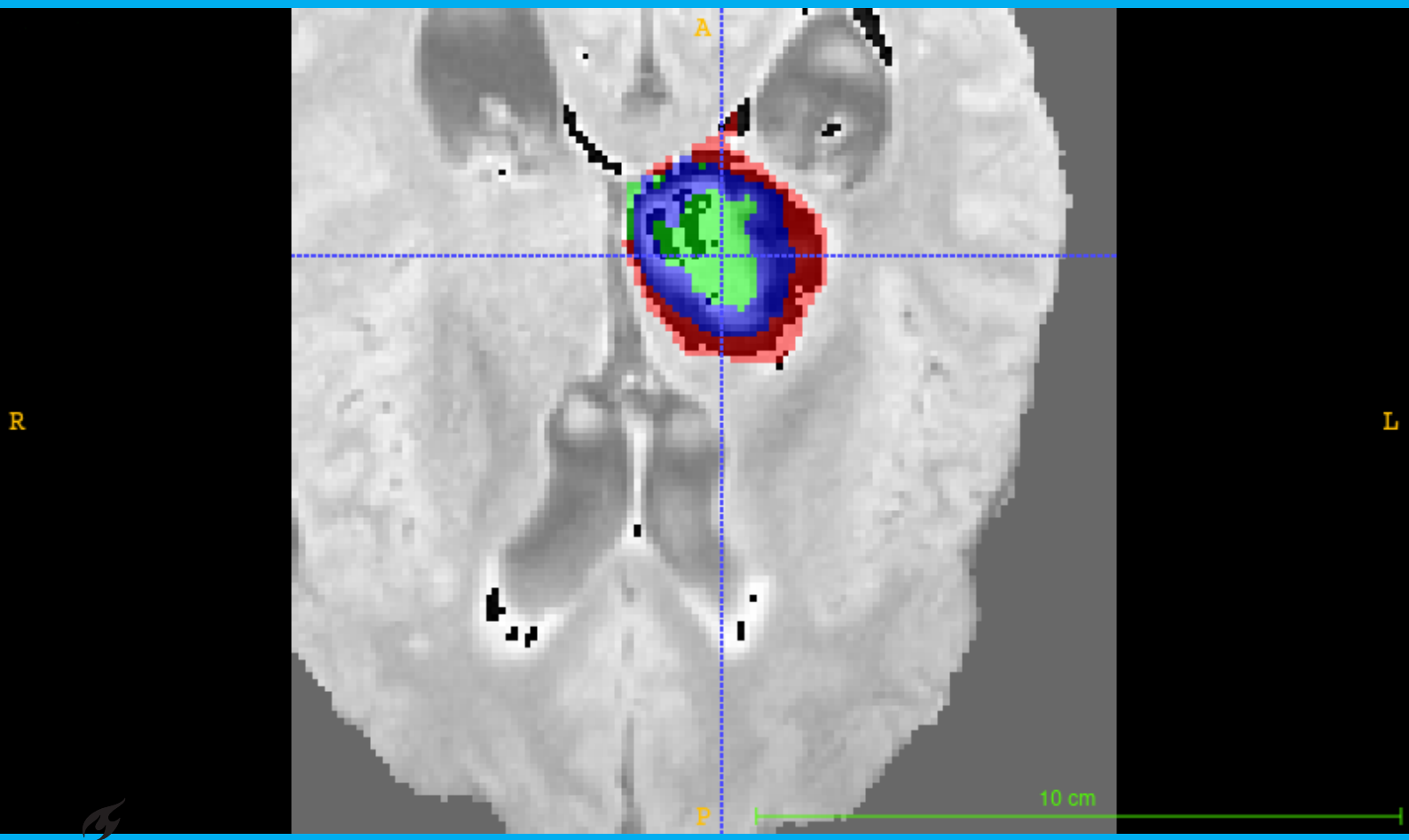
Automated Machine Learning in Medical Image Segmentation

C.S.O. van Gruijthuijsen



TU Delft

Automated Machine Learning in Medical Image Segmentation

by

C.S.O. van Gruijthuijsen

Delft University of Technology,

Student number:	4353528	
Supervisors:	Dr. W. Pan,	TU Delft
	Ir. M. Starmans,	Erasmus MC
	Ir. K. van Garderen,	Erasmus MC
	Dr. ir. S. Klein,	Erasmus MC

TU Delft

Abstract

Semantic segmentation of medical imaging delivers important insights to physicians. In recent years, the application of machine learning models in this field has increased. However, the design with the best performance of these machine learning models differs for each task and the design process is difficult and time-consuming. Automated Machine Learning (AutoML) aims to automate this process by selecting the algorithms and optimizing their hyperparameters for each task. In this thesis, an AutoML approach was applied to medical image segmentation, using Bayesian Optimization with HyperBand (BOHB) to optimize hyperparameters of U-net, a commonly used design. In our approach, we also optimize several hyperparameters used in preprocessing and the training procedure. To achieve that, we investigated the effects of hyperparameters on the performance, compared BOHB to Random Search (RS) and the overall approach was compared to the state-of-the-art, all tasked to segment brain tumors in a single dataset.

The results show that minimally trained configurations already significantly predict the best performing configurations, a necessary prerequisite for BOHB. Furthermore, we show that BOHB does not significantly perform differently than RS when applied to a small search space, stressing the need of matching the search space to the search strategy. We also show that several hyperparameters significantly impact the performance for this task, whereas others do not impact the performance directly.

However, this approach does not yet outperform the state-of-the-art, due to several differences, most importantly in data augmentation. Further research can use our approach on other datasets, include hyperparameters that were outside our scope, or optimize hyperparameters for other objectives, such as computational resources or explainability.

Preface

This thesis documents the research I conducted to conclude my MSc Mechanical Engineering at Delft University of Technology. It describes the work I performed during a 10-month project. This project was a collaboration with the Radiology en Medical informatics departments of the Erasmus Medical Center, on the application of Automated Machine Learning in medical image segmentation.

First of all, I would like to thank both Martijn en Karin, for their daily supervision of my project. It was amazing to work together with you, both on a personal level and academically. I have been amazed by your expertise and eye for detail. Our meetings where I could ask and discuss anything were great, and invaluable for my project. I have been inspired by your unfailing enthusiasm and positive attitude.

Second, I would like to thank Stefan for his part in my supervision. Your questions and comments encouraged and challenged me to bring my work to the next level, for which I am grateful.

Finally, I would like to thank Wei for being my TU Delft supervisor. You have enabled me to pursue a project that I thought and think is very interesting, even though it is not common to find projects outside the TU Delft for my specialization. You have given me a lot of freedom and trust in organizing my own project, which turned out to be something I enjoyed.

I feel very thankful to work with such knowledgeable, enthusiastic and friendly people on this project!

Coen van Gruijthuijsen
Delft, October 2021

Contents

1

Introduction

Medical images, such as made by techniques such as Magnetic Resonance Imaging (MRI), Computed Tomography (CT), and Positron Emission Tomography (PET), deliver insights to medical specialists with minimal risks for the patient. One of the techniques to analyze these images is semantic segmentation, i.e. classifying all voxels (3D-pixels) to specific classes, such as organs or tumors. An example is given in Fig. 1.1. There are several examples where semantic segmentation can be used, one of them is calculating cardiac parameters such as chamber volume or ventricle mass [55], part of the gold standard for treating cardiovascular diseases. Another clinical application is analyzing the (changes in) volume of a brain tumor [26]. Currently, tumor volumes are calculated using a measurement of two perpendicular diameters. This can only be used as a rough estimation of the volume, since tumors are not a sphere nor even convex. Using segmented images, more accurate results are possible. Unfortunately, this is typically too time-consuming to be performed manually in a clinical setting [5]. Machine learning can be used to automate the creation of these segmentations. In particular, deep learning methods such as convolutional neural networks are used extensively in recent research [1], for example U-net [49], OBELISK [20] or one of the many variants of U-net such as 3D-Unet [8] and V-Net [40] (See Chapter 2).

However, the design of neural networks has its own challenges. The most optimal design of a network depends on the problem. For example, the size of the training set, the clinical application and the size of the objects that have to be segmented all influence this design [2]. Furthermore, the performance cannot be easily predicted [24], which makes the development of these networks a time-consuming and trial-and-error process. This is further complicated by the fact that there is no clear consensus which basic architecture is optimal [20] [49].

The same holds for the complete pipeline: different pre- and postprocessing elements, such as resampling and normalization of the input images, but also training elements such as the loss function, have to be optimized for each problem [24]. These steps play a substantial role in the eventual performance and therefore need to be designed or adapted to the problem as well. For example, a previous study showed that a better performance is reached when using percentile clipping normalization for CT-scans, whereas designs for MRI scans z-scoring is preferred [25].

Automated Machine Learning (AutoML) tries to solve these design problems [24]. This field consists of multiple research domains, such as Network Architecture Search (NAS) and HyperParameter Optimization (HPO). While these domains overlap, NAS only considers the design of the network itself, whereas HPO is more general and can consider all design choices. At the moment AutoML is not extensively applied to problems in medical segmentation [24], with a few exceptions [65] [62]. There are several optimization strategies, such as CARS [61], DARTS [37] and BOHB [16], which are discussed extensively in chapter 2.

The Medical Segmentation Decathlon (MSD) [2] was a challenge first published in 2018 that invites participants to create a system that performs well across a wide variety of ten Biomedical segmentation datasets. This system cannot have specific code for a dataset and three datasets are hidden and not available during the training phase. This should encourage participants to develop either a gen-

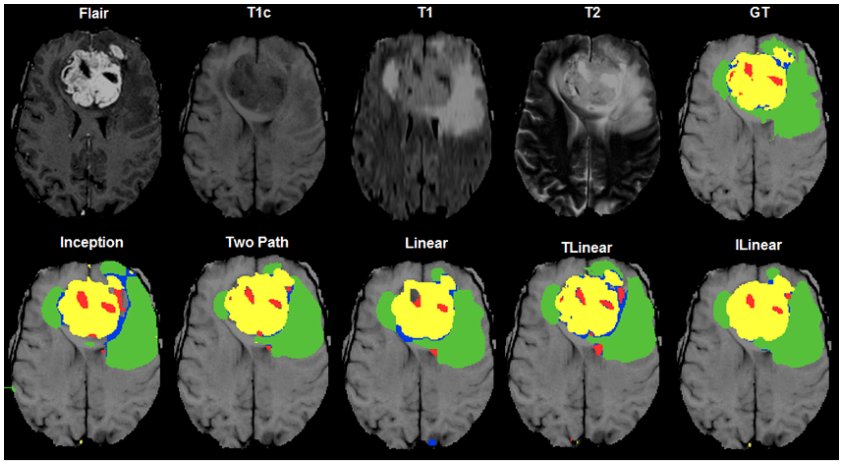


Figure 1.1: An example of medical segmentation of a tumor inside the brain (BRATS Dataset [39]). The top row shows different MRI-modalities and the ground-truth (right image). The bottom row shows results from different implementations [22]. Adopted from [22].

eral model *or* use AutoML in their designs. The top performer is nnU-net (no-new-U-net) [25], which achieved the top score across nearly all MSD tasks. However, nnU-net does not use AutoML; It adapts itself based on pre-defined rules and the fingerprint of the dataset. Other competitive contestants neither use AutoML [58] or use a search space for a specific subset of the hyperparameters, for example *K.A.V.athlon* [2].

This thesis fills this gap and aims to investigate the use of an AutoML algorithm in medical image segmentation, more specifically BOHB (see Section 2.5). For such an AutoML algorithm to work as intended, i.e. for it to be *viable*, the associated search space needs to be viable too; this search space needs to be computationally feasible and simultaneously include all configurations that potentially have the best performance. In other words, it cannot be too large nor too small. One of our research questions is focused on determining this search space. Furthermore, we have compared BOHB with Random Search and we compare this approach to the state-of-the-art. In short, this thesis aims to answer the following questions:

- What is a viable search space for AutoML strategies when those are applied to problems in medical segmentation? In other words, what is a viable subset of the complete search space?
- Is BOHB, one of the recommended AutoML practices, applicable in this situation and how does its performance compare with less complicated strategies, such as Random Search?
- Does AutoML (applied to the problems of the MSD) perform better than the state-of-the-art, such as nnU-net?

2

Theory

Neural networks for medical segmentation are based on CNN's for natural images, which we will explore first (Section 2.1). Afterward, we will discuss designs for medical segmentation (Section 2.2) and details about the training process, pre- and postprocessing in Section 2.3. In the final sections of this chapter we will discuss nnU-net (Section 2.4) and Automated Machine Learning (Section 2.5).

2.1. Convolutional Neural Networks

Neural networks for medical segmentation are based on those used for semantic segmentation of natural images. Therefore, to better understand these algorithms, this section explores neural networks used for analyzing natural images. More specifically, these networks are *convolutional neural networks* (CNNs). We will first introduce the fundamentals, which are used in advanced image handling networks, such as region-based networks or fully convolutional neural networks.

2.1.1. Neural networks

Neural networks are a branch of machine learning loosely based on a biological brain. A neural network processes input (for example, a picture of a handwritten digit) and returns an output, thus the displayed digit in the example. In other words, it is a function. A network can approximate with arbitrary precision a large range of (continuous) functions or mappings due to its structure [56]. A network learns by adjusting *weights* or *parameters*. These weights are distributed over *layers*. In a simple architecture, the output of a layer is the input for the next layer, but more complicated architectures will be discussed later in this chapter. One of the simplest layers (fully connected layers) has a weight for each combination of input and output variable. These weights are multiplied with the input and a bias is added to create the output. This output is fed into a non-linear function, an *activation function*, which makes it possible to estimate non-linear functions. Even a simple neural network has design choices, some of them parameterized, for example, the number of layers. To differentiate these design choices from the trainable parameters, we call them *hyperparameters*.

2.1.2. Convolutional Neural Networks

CNNs have been around for over 30 years [32], and are developed specifically for images. CNNs use convolution operations in at least some of their layers, hence their name. A convolution can be interpreted as an operation where a set of weights, i.e. a *kernel*, slides over the entire input image. For each pixel in the input image, its surroundings and the pixel itself are multiplied with the kernel, summed, and a bias is added to create an output pixel. Convolutional layers incorporate spatial relationships and some spatial in-variance in their design, which makes them very useful for image analysis. There are several other types of layers typically used in CNN's. Pooling [50] and normalization layers, which improve spatial in-variance and stability respectively. An example of a pooling layer is a max-pooling layer with a kernel size of 2x2: It represents every 2x2 pixels with the maximum value of those set of pixels.

2.1.3. Advanced image-handling neural networks

There are several types of problems associated with CNN's: Classification, object detection, and segmentation. Classification is detecting that an object belongs to a certain *class*, object detection is finding the *location* of this object and segmentation determining exactly which *pixels* or *voxels* belong to that object. For each of these pieces of information, large databases are made public. For example, ImageNet [14] is a large database with over 20.000 different classes, while COCO [35] has around 80 classes but the images are annotated with the exact location and segmentation masks of each object.

A CNN with only convolutional and pooling layers stacked in series does either not gain enough spatial in-variance or loses too much local information to accurately segment images or do object detection [17]. To solve this problem, many solutions have been proposed [6] [47] [48], where we will focus on Encoder-Decoder [3] networks, since a U-net (See Section 2.2) is an Encoder-Decoder network.

An Encoder-Decoder network is a Fully Convolutional Network, i.e. the output of each layer can be regarded as an "image". Besides convolutional layers, an Encoder-Decoder network has pooling layers which trade local information to be able to find global features [3]. However, both are needed to create an accurate mask. To have both types of information, the encoder creates, or "encodes", a set of features ranging from local (no pooling layers) to global (all pooling layers), which are used by the decoder to create the segmentation. There are several ways to encode the information of the higher resolutions. One way of encoding the local information is simply using skip connections to connect the high-resolution layers of the encoder with those of the decoder. Another way is storing and using the indices of the selected pixel in the pooling layers [3]. The former is the basic concept of U-net [49], which is discussed extensively in section 2.2.1.

2.1.4. Blocks

Computational power has become significantly cheaper during the development of CNNs. It therefore becomes feasible to replace single convolutional layers by *blocks*, or a subnet of a few layers. This means the researcher does not need to design the complete architecture, it can instead choose or design a few blocks and design a high-level architecture separately. There are many kinds of blocks, such as ResNet [18], DenseNet [21], and Deformable convolutions [12]. Deformable convolutions do not operate on a set of adjacent pixels, but instead use trainable vectors to determine which pixels are used to calculate an output pixel. ResNet blocks are also used in variants of U-nets (see Section 2.2) and are discussed in the next section.

ResNet In a traditional CNN, stacking more layers at a suitably deep model, the performance of the network drops [18]. Theoretically, however, adding more layers should not make a network less accurate. If those added layers are an *Identity* mapping, the evaluations and thus the accuracy of the network are equal. So if the performance drops with the added layers, the Identity mapping is not found. ResNet [18], shown in Fig 2.1, tries to solve this problem. It assumes that it is easier to learn a **0**-layer, where all weights are zero, than an identity mapping. In other words, it is easier to learn the *residual*. Because of this, the added layers can easily improve the performance, for example by focusing on edge cases [18]. ResNets are also trained faster than regular networks with a similar number of layers. This happens because large networks have to mitigate the *vanishing gradient* problem. This is the observation that the gradient for the first layers becomes very small, which makes it very hard to train those layers. Due to the skip-connections, i.e. the multiplication with the identity, the gradients do not vanish as much, and the network will converge faster.

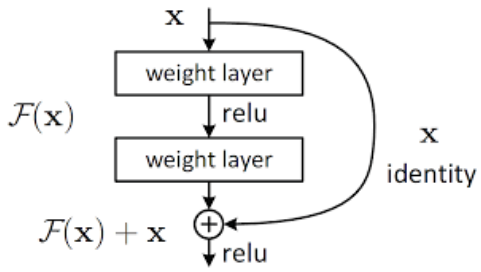


Figure 2.1: A simple form of a residual network [18].

2.2. Medical image segmentation designs

Medical image segmentation is similar to semantic segmentation of natural images, but it differs on a few key points; Medical images are typically three-dimensional, datasets are much smaller and in a single dataset there are typically only a few classes to segment. Although different base architectures, such as OBELISK [20] (which is based on deformable convolutions) exist, we will focus on U-net [49]. U-net is used as the base architecture in many solutions submitted to contests [2]. After discussing U-net [49], we will discuss some variations such as V-Net [40] and U-net GAN.

2.2.1. U-net

A U-net [49] is an encoder-decoder network. A schematic figure is shown in Fig 2.2. It is designed to capture features of every resolution to create accurate segmentation maps. In the encoder, which is on the left in the figure, the highest horizontal layer has the highest resolution and the outputs are downsampled using a pooling operation between each horizontal layer. So, at each layer the image has a lower resolution but a substantially larger field of view. Simultaneously, the output of each layer is connected to the part of the decoder with the same resolution. The detector is an encoder in reverse: it starts with the lowest resolution and ends with the original resolution. At each horizontal layer, data from both the previous layer in the decoder and the output of the connected encoder-layer is processed using convolutional and upsample operations until the original resolution is restored and the decoder generates the segmentation map.

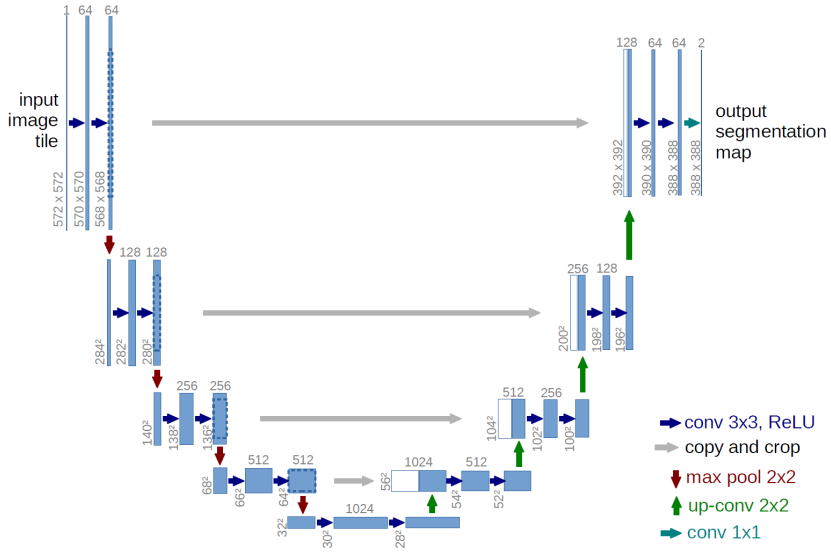


Figure 2.2: The U-net architecture as it was first proposed. The layers at the top have the highest resolution and those at the bottom have the lowest resolution. Adapted from [49].

2.2.2. Variations on U-net

As discussed above, medical images are typically 3D, while the original U-net paper only introduced a network that processed 2D images, and it calculated the segmentation map slice for slice. While this already showed good performance, it removes relationships between slices, which is why V-net and 3D-net replaced the 2D segmentations with the 3D-counterpart. Other approaches [45] [60] trained an ensemble of 2D networks, each processing a different orientation.

Other variations have also been applied, such as regulating the encoder using a Variational Auto-Encoder (VAE) [41] or implementing a Generative Adversarial Network (GAN) [9]. a VAE recreates the original image by using (a subset of) the encoded data. This helps training the encoder, which is helpful when the dataset is small. This strategy was used to achieve first place in the BraTS 2018 competition [41].

A GAN is composed out of 2 networks: a Generator and a Discriminator. The Discriminator tries to tell the output of the Generator and the ground truth (the segmentation labels) apart and the Generator tries to fool it. In this setting [9], the Generator is a U-net, and the Discriminator is combined with

a regular loss function, the Dice Loss (See Sec. 2.3.3). Adding a Discriminator to the loss function improves performance [9], however, it needs additional computational resources and we believe it is therefore not yet suitable to be implemented in our framework.

2.3. Preprocessing and other Pipeline Considerations

The performance of a trained algorithm does not only depend on the algorithm and design choices in the network itself such as in the previous Sections (Sec 2.1 and Sec. 2.2), but also on preprocessing, data augmentation, loss functions, and postprocessing. In other words, it depends on the complete *pipeline*. In this section, the focus is not on the algorithms themselves, but rather on the most important design choices in the rest of the pipeline.

2.3.1. Preprocessing

The most important preprocessing steps include resampling and normalization. Resampling is applied to ensure that the voxel size (the physical volume represented by one voxel) is equal across the complete dataset. This is necessary because medical scanners and their configurations can differ greatly in the voxel sizes they produce [59], while the physical volume is relevant information.

Normalization can also improve the learning capabilities of a network significantly [53]. Not only does it decrease the influence of the mentioned differences between scanners, but it also ensures that large numerical values in the input do not dominate smaller values, but instead that all features are weighted equally. There are various normalization methods, the most commonly used methods are z-score, percentile clipping, and min-max normalization [43]. Z-score normalization scales the image in such a way that the output has a standard Gaussian distribution. However, z-score normalization is not optimal when there are outliers present in the data [53]. Percentile clipping [25] is a modification of z-score normalization. It clips outliers first, for example to the lowest and highest 5 percentile, and only afterward applies z-score normalization. Finally, min-max normalizes the data by scaling the data linearly to have a pre-defined minimum and maximum value.

2.3.2. Data Augmentation

Data augmentation, i.e. manipulating the initial data to create more similar data points, is very effective at increasing both the amount and diversity of data [11]. This helps deep learning a lot since these techniques rely on large, diverse datasets [44]. There are many different forms of data augmentation in the field of medical images. These can be categorized as either spatial, kernel-wise, or intensity space transformations. All these transformations are designed such that they are 'safe', i.e. the label or segmentation map is still valid after the transformation. A list of possible data augmentations includes spatial transformations such as flipping, rotating, shearing, cropping, and elastic transformations [52]. Kernel operations such as sharpening or blurring the image are not used often, since those can also be represented inside the network itself. Last but not least, intensity-space augmentations include transformations such as adding Gaussian noise, changing the brightness (adding a bias), or changing the contrast. If the image has multiple modalities, this can be done separately for each modality.

2.3.3. Loss Functions

Loss functions are an important component of the training process, and there is not a single loss function that performs well over the complete range of datasets [38]. These loss functions can be classified into 4 categories; Distribution-based losses (i.e. Cross-entropy loss and variants), Region-based losses, boundary-based losses, and combinations of the previous categories.

Distribution-based losses are all derived from the Cross-entropy loss, which calculates the loss for each voxel and each class based on $\sum_i^N \sum_c^C -g * log(p)$, for all classes C (including background) and all voxels N. The losses for each combination of voxel and class can be combined in several ways, to, for example, compensate class imbalance (Weighted cross-entropy loss) or increase the importance of difficult predictions (i.e. Focal Loss [34]).

Whereas Distribution based losses are calculated for each voxel separately, Region based losses are calculated using the (fuzzy) sets of the ground truth and the prediction. The popular Dice Loss [15] calculates the loss using the Ground truth set G and (fuzzy) Prediction set P: $1 - \frac{2|G \cap P|}{|G|+|P|}$. Other variants are for example the Intersection over Union ($\frac{|G \cap P|}{|G \cup P|}$) and the Generalized Dice Loss [54], which

compensates for the frequency of different classes.

Boundary losses minimize the distance between the boundary of the ground truth and prediction segmentation [29]. Using this loss, misclassified voxels far away from the ground truth are penalized heavily. The Hausdorff loss [23], for example, calculates the distance for each voxel in the prediction set the distance to the ground truth, and vice versa. The sum of these distances is the complete loss. To prevent instability, they should always be combined with Region based losses, such as the Dice-loss [38].

However, typically the best losses are combinations of several losses, for example, the Dice Loss with the Focal loss or Hausdorff loss [38]. These are the most robust and achieve the best results across several datasets.

2.3.4. Post Processing

In several preprocessing steps, specific (assumed) a priori knowledge of the medical images is used to improve overall performance. For example, the a priori knowledge that differences in medical scanners can be (partly) compensated by normalization is commonly included in preprocessing steps. Similarly, postprocessing can be used to apply similar knowledge of the segmentation images. Largest Connected Component Analysis is one of those. If the training data only has one object per image, it is unlikely that unseen images will have multiple objects [25]. Therefore, by only keeping the largest object and removing all smaller ones, the usually small objects created by noise are removed and the overall performance is improved.

Another postprocessing operation applies a priori knowledge of the network itself by ensembling multiple networks. This reduces the overall error because different networks, with different starting conditions, will end up in different local minima [64]. There are several types of Ensembling, such as majority voting and mean ensembling [36].

Even though postprocessing steps can increase accuracy substantially, it is not used during training. This implies that it is less relevant for Automated Machine Learning since it can be applied after the network structure is optimized and the network itself is trained.

2.4. nnU-net: A step towards generalization

nnU-net [25] was designed for the Medical Segmentation Decathlon (MSD) [2], where it achieved first place overall. MSD is a collection of 10 diverse datasets in the medical segmentation field. Most participants were not able to generalize to the hidden datasets as is visible in Figure 2.3, but nnU-net was able to do so. It also participated in other challenges where it showed state-of-the-art results. In total, its design was validated on 23 datasets and 53 segmentation tasks, often outperforming specialized pipelines.

NnU-net works by dividing the hyperparameters into three sets; fixed, rule-based, and empirical hyperparameters. The last set only contains post-processing and ensemble selection and is therefore only searched through after the network has been trained. Hyperparameters in the rule-based group are decided by rules based on the fingerprint of the dataset. For example, the depth is taken as large as possible, as long as the necessary patch size stays similar in size as the original image or 128^3, whichever is smaller. It has similar rules for resampling, normalization, and batch size.

For specific datasets, nnU-net also uses a Cascaded 3D-Unet. This is used for datasets where the images are large and do not fit into one patch. To be precise, where the median image size has at least four times as many voxels as the maximum patch size. This is not the case for the Brain dataset of MSD, which is the dataset in our experiments. A cascaded U-net consists out of two stages, each with its own network. The large image is resampled to a lower resolution that does entirely fit in one patch. The network of the first stage is trained on this low-resolution dataset. The second stage upsamples the output of the first stage and concatenates that with the original image. It then uses this combined tensor to sample patches from and train the second network.

However, by defining most hyperparameters in either a fixed or rule-based way, these are ultimately chosen based on heuristics. These heuristics incorporate domain knowledge but might also be biased, consequently delivering sub-optimal designs. Instead, if these hyperparameters are left to an optimization method, the influence of bias can be reduced. In the next sections, several Automated Machine learning methods are introduced that can optimize these design choices.

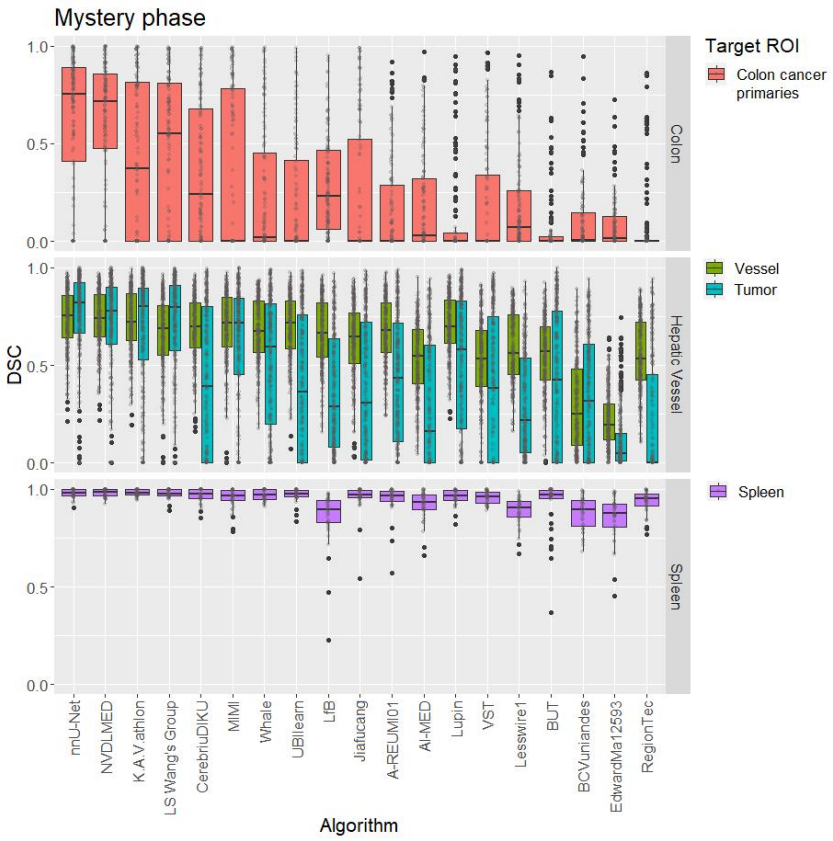


Figure 2.3: Perfomance on the hidden sets of the Medical Segmentation Decathlon [2]. Most participants did not generalize.

2.5. Automated Machine Learning

Algorithms are hard to design, which Automated Machine Learning tries to automate. AutoML has multiple specialized fields of research, such as Network Architecture Search (NAS), Meta-learning [13] and Hyperparameter Optimization (HPO). While these fields share overlap, NAS does typically fewer concessions to the search space [24]. It has several solutions for such a large search space, such as morphing existing networks one step at a time [57], Block based search [66] where a manually designed hypernetwork reduces a large part of the search space or simply by defining boundaries between a macro and micro level, such as in C2FNAS [62], which is an example in the medical segmentation domain. Meta-learning is the process where the knowledge of previous tasks is used to improve the learning process for new tasks. We will, however, focus on HPO: finding the most optimal combination of several hyperparameters, even if some hyperparameters are conditional on others.

2.5.1. HyperParameter Optimization (HPO)

A significant part of any architecture can be described by a set of *hyperparameters*. Finding the most optimal set of these hyperparameters, which is crucial for performance, is an optimization problem in a high-dimensional, diverse search space. Hyperparameters can be categorical, discrete, or continuous and with sufficient resources, a search space can already span 19 hyperparameters [24]. Furthermore, evaluating a single configuration to convergence is expensive in terms of computation cost; it is, therefore, all the more important that an optimization strategy finds the optimum with as few evaluations as possible. In this section, we discuss several optimization methods, which can be categorized into several categories; gradient-based methods, Bayesian optimization methods, and Bandit-based optimization methods. Finally, there are also optimization methods that combine several sub-strategies, these are discussed in section 2.5.5.

2.5.2. Gradient based methods

During training, the parameters are optimized using gradient-based methods. DARTS [37] applies gradient search to hyperparameters. To be able to differentiate the loss function with respect to those hyperparameters, DARTS has to ensure those are continuous, instead of discrete. DARTS achieves this by using a softmax function to select the hyperparameter defining an operation. It then uses a weighted ensemble for each operation with respect to this softmax. To mitigate some of the overfitting, the weights are updated with the training set and the hyperparameters based on a validation set [19]. An example of DARTS applied to medical segmentation is V-NAS [65]. V-NAS defines a hyper-network with 21 blocks. Each of these blocks is a softmax-weighted combination of three possibilities: a 2D convolution, a 3D convolution, or a Pseudo 3D convolution, which is a combination of a 2D and an orthogonal 1D convolution. One final drawback of DARTS is that it quickly becomes computationally heavy since for each operation it uses an ensemble that needs to be trained simultaneously. Therefore, the search space is limited to a small network with only a few options for each operation [65] [37].

2.5.3. Bayesian Optimization

Bayesian Optimization uses an auxiliary model to predict the most optimal hyperparameters [19]. To achieve that, it uses two models or functions: the auxiliary or (probabilistic) surrogate model and an acquisition function.

Two popular Bayesian optimization methods are Gaussian processes and Random Forests. Whereas Gaussian Processes [46] predict the performance given a specific configuration, Random Forests [4] predict the likelihood of the configurations given that the performance is either higher or lower than a threshold. Gaussian Processes predicts a Gaussian function for each configuration, using a covariance matrix to determine the influence of data points. Unfortunately, Gaussian Processes scale cubically with the number of observed data points. Random Forests only scale $O(nlogn)$, so they are much better equipped at handling large solution spaces [4]. The acquisition function finds the next configuration the method needs to sample. TPE achieves this by sampling k configurations from the probability function that performs better than the threshold. It furthermore calculates the probability that these configurations are drawn from the set of "worse" configurations. The acquisition function then selects the configuration where the probability ratio $P_{better} \ / \ P_{worse}$ is the highest. The method continues with evaluating this configuration and uses this new data point to update the model and threshold.

2.5.4. Bandit Based Optimization Methods

The easiest way to estimate the performance of a CNN is by training it. However, training many CNNs could reach computation power in the order of thousands of GPU days [24]. In the previous paragraphs, the computation power was reduced by limiting the number of evaluated configurations. In Bandit-based methods, the computational power is reduced by reducing the computational power or budget b for each evaluation. With a reduced budget, only a rough estimation for the eventual performance can be made. However, as long as the relative ranking of the architectures stays similar, the most promising configurations will still be found. These can then be trained completely to find the best performing architecture.

A basic but powerful strategy is *Successive Halving* [27]. It starts with a set of k random configurations. After training those for a specified budget, the optimizer drops the worst half. It doubles the budget for the remaining configurations and repeats this process until only one configuration is left. It performs well, but the researcher has to decide at the beginning how many configurations the strategy starts with. Too little, and good solutions are missed, too many, the solutions are not trained and the results are therefore not reliable. This is a trade-off between *exploring* enough solutions to find the optimal solution and simultaneously training (*exploiting*) the selected solution enough to be confident it is indeed the most optimal solution.
HyperBand [33] tries to mitigate this problem by dividing the budget into several groups. In each of these groups, a different number of starting solutions and assigned resources for each configuration are chosen. This ensures that many configurations are tried and that at least a subset of those configurations are trained enough that the predictions have enough fidelity. However, these strategies have still an important drawback. Both of these strategies work by selecting all solutions at the start. These methods cannot propose new solutions based on the evaluations of the existing ones. In the next section, we will discuss methods that mitigate these problems.

2.5.5. Combinations

One interesting combination is CARS [61] which combines gradient descent with a genetic algorithm. It uses a super network from which several networks are selected using genetic search. These networks use gradient descent, like DARTS (Section 2.5.2) to update themselves, and thus the super network. This greatly reduces the search time necessary than more naive implementations.

Another combination is Bayesian Optimization and HyperBand (BOHB) [16], which combines TPE and Hyperband. It achieves quickly a reasonable performance by using low fidelities and selection in Hyperband and a good final performance using Bayesian Optimization. It uses Hyperband to determine how many configurations are trained and how long each of them is trained. However, instead of selecting random configurations, the models are selected by a Bayesian Optimization method. This ensures that the information found from the first trials is used fully. It starts with a set of random samples since with little or no observations, there cannot be a valid model. Once N_{min} observations with a specific model are made, a TPE model is made for that budget. When BOHB samples new configurations, it will use the model created for the highest budget since that model has the highest fidelity. If performance can be estimated with low fidelity evaluations, BOHB can be very useful, since it is robust, efficient, and parallelizable [24].

3

Method

To run our experiments, we developed our framework, which we discuss in Section 3.1. We discuss the general experimental setup in Section 3.2. We used this framework to test the usefulness of BOHB (Section 3.4), determine which hyperparameters are useful to include (Section 3.3), and if our implementation of BOHB outperforms the state-of-the-art (Section 3.5).

3.1. Framework

We developed our framework, AutoMONAI, which uses BOHB to optimize several hyperparameters of a U-net (See Figure 3.1) and accompanying pipeline. Five hyperparameters describe the network architecture: the depth (number of horizontal layers in the figure), the number of channels in the first layer, and its multiplication factor between depth layers (which is two in the figure, i.e. the number of channels double every depth layer), convolution type (2D vs 3D) and finally the number of convolutions in each depth layer. Three hyperparameters describe other parts of the pipeline: normalization and resampling during the preprocessing stage and finally the loss function. All of these hyperparameters and their respective ranges are discussed in more detail in Section 3.3, where we investigate the influence of these hyperparameters on the performance. We implemented this in PyTorch [42] and MONAI [10]. Our experiments ran on our GPU cluster, which allocates its jobs on one of our RTX 2080 Ti 11GB GPUs. This limits the graphical memory to a maximum of 11 GB, which means some configurations are not possible. In total, several thousand GPU hours were used for these experiments and the development of AutoMONAI. There is one difference between our implementation of BOHB with the reported version: our implementation of BOHB does not parallelize.

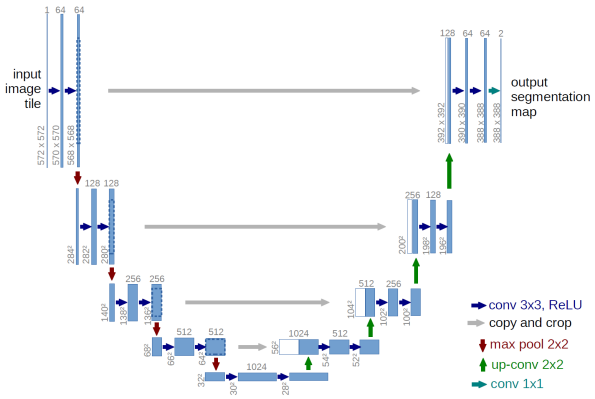


Figure 3.1: The U-net architecture [49], which is configured in our framework using multiple hyperparameters. See also Section 2.2

3.2. Experimental setup

In this section, we describe all settings that are the same across all experiments. Most of these settings are equal to the settings used in nnU-net [25], which we used to compare AutoMONAI in Section 3.5, to ensure performance differences in the final experiment are not based on differences in these settings. Settings that are different than nnU-net are motivated in Section 3.5.1. We used the Brain Dataset from the Decathlon challenge [2], which is largely based on the 2018 BraTS Dataset [39], to run our experiments on. Furthermore, we used the Dice score [15] to compare architectures, since it is widely used in literature. The randomly sampled patches had a size of 128^3 and 128^2 for 3D and 2D networks, respectively. The batch size was 2 for 3D networks and $2 * 128$ for 2D networks, so both batches contained the same number of voxels. After each convolution, instance normalization and LeakyReLU were applied. These networks were trained by an ADAM optimizer [30] with a learning rate of $3e-e4$, a β_1, i.e. first moment weight, of 0.9 and second moment weight β_2 of 0.999. We reduce the learning rate by a factor 5 whenever the improvement in the last 30 epochs is not at least 0.01%. All networks were trained 200 epochs, where one epoch is one sample from each image in the training set. The data was augmented using flipping (along any spatial axis) and adjusting contrast with a γ between 0.7 and 1.5.

3.3. Determining a viable search space

To answer the first question, "What is a viable search space for AutoML strategies when those are applied to problems in medical segmentation?", we had to find a computationally feasible search space that includes all configurations that potentially have the best performance. Since "computationally feasible" depends on the available hardware, we will focus on selecting hyperparameters that influence the performance and are likely to change between problems. Therefore, we had selected a set of candidate hyperparameters (introduced in Section 3.1 above) and determined if those were influential on the performance. We selected a standard configuration where we changed one hyperparameter at a time. To search around the expected optimum, this standard configuration is based on preliminary experiments and literature. The standard configuration and the range for each hyperparameter are shown in Table 3.1. In the next sections, we describe the range and motivation used for the hyperparameters and the analysis of the results.

3.3.1. Hyperparameter selection

We selected five hyperparameters (depth, channels at the first layer, channel multiplier, convolution type, and the number of convolutions in each depth layer) describing the architecture of the network, and three describing the rest of the pipeline, those being normalization and resampling during the preprocessing phase and the loss function.

The depth determines for a large part the eventual field-of-view since this field doubles after each downsampling layer. It also influences the number of learnable parameters. Both the learnable parameters and the field-of-view influence the performance. NnU-net use the largest depth that does not create a larger field-of-view than the original image, with a maximum of depth of 7 for 2D networks and 6 for 3D networks. NnU-net determines the depth for each spatial axis individually, while we have chosen to have a single depth for all spatial axes. For MSD's Brain dataset specifically, nnU-net has a depth of 6. We have chosen an inclusive range between a depth of 2 and 6 layers.

The number of channels in the first layer, together with the channel multiplier in the next paragraph, determine the number of channels in the network in this implementation. Theoretically, more channels always perform better (since unnecessary channels can be weighted in such a way that they do not influence the prediction), until computational constraints are exceeded. Therefore, a balance has to be found between the benefit of adding channels and the added computational cost. In literature, these values range from 16 [40] and 30 [25] to 64 [49]. For this hyperparameter, we have chosen a logarithmic scale with base 2, since we expect very little difference between two consecutive numbers and a computational speed-up is possible when the number of channels is a power of 2. Our range is $R_{fc} = 2^3, 2^4, ..., 2^7$.

U-net [49] double the number of channels after each downsampling step, and this choice has been

adopted in almost all variations [40] [25] [8]. However, we believe the optimal value of this hyperparameter is influenced by the type of problem. If the solution of a problem does not rely on a lot of global features, but rather on local features, it is better to allocate resources to those local features and therefore choose this multiplier to be lower, and vice versa. We have chosen for a range starting at 0.5 (so halving, instead of doubling, at each depth layer) to 3. If the number of channels is not an integer, it is rounded down.

U-net [49] (See Section 2.2.1) use 2D convolutions, where variations such as V-Net [40] uses 3D convolutions. Whereas 3D convolutions handle the 3D nature of medical images by design, this can also be adversarial. For example, when the last dimension adds very little information because the image is anisotropic [25], it is better to use 2D convolutions. We therefore compare two options for this hyperparameter: 2D vs 3D.

The number of convolutions in each depth layer is 2 in most variations [49] [40] [25] [8]. However, more convolutions can potentially extract more information from a certain resolution and provide a larger field of view. We investigate one to four convolutions, which can be bypassed with a residual path. Additionally, we use this hyperparameter to investigate the effect of removing the residual path. This last configuration has one convolutional layer with no residual path.

As mentioned in Section 2.3, normalization can improve performance, but the optimal version differs for different tasks. For the Brain dataset, nnU-net uses z-scoring. In this experiment, we have included z-scoring, 99.5 percentile clipping before z-scoring, min-max normalization (with a minimum of 0 and a maximum of 1), and not applying any normalization.

Resampling ensures the physical size of a single voxel is the same across the dataset, which is relevant information (see Section 2.3). nnU-net resamples images differently based on the fingerprint of the dataset [25]. If the image is anisotropic, for example, if the image has a large slice thickness, nnU-net resamples the out-of-plane axis using nearest-neighbor interpolation. All other axes are resampled using third-order spline interpolation. Our focus is on a different part of resampling, which is the voxel size the images are resampled to. We have chosen for no resampling, the mean and median voxel sizes.

There is not a single loss function that trains a network to the best possible Dice score (see Section 2.3.3) for every problem. However, loss functions that combine separate functions, typically score better. Networks trained using the following loss functions achieve the best Dice score in at least one out of four tested tasks [38]. These tasks include segmenting the liver, liver tumors, pancreas, and multi-organ segmentation. These loss functions are: Dice + Cross-Entropy, Dice + Focal Loss, Dice + Hausdorff loss, Dice + Top K loss. We included these functions in our search space. We also included the Dice loss and the Generalized Dice Loss, even though these functions did not perform as well in this study, these are often used in literature (The Dice loss is the most commonly used loss function in the MSD [2]). nnU-net, which we compared our framework with in the next section, uses a Dice + Cross-Entropy loss. However, their implementation is exactly 1.0 lower.

3.3.2. Statistical analysis

We are interested in the final Dice score on unseen data, i.e. the test set. Since this score is affected by noise caused by small changes in the weights, we looked at the steady-state (fully trained) part of the loss curve. Using the results of preliminary experiments, we assume this condition is true for the last 25 epochs, i.e. the last 5 values. We assume the noise is independent between every 5 epochs and used the last 5 calculated Dice scores for each configuration as samples of the population associated with the performance of each configuration. For each hyperparameter, an ANOVA test was applied to test for statistically significant differences between the performance of the networks using the different hyperparameter values. To correct for multiple testing, Bonferroni correction was applied. Hence, a p-value of $\frac{0.05}{7} \approx 0.007$ was considered statistically significant.

Hyperparameter	Range	Standard configuration
Depth	[2, 6]	4
Number of channels in the first layer	2^3 ... 2^7	2^5
Channel multiplier	[0.5 , 3]	2
Convolution type	2D or 3D	2D
Number of convolutions	[0, 3]	2
Normalization in preprocessing	None, z-scoring, percentile clipping, min-max	z-scoring
Loss function	Dice, GDL, DiceCELoss, Dice + Hausdorff loss, Dice + Top K loss, Dice + Focal loss	DiceCE Loss

Table 3.1: The search range for each investigated hyperparameter.

3.4. Investigating the applicability of BOHB

Literature recommends using BOHB as HPO method since it is an efficient method [24]. However, a prerequisite needs to be true; The performance ranking of partially trained networks needs to be an estimation for the ranking of fully trained networks. To validate this recommendation, we did two experiments: One to check if the prerequisite is true in our use case (Section 3.4.2) and a second experiment where we compare different settings of BOHB and Random Search (RS) (Section 3.4.3). In preparation for these experiments, we used a grid search across a small search space, discussed in Section 3.4.1.

3.4.1. Grid search

We computed a grid search across a small search space. This way, we know the complete performance surface after every 5 epochs of the defined search space, which raw data we analyzed in the first experiment. We used this data also in the second experiment, where we used this to mock the training of the networks during BOHB or RS. This speeds up the experiments but has as a side effect that a configuration always has the same performance, while this is not necessarily true. Several sources of noise are ignored, such as data augmentation and the initialization of the weights. We believe this noise to be small and do not influence the steady-state performance.

The small search space is a subset of the search space discussed in the previous paragraph, consisting out of three hyperparameters: depth, the number of channels in the first layer, and the convolution type. These hyperparameters all describe the network architecture, so these hyperparameters influence each other. This small search space is summarized in Table 3.2. Other hyperparameters are fixed on their default values, as shown in Table 3.1.

Hyperparameter	range
Convolution type	2D or 3D
Depth	[2, 6]
Number of channels	2^3 ... 2^7

Table 3.2: The hyperparameters of the small search space used to validate the use of BOHB.

3.4.2. Validate a prerequisite

A prerequisite of BOHB is the ability to predict the final *ranking* of the different configurations based on the results after the networks have only been trained for a small number of epochs [16]. Using the results of the grid search, we calculated the ranking every 5 epochs. To compare these rankings, we used Kendall's Tau (τ) [28], a correlation metric between two rankings. With a correlation of 1, the rankings are equal, with 0 there is no correlation and a τ of -1 indicates a negative correlation, i.e. one is the reverse of the other. We calculate this correlation between the rankings of the Dice score after every 5 epochs *and* the Dice score after the final training epoch, i.e. 200 epochs. For large N ($N > 10$), the formula in Eq. 3.1 follows a normal distribution. This should therefore be larger than 1.96 to have

a significance level of 0.05.

$$z = \frac{3 * \tau\sqrt{N * (N-1)}}{\sqrt{2(2N+5)}} \tag{3.1}$$

3.4.3. Comparing BOHB and RS settings

We compared different settings of BOHB and RS, which are summarized in Table 3.3. We mocked the training procedure of the networks using the results of the grid search. We compared three settings of BOHB. We compare η (at each step in an iteration, it keeps $\frac{1}{\eta}$ of the configurations), the number of iterations, and the optimization function BOHB uses to compare networks. The latter is either the loss function calculated across patches of the validation set (i.e. the validation loss) or the Dice score inferred across the complete image of the same set. The validation loss is commonly used to monitor overfitting, so if it can be used simultaneously to compare configurations, it is not necessary to compute another metric. We use $1 - Dice$ because BOHB minimizes this value. For η, we use the original 2 of Successive Halving [27] and the more recent 3 [24]. Furthermore, We compared these settings with a baseline, RS, with two settings for the number of iterations. 19 and 38 RS iterations can train the same number of epochs as BOHB with an $\eta = 3$ and respectively 5 and 10 iterations.

Since BOHB starts with randomly sampling configurations until it can build a model, every BOHB run is different. It will therefore not find the same configuration every run (Every RS run is also different, of course). Therefore, we compared multiple runs (n=10) for each setting. In total there are 100 different runs. We define the best Dice score of a fully trained network to be the final result of a run. We compare the results of the different BOHB and RS runs with a Kruskall-Wallis test [31] with a significance level $\alpha = 0.05$, since we do not expect the results to be normally distributed; the best loss should asymptotically reach the best loss of the search space.

Optimizer	Metric	η	Iterations
BOHB	1-Dice score	3	5
BOHB	1-Dice score	2	5
BOHB	1-Dice score	3	10
BOHB	1-Dice score	2	10
BOHB	DiceCELoss	3	5
BOHB	DiceCELoss	2	5
BOHB	DiceCELoss	3	10
BOHB	DiceCELoss	2	10
RS	-	-	19
RS	-	-	38

Table 3.3: The different settings used for BOHB and RS. The metric is used by BOHB internally to compare configurations.

3.5. Comparison with the state-of-the-art

Our third research question is: *Does AutoML (applied to the problems of the MSD) perform better than the state-of-the-art, such as nnU-net?* To this end, we have compared our framework to nnU-net [25], the winner of the MSD [2] (see Section 2.4 for more specific details) on the Brain dataset of the MSD. Many hyperparameters, which are summarized in Table 3.4, are either the same as nnU-net or are optimized by BOHB. The differences that remain, are discussed in the next section. AutoMONAI uses the recommended BOHB parameters of $\eta = 3$, a minimum and maximum budget of 5 and 200 epochs, and it iterates 5 times, which gives a total budget of 3700 epochs.

3.5.1. Differences with nnU-net

Whereas nnU-net crops images to the region of non-zero values to reduce the memory footprint, our framework does not have that functionality. We do not believe this has a significant influence on the performance. While nnU-net adapts its patch size so that it is as large as possible, with a maximum of 128^3 or 128^2, we have fixed our patch size to this maximum. If we implemented that our patch size changes based on the configuration, we would change multiple hyperparameters by only explicitly changing one. We do not think this has a significant impact since the patch size for this dataset is the

Hyperparameter	nnU-net	In search space	Search range
Preprocessing			
Resampling	To median voxel size	X	None, Mean and median voxel size
Normalization	Z-scoring	X	None, z-scoring, percentile clipping, min-max
Cropping	To non-zero image		Not implemented
Data augmentation	Rotation, Scaling, Flipping, Gamma augmentation, elastic scaling		Fixed: flipping, gamma augmentation, elastic scaling
Patch size	128^3 and 128^2		Fixed: 128^3 and 128^2
Batch size	2 and 89		Fixed: 2 and 256
Patch selection	No oversampling of foreground classes (in this dataset)		Fixed: no oversampling (anywhere)
Network architecture			
Depth	6	X	2 ... 6
Channels in the first layer	30	X	2^3 ... 2^7
Multiplication factor channels	2	X	0.5 ... 3.0
Convolution type	Both 2D and 3D	X	2D and 3D
Number of convolutions	2	X	0 .. 3
Resnet like structure	Not applied		Fixed: applied
Activation function	Leaky Relu		Fixed: Leaky Relu
Normalization	Instance normalization		Fixed: instance normalization
Training procedure			
Loss function	Dice + Cross Entropy loss	X	Dice, GDL, DiceCELoss, Dice + Hausdorff loss, Dice + Top K loss, Dice + Focal loss
Optimizer	Adam		Fixed: Adam
Learning rate (LR)	3e-4		Fixed: 3e-4
LR scheduler	Reduce by a factor 5 after 30 epochs if exponential moving average of the training loss does not improve by 5e-3		Fixed: reduce by a factor 5 after 30 epochs if best score does not improve by 0.01%
Epoch definition	250 instances		Fixed: all instances in the training set
Stop definition	When LR is lower than $1 * 10^{-6}$ and exponential moving average of validation loss does not improve by at least $5 * 10^{-3}$		Fixed: 200 epochs
Cross-validation	Five way split		Fixed: four way split (60%-20%)
Postprocessing			
Largest connected component analysis	Applied		Fixed: applied
Ensembling	Cross-validation ensembling and ensembling of best architectures		Fixed: only cross validation ensembling
Inference	Center voxels weighted higher, 50% overlap		Fixed: Gaussian distribution, 50% overlap

Table 3.4: Comparison of the hyperparameters between nnU-net and our framework.

same for both designs. Similarly, nnU-net tries to use an as large as possible batch size. In this case, that is 2 for 3D networks and 89 for 2D networks, while we have fixed that to 2 for 3D networks and 256 for 2D networks. The patches for the 2D networks are not randomly sampled. Rather, we randomly sample 3D patches which we transform into 2D samples by extending the batch axis to include every sample from the z-axis.

Our data augmentation is also less extensive due to implementation details of MONAI. We did not implement rotation, elastic deformation, or scaling, which gave unpredictable errors, which we did not manage to solve.

nnU-net defines an epoch as 250 training batches because it is designed to do the same for each dataset. However, a dataset with more images can have different design specifications than smaller sets, so we use the more commonly used definition of training with all images in the training set once.

nnU-net applies five-fold cross-validation across the entire dataset and reports the mean of the results achieved on each validation set. Furthermore, they provide the results the ensemble of these five-fold cross-validation achieves on the test set provided by MSD. Since we have not used this test set, which is something we have reserved for future work, but do we want to implement ensembling we have chosen to use one fold as a test set. The other four folds are used in a four-fold cross-validation split, on which we train the networks and ensemble those. We do not report results on the validation sets, since these metrics are used in BOHB, and the system might overfit on the validation split.

nnU-net reduces the learning rate by 5 if the exponential moving average does not improve with at least $5 * 10^{-3}$ in the last 30 epochs, while our system reduces the learning rate with the same factor when the loss does not improve by at least 0.01 % in the same time, which is a build-in learning rate scheduler of PyTorch. We do not believe this difference is very large.

nnU-net ensembles the different architectures it trains (2D and 3D) and ensembles them, to select the best out of all of these combinations. Our framework does not do that, which we consider future work. However, nnU-net did not improve the performance by ensembling the different architectures on the Brain dataset.

4

Results

4.1. Determining a viable search space

In this experiment, we did a 1D search to determine which hyperparameter influence the performance. The loss curves of these experiments are visible in Appendix A. To determine whether the converged performance is from a different population, we performed multiple ANOVA tests, which results are shown in Table 4.1. Results are significant when the p-value is (approximately) 7.14×10^{-3} or lower. Our experiment indicates that the convolution type (p=5.64×10^{-6}), the number of channels in the first layer (p= 4.04×10^{-4}), the multiplication factor of the channels between each depth layer (p= 4.54×10^{-17}), and the loss function (p= 1.44×10^{-9}) are significantly influencing the eventual performance, and therefore need to be optimized. However, normalization (p=1.42×10^{-1}), resampling (p=1.92×10^{-1}), the depth (p=3.93×10^{-2}) and the number of convolutions (p=2.69×10^{-1}) are not significantly influencing the performance.

Hyperparameter	ANOVA	p-value
Depth	3.09	3.93×10^{-2}
Convolution type	111	5.64×10^{-6}
Number of convolutions	1.40	2.69×10^{-1}
Multiplication factor channels	3.09	4.54×10^{-17}
Channels in first layer	10.8	4.04×10^{-4}
Loss function	29.9	1.44×10^{-9}
Normalization	2.09	1.42×10^{-1}
Resampling	1.81	1.92×10^{-1}

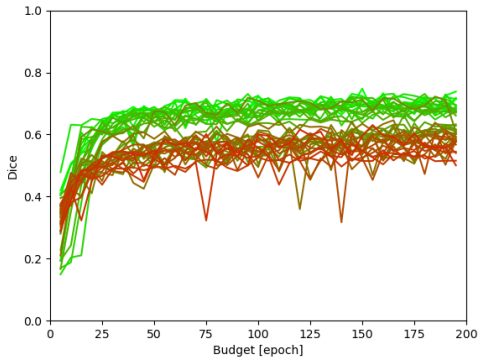
Table 4.1: ANOVA values and corresponding p-values for every one dimensional search across each hyperparameter.

4.2. Investigating the applicability of BOHB

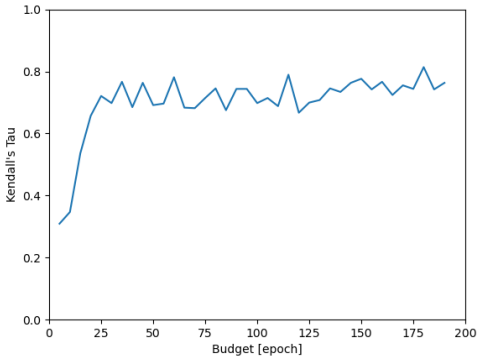
In this section, we discuss the results which investigates the applicability of BOHB. First, we discuss the relationship between intermediate and final rankings of hyperparameter configurations (Sec. 4.2.1). Afterwards, we discuss the results of the comparison between different BOHB and RS settings (Sec. 4.2.2).

4.2.1. Relationship between intermediate and final rankings

We have trained all configurations in the small search space, which spans the depth, convolution type and number of channels in the first layer. Fig 4.1a shows the loss functions of all successful configurations. Successful, since some configurations exceeded our computational limits and therefore do not have loss curves. The loss functions are color-coded based on their final performance; the green lines have the best final performance, red the worst. The performance curves can be visually split into two sets, which corresponds with the convolution type; 2D configurations perform better than 3D configurations. Two observations with regards to the ranking can be made from this graph: large shifts (> 10)



(a) The loss curves of all successful (not exceeding computational limits) configurations. The curves are color coded based on the final ranking, with green for the best performance, red the worst.



(b) Kendall's τ between the ranking after a varying number of epochs and after 200 epochs.

Figure 4.1: Comparing results on the small search space (depth, convolution type and number of channels in the first layer).

are almost exclusively in the initial 25 epochs and there are small shifts even in the final stages of the training process. The latter observation is partially explained by the small performance differences.

These observations can also be shown using Kendall's τ, plotted in Fig. 4.1b: the correlation rises quickly, but settles on a level between 0.7 and 0.8. Already at the first data point (5 epochs) with $\tau = 0.31$, there is a significant correlation. Recalling the formula for transforming Kendall's tau in a normal distribution (Eq. 4.1), and filling in $N = 50$, we calculated $z = 3.17$, which is larger than 1.96 and therefore the correlation is statistically significant.

$$z = \frac{3 * \tau \sqrt{N * (N-1)}}{\sqrt{2(2N+5)}} \tag{4.1}$$

4.2.2. Comparing different settings of BOHB and RS

We compared different settings of BOHB and RS. We define the Dice score of a single run as the best result achieved during the run. The average Dice score and the worst Dice score for every setting of BOHB and RS are shown in Table 4.2. The best performance achieved by each setting is not shown, since for almost all settings this was the theoretical maximum of 0.719. Most results are very similar, which is also shown by the Kruskall-Wallis test, which had a p-value of 0.22, which means that there is no significant difference between any setting.

Optimizer	metric	η	iterations	Worst score	Mean Dice score
BOHB	1-Dice	3	5	0.691	0.714
BOHB	1-Dice	2	5	0.691	0.713
BOHB	1-Dice	3	10	0.691	0.715
BOHB	1-Dice	2	10	0.708	0.718
BOHB	DiceCELoss	3	5	0.691	0.714
BOHB	DiceCELoss	2	5	0.704	0.715
BOHB	DiceCELoss	3	10	0.714	0.718
BOHB	DiceCELoss	2	10	0.691	0.715
Random Search	-	-	19	0.700	0.715
Random Search	-	-	38	0.714	0.719

Table 4.2: The mean (N=10) and worst Dice score achieved by the different BOHB and Random search settings. The theoretical (grid search) optimum is 0.719.

4.3. Comparing AutoMONAI with nnU-net

In this section, we compare the performance of AutoMONAI with the state-of-the-art, nnU-net. The Dice score achieved on the test set is shown in Table 4.3. This table shows that AutoMONAI performs slightly worse than nnU-net in two out of three classes (0.77 vs 0.84 and 0.56 vs 0.62) the designs had to segment. Only for the last class, the enhancing tumor, both designs perform similarly (0.79 and 0.80). AutoMONAI, which uses 2D convolutions, even slightly outperforms the 2D variant of nnU-net in this last class (0.77).

Algorithm	Brain dataset		
	1	2	3
nnU-net 3D	0.8391	0.6192	0.7959
nnU-net 2D	0.7860	0.5865	0.7742
AutoMONAI	0.7721	0.5646	0.7925

Table 4.3: The Dice score on the different segmentation classes (1: edema, 2: non-enhancing tumor and 3: enhancing tumor) of the Brain dataset. Reported data from nnU-net 2D network is from [25].

Figure 4.2 shows the loss curves of nnU-net and the best configuration in our framework. There are a few differences between the two graphs which do not reflect a difference in performance. The loss curve calculated in nnU-net is a constant 1.00 lower than calculated in AutoMONAI. This does not effect the performance, since the gradient is the same. Furthermore, our framework only calculates the Dice score and validation loss every 5 epochs, to reduce computation cost.

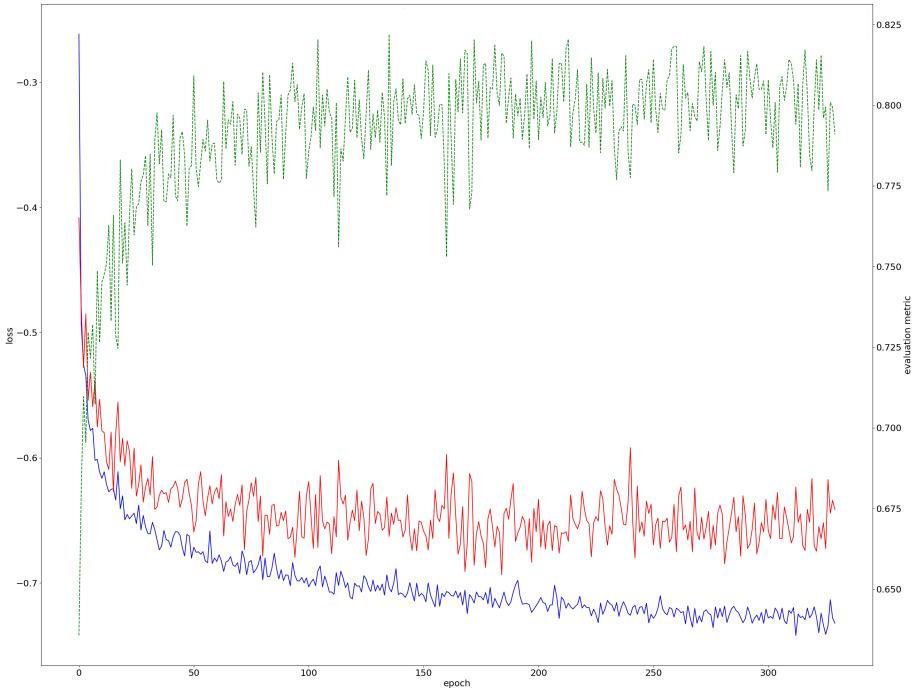
A noticeable difference is that nnU-net converges quicker and to a higher Dice score (0.8 vs 0.7), unfortunately slightly hidden by the different axes for the Dice score. However, the loss curve of AutoMONAI converges to a lower final loss, when corrected for the shift explained in the previous paragraph. Another interesting detail is the sudden drop in the loss of AutoMONAI around 25 epochs, where the loss curve of nnU-net drops more gradually. Our model has a validation loss curve that is more similar to the training loss curve than nnU-net, implying AutoMONAI has less overfitting than nnU-net.

The best configuration found by BOHB is shown in Table 4.4. This configuration shows similarities with the previous experiments. For example, BOHB best configuration has 2D convolutions, which was also substantially better than 3D convolutions in the grid search as shown in Section 4.2.

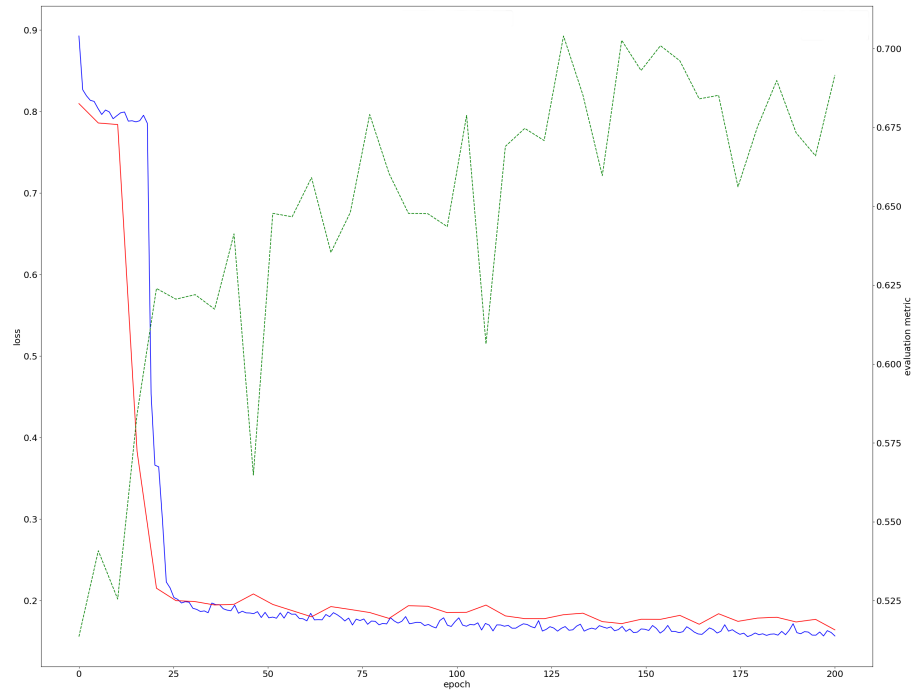
Hyperparameter	Value
Depth	3
Convolution layers	1
Channels first layer	2^6
Channel multiplier	2.05
Convolution type	2D
Normalization	Percentile clipping
Resampling	None
Loss function	DiceTopK

Table 4.4: Best configuration found by BOHB.

The boxplot shown in Figure 4.3 shows the median and lower and upper quartile of the Dice for each class, which is calculated on the test set. The Figure shows that the Dice score for the second class is lower and has a larger variance, which is common for the Brain dataset [2]. Both nnU-net and AutoMONAI achieve similar results on the second class, the non-enhancing tumor. Interestingly enough, the median of the Dice score achieved by nnU-net on the third class is substantially higher, while the mean is similar.

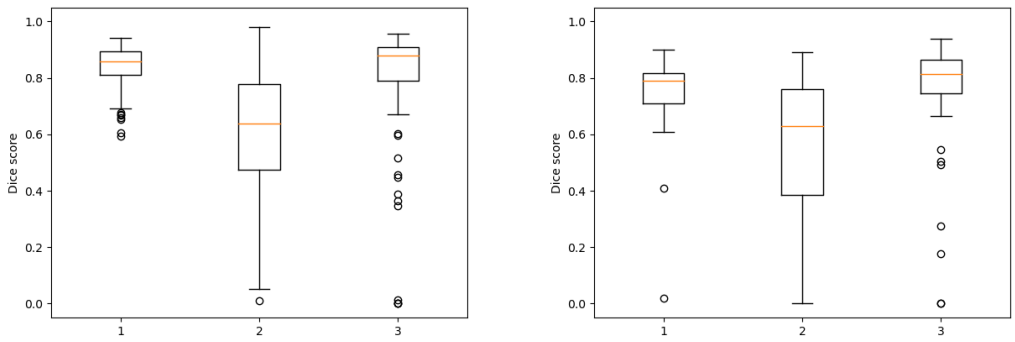


(a) nnU-net's loss curve.



(b) AutoMONAI's loss curve.

Figure 4.2: Loss and performance curves of AutoMONAI and nnU-net of a single fold on the Brain dataset. The green curve is the Dice score, with the axis on the right of the figure, which is different for both figures. The loss curves (Blue for the train loss, red for the validation loss) have the axis on the left side. The loss of nnU-net is exactly 1 lower due to a difference in calculation.



(a) nnU-net (b) AutoMONAI

Figure 4.3: Boxplots of nnU-net and AutoMONAI for each class (1: edema, 2: non-enhancing tumor and 3: enhancing tumor) of the Brain dataset based on the test set. The box extends from the lower to upper quartile of the data, with an orange line for the median.

5 Discussion

This chapter summarizes the results (Chapter 4) and interprets them to answer the research questions: (1) What is a viable search space? (Section 5.1), (2) Is BOHB a useful HPO optimizer in this use case? (Section 5.2) and (3) Does AutoML perform better than the state-of-the-art? (Section 5.3). Finally, we discuss limitations of this work (Section 5.4).

5.1. Determining a viable search space

The number of convolutions in the first layer, the multiplication factor, the convolution type, and the loss function significantly influence the performance. The depth, the number of convolutions, resampling, and normalization do not influence the performance significantly; we will discuss the potential reasons here. Surprisingly, 2D networks outperform 3D networks in our setting, while literature suggests that 3D networks should perform better [25].

Resampling does not change the Brain dataset, since it is already resampled to have the same voxel size. Differences in configuration do therefore not result in different computations or performance. The difference between normalization procedures is not significant, likely because the network itself normalizes after each convolution. An indication in this direction is the fact that the configuration without any normalization slightly trails behind at the start of the training procedure. Assuming normalization is beneficial, this configuration learns to rely on the first normalization in the network. One study [51] showed that normalization is beneficial in general, however, it also showed that the effect of normalization diminishes for more difficult problems, larger samples sizes, and larger networks. these effects occur when the previously mentioned effect occurs, i.e. when networks are capable of self-scaling.

Increasing the depth or the number of convolutions should theoretically not have a negative impact on the performance, since a convolution can be an Identity operation and the outputs of the additional depth layers can simply be weighted zero. However, increasing these hyperparameters does increase memory usage, which is limited. This means these hyperparameters influence the search range of other, significant hyperparameters. For example, a higher depth lowers the upper bound of the number of channels without reaching these limits. The same is true the other way around: by reducing the memory footprint with regards to insignificant hyperparameters, an optimizer can use the freed memory to expand the range on a more significant hyperparameter.

Based on these one-dimensional searches, a search space should include at least the convolution type, the number of channels in the first layer, its multiplication factor between depth layers, and the loss function, since optimizing those has a significant effect on the performance. We think it is also useful to include the depth and the number of convolutions since these hyperparameters influence the range of other hyperparameters. The effect of resampling needs to be investigated on a dataset where the operation has an effect. Normalization, however, can probably be omitted to make room for other, not investigated hyperparameters.

5.2. Investigating the applicability of BOHB

The analysis using Kendall's Tau shows that the ranking after 5 epochs can be used as a rough estimator for the eventual ranking. However, it also shows that the final ranking is influenced by noise, since Kendall's tau does not reach 1, but settles between 0.7 and 0.8. Repeating the experiment or extending the number of epochs, will result in a different final ranking. This noise is caused by small perturbations in the weights and the small performance differences.

Differences in performances between the different configurations of BOHB and Random Search are not significant when applied on the used small search space (which has only 50 possible configurations). Even though this could be explained by insufficient statistical power or simply no significant difference between the methods, even when the search space is sufficiently large, this is not likely. We used a substantially larger sample size (n=10) than the recommended lower bound (n=5) and the literature [16] shows there is a difference between these methods when used to optimize neural networks for natural images. Therefore, the most likely reason is the small search space. Since there are only 50 total different options, a Random search with 38 iterations has a $1 - (49/50)^{38} = 0.53$ chance to find the best solution. BOHB with 10 iterations can potentially explore 128 different configurations, although many of those only with a small budget. However, if the number of iterations becomes much lower, it acts as a random search combined with HyperBand, since it cannot build a useful Bayesian model. Therefore, with such a small search space, the methods largely overlap and differences can occur when these methods are applied to a larger search space.

5.3. Comparison with nnU-net

The results show that our framework does not perform as well as the state-of-the-art in two out of three classes, and has similar performance in the last class. This performance difference may be attributed to several reasons, which we discuss in the next paragraph. However, we also found that AutoMONAI has less overfitting, since the training and validation loss curve are nearly identical, whereas these loss curves measured at nnU-net have larger differences.

First of all, the train-test splits were not the same for nnU-net and our framework. This might contribute to these differences, however, it might also partly compensate for a larger difference. More importantly, we think the difference in data augmentation between our designs is the largest contributor to the performance gap. We skip rotations and scaling in the data augmentation process, due to unfortunate problems in the implementation. Other differences, such as the different learning rate scheduler or the sampling of training patches can also influence performance. Another difference between the implementation is the sampling of 2D patches. Whereas nnU-net samples 2D patches, we resize 3D patches to 2D patches. This does not sample all 2D patches equally likely and creates a large batch size of $2 * 128$. Finally, there are differences in the stopping criteria and the learning rate scheduler. Whereas AutoMONAI stops training after 200 epochs, nnU-net stops training when there is little improvement and the learning rate is low. This is also influenced by both learning rate schedulers, which reduces the learning rate in nnU-nets implementation with much lower criteria than in AutoMONAIs implementation. Assuming the loss is close to an optimum, nnU-net reduces the learning rate when the performance improvement is less than 1×10^{-3}, whereas AutoMONAI reduces the learning rate when the performance improvement is less than only 2×10^{-5}.

5.4. Limitations and future work

This section discusses the limitations of this work and proposes future work. As mentioned in the previous paragraph, we use a less extensive data augmentation process, and we think this negatively impacts the performance. While we do not expect this potential improvement would let AutoMONAI surpass nnU-net in performance, it should be addressed in future iterations of our framework.

This work is also limited because we have done all of our experiments on a single dataset. This has consequences for the interpretation of the conclusions. All of our conclusions can only be applied to MSD's Brain Dataset. For example, as mentioned in the paragraphs above, the hyperparameter resampling does not have a significant influence on the performance attributed to the fact that the experiment ran on this dataset. Since AutoML is applied to many datasets, it is important to note that our conclusions can only be applied to this single dataset. We suggest future work applies this

framework to the other datasets of the MSD or other datasets in medical image segmentation.

Finally, due to the heavy computational costs of our framework and experiments, we have been limited in validating our results. Not only were we limited in applying AutoMONAI to multiple datasets (as mentioned above), we also have not completely done five-fold test-train cross-validation. We have trained 4 networks which ensemble evaluated the test set. It would have been more accurate to train 4 networks for each fold and report the performance of each ensemble on the corresponding test set. It has also limited us in the size of the used search space and comparing BOHB and RS on the complete search space.

One way to reduce the effect of these computational costs is by parallelizing BOHB, which is what we have left for future work. This is one of the major advantages of BOHB compared to Bayesian Optimization methods such as SMAC or TPE [24]. By implementing this feature, it becomes possible to increase the search space accordingly, compare the performance achieved by different search spaces or use more iterations for BOHB, to ensure the algorithm is converged.

Another potential future research direction that reduces the time BOHB needs to converge, is using Meta-Learning. BOHB uses a model with the highest fidelity to sample new configurations and start without a model, where it uses random sampling. This random sampling can be, for example, be replaced by a Meta-Model. One of this Meta-models [7] discretizes the configurations investigated in previous tasks and selects the top-K configurations of this discretized configurations.

Furthermore, the performance of AutoMONAI may be improved by ensembling the top-performing configurations found by BOHB instead of the single best configuration. nnU-net does something similar; it ensembles every combination of 2D, 3D, and Cascade-3D (when applicable) networks and selects the best combination afterward. Since the performance differences between configurations found by BOHB are very small, an ensemble can improve performance. However, we assume that these configurations need to be sufficiently diverse, which might be difficult to determine.

Finally, instead of focusing on improving performance or runtime, future work could find solutions on a Pareto front, where performance is traded against other objectives, such as computational costs or the explainability of the returned networks. The explainability of any automated system can increase trust in the system. Various papers have been published on the explainability of CNN's [63], with several techniques such as class saliency, activation maps, and guided backpropagation to explain semantic segmentation. Explainability of the returned models is based on the complexity, for example, by adding layers, more activation maps are presented to the reviewer of the models, which might make it more difficult to explain.

6

Conclusion

The goal of this thesis was to investigate the possibilities of implementing BOHB for a medical segmentation problem, which resulted in our framework AutoMONAI. We have investigated several hyperparameters to be included in a search space, compared BOHB with RS, and finally compared our approach with the state-of-the-art. These experiments found several of the mentioned hyperparameters to significantly influence the performance. We have also found that that the ranking after five epochs is largely correlated with the ranking after 200 epochs. This indicates there is a correlation between the ranking after five epochs and the final ranking. However, these experiments also showed that the final ranking is influenced by noise since the correlation between the one before last ranking and the last ranking is large, but these rankings are not completely equal. This finding shows that the prerequisite of BOHB is valid, but we did not find a significant difference between any setting of BOHB or RS on the small search space, which optimized the depth, number of channels, and convolution type. Finally, our framework achieves a similar performance as nnU-net on MSD's Brain dataset.

A

Loss curves of 1D searches

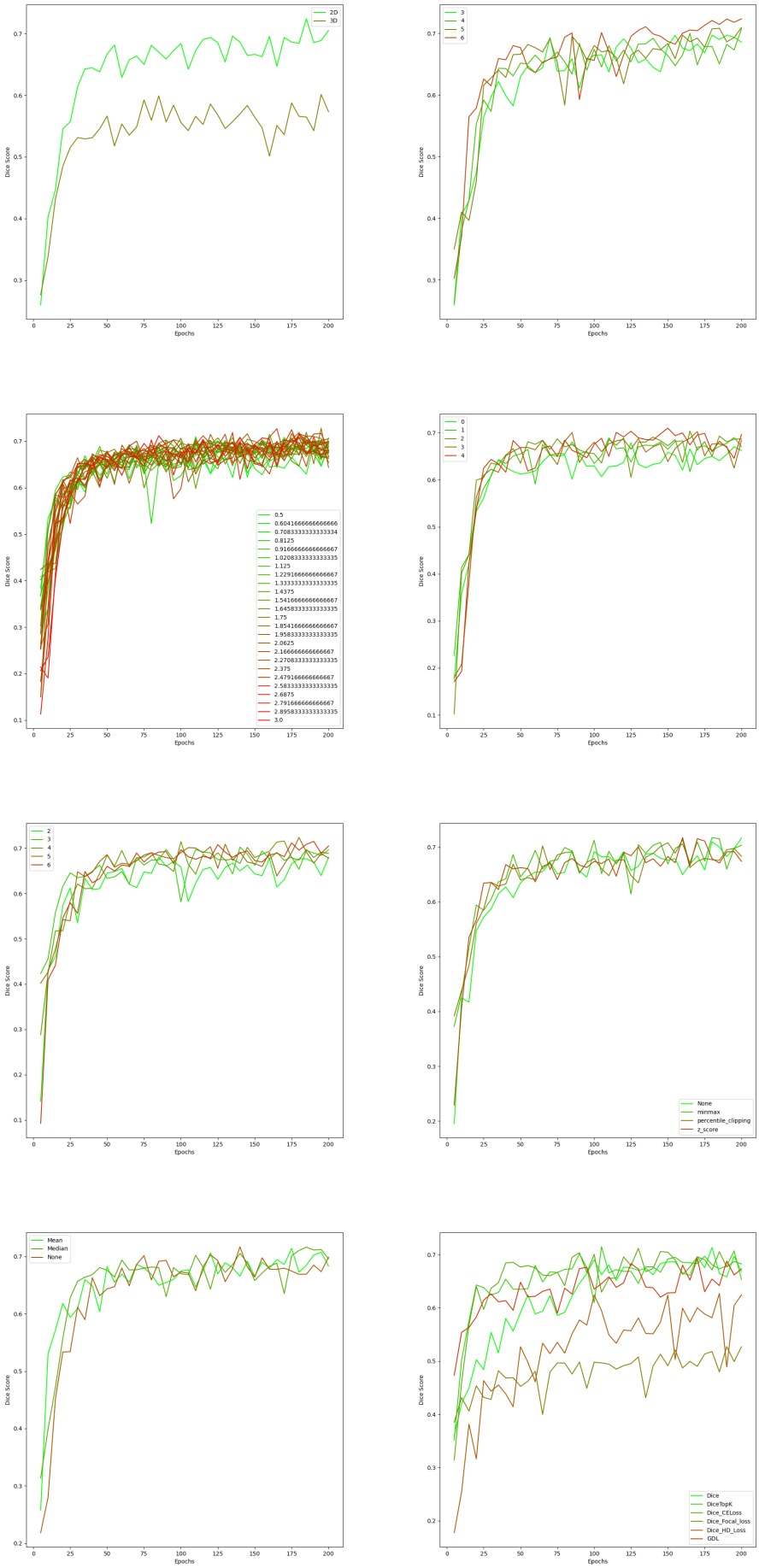


Figure A.1: The Dice score curves, with a single graph for each hyperparameter. From left to right, top to bottom: Convolution type, channels in the first layer, channel multiplier, convolutions in each layer, the depth, normalization, resampling and the loss function.

Bibliography

[1] Fouzia Altaf et al. “Going deep in medical image analysis: concepts, methods, challenges, and future directions”. In: *IEEE Access* 7 (2019), pp. 99540–99572.

[2] Michela Antonelli et al. “The Medical Segmentation Decathlon”. In: *arXiv preprint arXiv:2106.05735* (2021).

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. “Segnet: A deep convolutional encoder-decoder architecture for image segmentation”. In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[4] James Bergstra et al. “Algorithms for hyper-parameter optimization”. In: *25th annual conference on neural information processing systems (NIPS 2011)*. Vol. 24. Neural Information Processing Systems Foundation. 2011.

[5] Patrick Bilic et al. “The liver tumor segmentation benchmark (lits)”. In: *arXiv preprint arXiv:1901.04056* (2019).

[6] Daniel Bolya et al. “Yolact: Real-time instance segmentation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9157–9166.

[7] Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. “Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results”. In: *Machine Learning* 50.3 (2003), pp. 251–277.

[8] Özgün Çiçek et al. “3D U-Net: learning dense volumetric segmentation from sparse annotation”. In: *International conference on medical image computing and computer-assisted intervention*. Springer. 2016, pp. 424–432.

[9] Marco Domenico Cirillo, David Abramian, and Anders Eklund. “Vox2Vox: 3D-GAN for brain tumour segmentation”. In: *arXiv preprint arXiv:2003.13653* (2020).

[10] MONAI Consortium. *MONAI: Medical Open Network for AI*. Version 0.6.0. If you use this software, please cite it using these metadata. Mar. 2020. DOI: 10.5281/zenodo.5525502. URL: https://doi.org/10.5281/zenodo.5525502.

[11] Ekin D Cubuk et al. “Autoaugment: Learning augmentation strategies from data”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 113–123.

[12] Jifeng Dai et al. “Deformable convolutional networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 764–773.

[13] Casey Davis and Christophe Giraud-Carrier. “Annotative experts for hyperparameter selection”. In: *AutoML Workshop at ICML*. 2018.

[14] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[15] Lee R Dice. “Measures of the amount of ecologic association between species”. In: *Ecology* 26.3 (1945), pp. 297–302.

[16] Stefan Falkner, Aaron Klein, and Frank Hutter. “BOHB: Robust and efficient hyperparameter optimization at scale”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1437–1446.

[17] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[18] Kaiming He et al. *Deep residual learning for image recognition. CoRR abs/1512.03385 (2015)*. 2015.

[19] Xin He, Kaiyong Zhao, and Xiaowen Chu. “AutoML: A Survey of the State-of-the-Art”. In: *Knowledge-Based Systems* 212 (2021), p. 106622.

[20] Mattias P Heinrich, Ozan Oktay, and Nassim Bouteldja. “OBELISK-Net: Fewer layers to solve 3D multi-organ segmentation with sparse deformable convolutions”. In: *Medical image analysis* 54 (2019), pp. 1–9.

[21] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[22] Saddam Hussain, Syed Anwar, and Muhammad Majid. “Segmentation of Glioma Tumors in Brain Using Deep Convolutional Neural Network”. In: *Neurocomputing* 282 (Dec. 2017). DOI: 10.1016/j.neucom.2017.12.032.

[23] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. “Comparing images using the Hausdorff distance”. In: *IEEE Transactions on pattern analysis and machine intelligence* 15.9 (1993), pp. 850–863.

[24] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[25] Fabian Isensee et al. “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation”. In: *Nature* 18.2 (2021), pp. 203–211.

[26] Bonnie N Joe et al. “Brain tumor volume measurement: comparison of manual and semiautomated methods”. In: *Radiology* 212.3 (1999), pp. 811–816.

[27] Zohar Karnin, Tomer Koren, and Oren Somekh. “Almost Optimal Exploration in Multi-Armed Bandits”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1238–1246.

[28] Maurice G Kendall. “A new measure of rank correlation”. In: *Biometrika* 30.1/2 (1938), pp. 81–93.

[29] Hoel Kervadec et al. “Boundary loss for highly unbalanced segmentation”. In: *International conference on medical imaging with deep learning*. PMLR. 2019, pp. 285–296.

[30] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[31] William H Kruskal and W Allen Wallis. “Use of ranks in one-criterion variance analysis”. In: *Journal of the American statistical Association* 47.260 (1952), pp. 583–621.

[32] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[33] Lisha Li et al. “Hyperband: A novel bandit-based approach to hyperparameter optimization”. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765–6816.

[34] Tsung-Yi Lin et al. “Focal loss for dense object detection”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[35] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[36] William P Lincoln and Josef Skrzypek. “Synergy of clustering multiple back propagation networks”. In: *Advances in neural information processing systems*. 1990, pp. 650–657.

[37] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “Darts: Differentiable architecture search”. In: *arXiv preprint arXiv:1806.09055* (2018).

[38] Jun Ma et al. “Loss odyssey in medical image segmentation”. In: *Medical Image Analysis* (2021), p. 102035.

[39] Bjoern H Menze et al. “The multimodal brain tumor image segmentation benchmark (BRATS)”. In: *IEEE transactions on medical imaging* 34.10 (2014), pp. 1993–2024.

[40] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. “V-net: Fully convolutional neural networks for volumetric medical image segmentation”. In: *2016 fourth international conference on 3D vision (3DV)*. IEEE. 2016, pp. 565–571.

[41] Andriy Myronenko. “3D MRI brain tumor segmentation using autoencoder regularization”. In: *International MICCAI Brainlesion Workshop*. Springer. 2018, pp. 311–320.

[42] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[43] S Patro and Kishore Kumar Sahu. “Normalization: A preprocessing stage”. In: *arXiv preprint arXiv:1503.06462* (2015).

[44] Luis Perez and Jason Wang. “The effectiveness of data augmentation in image classification using deep learning”. In: *arXiv preprint arXiv:1712.04621* (2017).

[45] Mathias Perslev et al. “One network to segment them all: A general, lightweight system for accurate 3d medical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2019, pp. 30–38.

[46] Carl Edward Rasmussen. “Gaussian processes in machine learning”. In: *Summer school on machine learning*. Springer. 2003, pp. 63–71.

[47] Joseph Redmon et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[48] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *arXiv preprint arXiv:1506.01497* (2015).

[49] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[50] Dominik Scherer, Andreas Müller, and Sven Behnke. “Evaluation of pooling operations in convolutional architectures for object recognition”. In: *International conference on artificial neural networks*. Springer. 2010, pp. 92–101.

[51] Murali Shanker, Michael Y Hu, and Ming S Hung. “Effect of data standardization on neural network training”. In: *Omega* 24.4 (1996), pp. 385–397.

[52] Connor Shorten and Taghi M Khoshgoftaar. “A survey on image data augmentation for deep learning”. In: *Journal of Big Data* 6.1 (2019), pp. 1–48.

[53] Dalwinder Singh and Birmohan Singh. “Investigating the impact of data normalization on classification performance”. In: *Applied Soft Computing* 97 (2020), p. 105524.

[54] Carole H Sudre et al. “Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations”. In: *Deep learning in medical image analysis and multimodal learning for clinical decision support*. Springer, 2017, pp. 240–248.

[55] Giles Wesley Vick III. “The gold standard for noninvasive imaging in coronary heart disease: magnetic resonance imaging”. In: *Current opinion in cardiology* 24.6 (2009), pp. 567–579.

[56] Sun-Chong Wang. “Artificial neural network”. In: *Interdisciplinary computing in java programming*. Springer, 2003, pp. 81–100.

[57] Tao Wei et al. “Network morphism”. In: *International Conference on Machine Learning*. PMLR. 2016, pp. 564–572.

[58] Yingda Xia et al. “3d semi-supervised learning with uncertainty-aware multi-view co-training”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 3646–3655.

[59] Wenjun Yan et al. “The domain shift problem of medical image segmentation and vendor-adaptation by Unet-GAN”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2019, pp. 623–631.

[60] Jiancheng Yang et al. “Reinventing 2D convolutions for 3D images”. In: *IEEE Journal of Biomedical and Health Informatics* (2021).

[61] Zhaohui Yang et al. “Cars: Continuous evolution for efficient neural architecture search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1829–1838.

[62] Qihang Yu et al. “C2fnas: Coarse-to-fine neural architecture search for 3d medical image segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 4126–4135.

[63] Quanshi Zhang and Song-Chun Zhu. “Visual interpretability for deep learning: a survey”. In: *arXiv preprint arXiv:1802.00614* (2018).

[64] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. “Ensembling neural networks: many could be better than all”. In: *Artificial intelligence* 137.1-2 (2002), pp. 239–263.

[65] Zhuotun Zhu et al. “V-nas: Neural architecture search for volumetric medical image segmentation”. In: *2019 International Conference on 3D Vision (3DV)*. IEEE. 2019, pp. 240–248.

[66] Barret Zoph et al. “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.