted, rad PCA, eng handcrafted,radiomics	0.259	-0.010	0.041	0.252	0.123
Only engineered handcrafted	0.160	0.001	0.050	0.009	0.002
Handcrafted and radiomics PCA	0.151	-0.014	0.032	0.522	0.230
Handcrafted and radiomics similarity	0.164	-0.003	0.035	0.516	0.237
Only Handcrafted	0.153	-0.002	0.030	0.582	0.242
Handcrafted, eng handcrafted, radiomics PCA	0.146	-0.001	0.031	0.553	0.298
Handcrafted, eng handcrafted, radiomics sim	0.157	0.005	0.030	0.583	**0.308**
Handcrafted and user-given	0.155	-0.012	0.030	0.570	0.276
Only entire radiomics vectors	0.228	0.012	0.039	0.407	0.264

Table 8: Results using LamdaMART as the meta-model to predict a ranking based on IoU scores, with colsample bytree and subsample set to 0.9. Results averaged over 20 random seeds

	Kendall Tau	Diff IoU	Z-score	Pearson
All meta-features	-0.009	0.037	0.403	0.273
Only hyperparameters	0.010	0.052	0.087	0.129
Handcrafted and engineered handcrafted	0.000	0.032	0.486	0.302
Radiomics similarity scores	0.014	0.048	0.267	0.165
Handcrafted, rad sim, eng handcrafted,radiomics	-0.011	0.037	0.406	0.278
Handcrafted, rad PCA, eng handcrafted,radiomics	-0.015	0.037	0.408	0.281
Only engineered handcrafted	-0.004	0.034	0.427	0.290
Handcrafted and radiomics PCA	-0.005	0.034	0.523	0.298
Handcrafted and radiomics similarity	-0.002	0.034	0.531	**0.317**
Only Handcrafted	-0.002	**0.032**	**0.556**	0.289
Handcrafted, eng handcrafted, radiomics PCA	0.001	0.032	0.520	0.307
Handcrafted, eng handcrafted, radiomics sim	-0.002	0.033	0.491	**0.317**
Handcrafted and user-given	-0.001	0.034	0.521	0.290
Only entire radiomics vectors	-0.002	0.036	0.504	0.307

Table 9: Results using LamdaMART as the meta-model to predict a ranking based on IoU scores. Here LamdaMART is deterministic and always uses all datapoints and features.

	Kendall Tau	Diff IoU	Z-score	Pearson
All meta-features	-0.016	0.036	0.398	0.264
Only hyperparameters	0.010	0.053	0.058	0.122
Handcrafted and engineered handcrafted	-0.012	0.036	0.371	0.291
Radiomics similarity scores	0.018	0.051	0.219	0.163
Handcrafted, rad sim, eng handcrafted,radiomics	-0.008	0.037	0.437	0.286
Handcrafted, rad PCA, eng handcrafted,radiomics	-0.009	0.033	0.529	0.269
Only engineered handcrafted	-0.008	0.033	0.434	0.292
Handcrafted and radiomics PCA	0.025	0.032	0.567	0.315
Handcrafted and radiomics similarity	0.036	0.036	0.507	**0.346**
Only Handcrafted	-0.003	0.030	**0.608**	0.299
Handcrafted, eng handcrafted, radiomics PCA	-0.012	0.031	0.516	0.318
Handcrafted, eng handcrafted, radiomics sim	0.008	**0.028**	0.606	0.291
Handcrafted and user-given	0.002	0.031	0.552	0.279
Only entire radiomics vectors	0.022	0.036	0.530	0.323

Table 10: Results using Random Forest as the meta-model to predict binned IoU scores. Results are averaged over 20 random seeds.

	RMSE	Kendall Tau	Diff IoU	Z-score	Pearson
All meta-features	2.887	0.006	0.037	0.411	0.317
Only hyperparameters	3.256	0.023	0.048	0.183	0.130
Handcrafted and engineered handcrafted	2.878	0.004	0.033	0.479	0.351
Radiomics similarity scores	2.950	0.025	0.040	0.416	0.264
Handcrafted, rad sim, eng handcrafted,radiomics	2.906	0.008	0.040	0.364	0.296
Handcrafted, rad PCA, eng handcrafted,radiomics	2.935	0.004	0.036	0.415	0.322
Only engineered handcrafted	2.925	-0.001	0.032	0.509	0.335
Handcrafted and radiomics PCA	2.780	0.006	0.035	0.491	0.359
Handcrafted and radiomics similarity	2.747	0.025	0.037	0.476	**0.361**
Only Handcrafted	2.829	0.010	**0.029**	**0.601**	0.346
Handcrafted, eng handcrafted, radiomics PCA	2.893	0.010	0.035	0.448	0.341
Handcrafted, eng handcrafted, radiomics sim	2.868	0.019	0.036	0.444	0.346
Handcrafted and user-given	2.815	0.005	0.031	0.543	0.336
Only entire radiomics vectors	2.809	0.022	0.037	0.488	0.317

Table 11: Results using LamdaMART as the meta-model to predict binned IoU scores, with colsample bytree and subsample set to 0.9. Results averaged over 20 random seeds

	Kendall Tau	Diff IoU	Z-score	Pearson
All meta-features	0.006	0.034	0.477	0.333
Only hyperparameters	0.016	0.052	0.081	0.159
Handcrafted and engineered handcrafted	0.007	0.032	0.525	0.358
Radiomics similarity scores	0.021	0.047	0.277	0.203
Handcrafted, rad sim, eng handcrafted,radiomics	0.009	0.035	0.454	0.320
Handcrafted, rad PCA, eng handcrafted,radiomics	0.004	0.035	0.455	0.338
Only engineered handcrafted	0.013	0.033	0.462	0.329
Handcrafted and radiomics PCA	0.008	0.034	0.543	0.359
Handcrafted and radiomics similarity	0.008	0.035	0.498	**0.372**
Only Handcrafted	0.004	0.033	0.544	0.347
Handcrafted, eng handcrafted, radiomics PCA	0.006	**0.031**	0.534	0.353
Handcrafted, eng handcrafted, radiomics sim	0.008	0.033	0.509	0.354
Handcrafted and user-given	0.006	0.033	**0.551**	0.341
Only entire radiomics vectors	0.008	0.036	0.520	0.357

Table 12: Results using LamdaMART as the meta-model to predict binned IoU scores. Here LamdaMART is deterministic and always uses all data points and features.

	Kendall Tau	Diff IoU	Z-score	Pearson
All meta-features	-0.034	0.034	0.463	0.352
Only hyperparameters	0.014	0.053	0.051	0.157
Handcrafted and engineered handcrafted	0.016	**0.030**	0.523	0.353
Radiomics similarity scores	0.014	0.043	0.363	0.205
Handcrafted, rad sim, eng handcrafted,radiomics	0.004	0.034	0.412	0.321
Handcrafted, rad PCA, eng handcrafted,radiomics	0.005	0.039	0.377	0.298
Only engineered handcrafted	0.007	0.029	0.571	0.366
Handcrafted and radiomics PCA	0.016	0.033	0.524	0.357
Handcrafted and radiomics similarity	0.048	0.040	0.450	**0.368**
Only Handcrafted	0.021	0.034	0.540	0.341
Handcrafted, eng handcrafted, radiomics PCA	0.001	0.038	0.472	0.346
Handcrafted, eng handcrafted, radiomics sim	0.013	0.033	0.517	0.359
Handcrafted and user-given	0.005	0.031	**0.583**	0.365
Only entire radiomics vectors	-0.014	0.041	0.400	0.356