# Hyperparameter Optimization of Deep Learning Models for Biomedical Image Segmentation

---

A

Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Doctor of Philosophy

by

William Adorno III

August 2021

# APPROVAL SHEET

This

Dissertation

is submitted in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

Author: William Adorno III

This Dissertation has been read and approved by the examing committee:

Advisor: Donald Brown

Advisor:

Committee Member: Laura Barnes

Committee Member: Peter Beling

Committee Member: Sana Syed

Committee Member: Marcel Durieux

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Jennifer L. West, School of Engineering and Applied Science

August 2021

Hyperparameter Optimization of Deep Learning Models for Biomedical Image Segmentation

Abstract

Hyperparameter Optimization of Deep Learning Models for Biomedical Image Segmentation

by

William Adorno III

Doctor of Philosophy in Systems and Information Engineering

University of Virginia

Deep image segmentation of biomedical images is growing in prevalence as an essential tool for extracting and analyzing clinical data. Like other machine learning techniques, deep segmentation models contain hyperparameters that can have a significant impact on prediction performance. A considerable amount of time and computational resources are required to tune hyperparameters, so researchers often settle on settings from previous experience. In this dissertation, the performance and feasibility of multiple hyperparameter optimization techniques are evaluated to prove the efficacy of these methods. The evaluated techniques include random search, Bayesian optimization, multi-armed bandit, and two novel approaches. The first novel approach, Random Search with Statistical Reduction (RSSR), was designed to deal with optimization trials runs that contain bimodal-distributed response data. RSSR also enables the inclusion of a user's utility function within the non-parametric statistical testing to reduce the settings space. The second novel approach, Gaussian Mixture with Epsilon Greedy, was designed to ensure continued exploration of the hyperparameter space during long optimization runs.

To evaluate these approaches, three varying biomedical image segmentation datasets are utilized to foster a robust comparison. These datasets include: eosinophil detection in Eosinophilic Esophagitis patients, multiple cell type detection in 3D cardiovascular florescent microscopy, and handwritten vital sign detection in surgical flowsheet graphs. The results from the evaluation revealed that the Gaussian process-based Bayesian optimization consistently produced higher validation set accuracy when computation time is limited and was also the most Pareto efficient option. When ample computation time is available, the RSSR approach is able to find high validation accuracy by effectively reducing the search space. Several hyperparameters such as batch normalization, learning rate, and batch size were proven as crucial in reaching minimum validation loss. With these findings, researchers will be more encouraged to perform hyperparameter optimization on real-world image segmentation problems and will know the most effective techniques to execute the process depending on their situation.

To Jennifer, Liam, and Darcie.

# Contents

**Contents** **ii**

**List of Figures** **iv**

**List of Tables** **viii**

**1 Introduction** **1**
1.1 Biomedical Image Segmentation . . . . . 1
1.2 Hyperparameter Optimization . . . . . 2
1.3 Problem Statement . . . . . 2
1.4 Dissertation Overview . . . . . 4

**2 Literature Review** **6**
2.1 Convolutional Neural Networks . . . . . 6
2.2 Deep Image Segmentation . . . . . 8
2.3 Hyperparameter Optimization Techniques . . . . . 9

**3 Segmentation Datasets** **12**
3.1 Eosinophilic Esophagitis Biopsy Images . . . . . 12
3.2 Cardiovascular Immunofluorescent 3D Images . . . . . 15
3.3 Vital Signs Graph from Surgical Flowsheet . . . . . 18

**4 Methodology** **21**
4.1 Experimental Setup . . . . . 21
4.2 Tested Hyperparameters . . . . . 23
4.3 Existing Optimizations . . . . . 24
4.4 Random Search with Statistical Reduction (RSSR) . . . . . 25
4.5 Gaussian Mixture with Epsilon-Greedy (GMEG) . . . . . 33

**5 Results** **37**
5.1 Comparison of Optimization Techniques . . . . . 37
5.2 Analysis by Hyperparameter . . . . . 44
5.3 Pareto Efficiency of Optimizations . . . . . 48

**6 Conclusion and Future Work** **54**

**Bibliography** **57**

# List of Figures

3.1 This figure shows the process of converting a biopsy sample into digitized whole-slide image and then split into many 512 × 512 pixel patches. . . . . 13
3.2 This figure shows examples of patch-level annotations of eosinophils on EoE biopsy patches. The eosinophils were outlined by trained experts on specialized software and then converted into ground-truth segmentation masks. . . . . 13
3.3 This figure shows three different examples of examining linkages between patient-level eosinophil count statistics at initial biopsy with a patients' phenotypes. For instance, the left-most chart shows the the 4 to 6 Food Elimination Diet (Diet . 14
3.4 This figure shows an example at a single $Z$-level of the two different annotation strategies used to represent cell types within the cardiovascular immunofluorescent images. On the left, the nuclei cells are marked with red arrows, the SMCs are annotated with ellipses, and the macrophages are annotated with circles. On the right, only nuclei cells are outlined in the traditional annotation format. These annotations correspond with the blue fluorescent antibody stain and the red arrow markings on the left image. . . . . 16
3.5 This figure shows an example of one patch of a cardiovascular immunofluorescent image. The top row shows each of cell type channels with maximum intensity projection of the $Z$-axis applied. The colors refer to each antibody stain. The bottom row shows segmentation masks for annotation strategy that most accurately counted the total number of cells. . . . . 17
3.6 This figure shows an example of a paper flowsheet that includes handwritten perioperative data. The flowsheet includes sections such as the medications administered, surgical info checkboxes, and the graph to track vital signs. The vital signs graph is outlined in red. . . . . 18
3.7 This figure shows an example of a vital signs graph that has been converted into digitized numerical values. On the left, is a section of the vital signs graph from a perioperative flowsheet with the symbols manually recorded by hand. On the right, is an example of the final end result in converting of all the symbols on the graph into tabular data for further analysis. . . . . 19

3.8 This figure shows an example of the annotations performed on vital signs graphs. The top image is an example vital signs graph. The bottom image is annotations for each of the three symbols. The systolic BPs are blue squares, the heart rates are green dots, and the diastolic BPs are red squares. Each symbol is separated into different segmentation masks to train separate models. . . . . 20

4.1 This figure shows the validation Dice results for systolic blood pressure symbol detection. A total of 147 trial results were obtained in a 2-hour random search optimization. The left chart shows a boxplot of the validation Dice. The middle chart shows a histogram of the same response. It is clear from these two charts the data is bimodal. The chart on the right shows the two Gaussian distributions that were fit to this data and the intersection between them. This intersection splits good trial runs from bad trials. . . . . 27

4.2 This figure shows the validation Dice results for systolic blood pressure symbol detection by batch normalization. The left chart contains all 147 trials, while the right chart contains the 34 good trials. It appears that batch normalization contributes to higher validation Dice and future random search trials should always contain batch normalization. . . . . 28

4.3 This figure shows the validation Dice results for systolic blood pressure symbol detection by Adam optimizer learning rate. The left chart contains all 147 trials, while the right chart contains the 34 good trials. It appears that learning rates of $2 \times 10^{-2}$, $2 \times 10^{-3}$, and $2 \times 10^{-4}$ contributes to higher validation Dice and future random search trials should consider only testing these settings . . . . . 29

4.4 This figure shows the validation Dice results for systolic blood pressure symbol detection by continuous dropout rate. The chart on the left shows all 147 trials, while the chart on the right is only the 34 good trials. The right chart shows a minor correlation of higher dropout rates producing higher validation Dice results. 31

4.5 This figure shows the validation Dice results for systolic blood pressure symbol detection by categorical dropout rate. By converting into categorical, the established statistical comparison methods can be implemented. The right chart still shows a trend with higher dropout rate, but the sample sizes can be incorporated to determine if this trend is statistically significant. . . . . 32

4.6 This figure shows the validation Dice results for systolic blood pressure symbol detection by continuous dropout rate. The colored ellipses show where the seven Gaussian clusters are located with results from all completed trial runs. The top-right cluster has the highest mean validation Dice, so the greedy action is to sample from this distribution. The exploitation dropout rate will likely be in the 0.3 to 0.4 range using a multi-variate normal random number generator. This methodology was also flexible for handling bad runs as about 5 clusters formed in the low response area. . . . . 34

4.7 This figure shows the validation Dice results for systolic blood pressure symbol detection by continuous dropout rate. The colored ellipses show where the 12 Gaussian clusters were located with results from all completed trial runs. it appears the top-left cluster was now the greedy option. . . . . 36

5.1 Performance Results by EoE Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that GP-UCB and GMEG 1 were best-suited for the EoE dataset. . . . . 39
5.2 Performance Results by Cardiovascular Nuclei Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that Hyperband and RSSR were best-suited for the CV Nuclei dataset. . . . . 39
5.3 Performance Results by Cardiovascular SMC Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that random search and RSSR were best-suited for the CV SMC dataset. . . . . 40
5.4 Performance Results by Cardiovascular Macrophage Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that random search and RSSR were best-suited for the CV macrophage dataset. . . . . 40
5.5 Performance Results by Vital Signs Graph Heart Rate Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that RSSR and GP-UCB were best-suited for the GR HR dataset. . . . 41
5.6 Performance Results by Vital Signs Graph Diastolic BP Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that RSSR and RS were best-suited for the GR DBP dataset. . . . . 41
5.7 Performance Results by Vital Signs Graph Systolic BP Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that RSSR was best-suited for the GR DBP dataset. . . . . 42
5.8 Batch Normalization Selection as the Best Setting. It was clear that batch normalization should be set to True and remain active throughout the neural network architecture for all datasets. . . . . 46
5.9 Learning Rates Selection as the Best Setting. Learning rates of $2 \times 10^{-2}$, $2 \times 10^{-3}$, and $2 \times 10^{43}$ were the most popular, but there were some trends for which of these the datasets prefer. . . . . 47
5.10 Number of Initial Filters Selection as the Best Setting. The middle number of filter option, 24, was selected the most, but the overall distribution of selected setting was fairly flat. This does show that more filters was not always better for model generalization. . . . . 47
5.11 Batch Size Selection as the Best Setting. Batch size appeared highly dependent on which dataset was being used. The GR data greatly preferred a batch size of 2, while the others fluctuated between 4 or 8. . . . . 48

5.12 Dropout Rate Select as the Best Setting. No particular dropout rate appeared to correspond with higher performance, but there were a few datasets that seemed to prefer higher dropout rates. . . . . 49
5.13 EoE Dataset Pareto Efficiency of Optimizations. This figure shows a scatterplot of the maximum validation Dice scores versus the time these scores occurred during each each optimization. The points in the upper-left were the most Pareto efficient. GP-UCB and GMEG 1 were both efficient on the EoE data. . . . . 50
5.14 CV Nuclei Dataset Pareto Efficiency of Optimizations. This figure shows a scatterplot of the maximum validation Dice scores versus the time these scores occurred during each each optimization. The points in the upper-left were the most Pareto efficient. GMEG 1 appeared most efficient on the CV nuclei data. . . . . 50
5.15 CV SMC Dataset Pareto Efficiency of Optimizations. The RS optimization appears efficient on this dataset, but this was likely due to random chance of testing a near-optimal setting as one of the early trials. . . . . 51
5.16 CV Macrophage Dataset Pareto Efficiency of Optimizations. The RS optimization appears efficient on this dataset, but this was likely due to random chance of testing a near-optimal setting as one of the early trials. . . . . 52
5.17 Vital Signs HR Dataset Pareto Efficiency of Optimizations. The GP-UCB appeared as the most efficient on the graph reading HR dataset. . . . . 52
5.18 Vital Signs Diastolic BP Dataset Pareto Efficiency of Optimizations. The RS and GP-UCB optimizations appear as the most Pareto efficient on this dataset. . . . 53
5.19 Vital Signs Systolic BP Dataset Pareto Efficiency of Optimizations. The GP-UCB optimization was the most Pareto efficient on this dataset. . . . . 53

# List of Tables

3.1 Training and validation set patch counts for each cell type. . . . . 16

4.1 This table shows the U-Net architecture used throughout all the experiments. The left side of the table was the contracting side of the U-Net (processes downwards). The right side was the expansion side (processes upwards in the table). The variables in the tensor output size refer to: H = image height, W = image width, C = number of input channel, and F = number of initial filters. . . . . 22
4.2 This table shows the statistical comparison results of learning rate for the five possible settings on all trials. An entry receives a -1 or 1 if the distributions are statistically different according to a Mann-Whitney U test. A -1 or 1 is assigned based on if the mean of the row entry is less than or greater than the column entry. A learning rate of $2 \times 10^{-5}$ was significantly lower in validation Dice performance than three other settings. . . . . 29
4.3 This table shows the statistical comparison results of learning rate for the five possible settings on only good trials. A learning rate of $2 \times 10^{-3}$ was significantly higher in validation Dice performance than three other settings. . . . . 30
4.4 This table combines the all trials and only good trial results for learning rate setting comparison by adding the two previous tables together. Now, both $2 \times 10^{-1}$, $2 \times 10^{-5}$ appear dominated by the other settings. . . . . 30
4.5 This table is the final step in the statistical comparison process. It sums down the columns of the previous table to create a metric for each setting that can be used to determine which should remain or be removed. A learning rate of $2 \times 10^{-3}$ appears to perform the best and should definitely remain, while $2 \times 10^{-1}$ and $2 \times 10^{-5}$ should be considered for removal. . . . . 31
4.6 This tables shows the final results of the statistical comparison between the five categorical dropout rates. Even though there did appear to be a minor dropout rate and validation Dice trend, it was not a strong of difference to warrant eliminating dropout settings for future random search testing. The safest conclusion is to continue to search the entire dropout rate range. . . . . 32

5.1 2-Hour Scenario Performance Comparison. This table shows the weighted ranking scores for each optimization technique. GP-UCB clearly dominates the other optimizations at the 2-hour scenario. . . . . 43

5.2 4-Hour Scenario Performance Comparison. This table shows the weighted ranking scores for each optimization technique. GP-UCB scored the best overall, but RS dominates the entire CV dataset. . . . . 43
5.3 8-Hour Scenario Performance Comparison. This table shows the weighted ranking scores for each optimization technique. RSSR scored the best overall, but the performance was very inconsistent across all datasets. . . . . 44
5.4 16-Hour Scenario Performance Comparison. This table shows the weighted ranking scores for each optimization technique. RSSR scored the best overall, but the performance was inconsistent across all datasets. . . . . 45
5.5 24-Hour Scenario Performance Comparison. This table shows the weighted ranking scores for each optimization technique. GP-UCB scored the best overall, but RS was also the best ranking for three of the seven datasets. . . . . 45

6.1 Comparison of User-established Hyperparameters with Best Hyperparameter Optimization Settings . . . . . 55

# Chapter 1

# Introduction

## 1.1 Biomedical Image Segmentation

The growth of Artificial Intelligence (AI) and Machine Learning (ML) has heavily impacted many fields and the biomedical domain is no exception [28]. Computer vision, in particular, has assisted in the diagnosis of diseases by finding delineating patterns and features in medical imaging such as biopsy tissue samples, magnetic resonance images, X-rays, ultrasounds, and fluorescent microscopy [48]. Traditional classification Convolutional Neural Networks (CNN) were the most well-known approach for biomedical image problems, but image segmentation models have emerged as another useful tool for the community [10]. The main advantage of image segmentation is the ability to detect and represent specific features within images. Classification CNNs and their black-box nature are mainly focused on prediction rather than discovery. Image segmentation allows medical researchers to target certain aspects of images that are known to have clinical significance and thus can lead to important findings and improved health outcomes.

Image segmentation is a supervised modeling approach, so annotated or labeled data is required to train, validate, and test the models. Depending on the particular problem, expert medical professionals such as histopathologists are utilized to annotate the locations of the desired features. The annotations are performed in specialized software that allow the user to draw outlines around objects at the pixel level. These annotations are typically stored as an XML or JSON file and then are processed and converted into an image form which is known as a segmentation mask. For semantic segmentation, the original images and the segmentation masks are the only inputs required to develop a model. Image segmentation model architectures such as the U-Net differ from classification CNNs by being fully-convolutional to ensure that the output prediction mask has the same resolution as the input image [37]. Like other deep learning models and CNNs, image segmentation models can take a considerable amount of time to train based on the size and resolution of available image datasets and the complexity of the model.

## 1.2 Hyperparameter Optimization

A common concern with developing almost any AI/ML project is selecting the model hyperparameters before training. Image segmentation models can contain millions of trainable parameters that are adjusted to detect desired features. Although, even one hyperparameter can greatly impact model performance or how well these parameters are trained. Some of the hyperparameters are directly involved with model training such as the optimizer learning rate, the number of images to process in each train set batch, and the proportion of nodes that are eliminated with a dropout layer approach. The hyperparameter problem can also include attributes of the neural network, but neural architecture search should be utilized for creation of the entire network [22]. The neural network attributes that can be optimized through hyperparameter optimization include how many layers are in the network, the number of nodes or filters per layer, and the kernel size of each filter.

Often, an ML researcher will begin using hyperparameter settings that worked well in previous experiences or have some basis in literature review. This can be problematic for several reasons. First, the researcher may not have experience tuning a particular model or is using a brand-new dataset. Second, while existing literature may provide acceptable recommendations for certain techniques or settings, there is no guarantee that the high-performance will translate to other datasets. Lastly, settings that are user-established may seem to train quickly and reach a satisfactory level of performance, but there could be other settings that lead to even more efficient training and optimal performance. For these reasons, hyperparameter optimization approaches are utilized to provide a structured method to find the most appropriate hyperparameter settings [27].

There are many different methods for hyperparameter optimization. If all of the settings are discrete, then an exhaustive search of every possible setting combination can be executed to find the optimal setting [9]. However, for obvious reasons, this strategy is infeasible for many deep learning researchers. There are often continuous variable settings involved in hyperparameter tuning such as dropout rate. The search space can become incredibly large as the number of hyperparameters increases which makes testing every combination impossible. Lastly, training deep learning models often requires a large amount of time and computational expense, so efficiency is critical when considering optimization techniques. Every ML researcher has varying requirements and capabilities regarding the amount of time and computational effort they can put towards hyperparameter optimization. Therefore, each modeling scenario may require a certain optimization technique that performs best in that situation.

## 1.3 Problem Statement

The goal of this dissertation was to evaluate the performance of hyperparameter optimization techniques on varying biomedical image segmentation datasets. When demonstrating optimization performance on computer vision problems with CNNs, these techniques were

usually evaluated on benchmark datasets such as MNIST and CIFAR-10 [9, 8]. These datasets are typically not representative of real-world image datasets, because the images are of very low-resolution [39]. Hyperparameters can be tuned on these benchmark datasets quickly due to the smaller image size. Also, these datasets contain many images, but hyperparameters can still be successfully tuned on smaller partitions of the full data. It was important to evaluate optimization techniques on real-world datasets to determine if they are feasible, worthwhile, and capable of improving performance in real applications.

It would not be appropriate to claim one optimization as best for all image segmentation problems if it was only evaluated on one real-world dataset. Every dataset has different characteristics that could lead to completely different hyperparameter settings and optimization paths. Therefore, this dissertation utilizes three different biomedical image projects to evaluate the optimization approaches. These three projects differ greatly in medical application, as well as, image appearance and characteristics. Two of the projects even feature three different types of annotations within the project to do further evaluation. This enables assessment of whether the optimization comparison and optimal hyperparameter settings are similar within the same dataset regardless of annotation. Applying hyperparameter optimization to each of these problem sets will improve prediction performance of each trained model which ultimately results in more accurate clinical information obtained from these images.

With an unlimited amount of time and computational resources, the actual technique used for hyperparameter optimization has less importance. In almost all real-world scenarios, ML researchers have time constraints and computational budgets, so there iss a range at which optimization technique selection can greatly benefit the user. The dissertation experiments covered time constraints that range from 2 to 24-hours of total computational runtime. Runtime was more effective at comparing models than the number of runs or trials, because certain techniques may generate more runs as part of the optimization strategy. A total of five different hyperparameter settings were optimized from areas that include model generalization, training efficiency, and neural architecture size.

In this study, hyperparameter optimization techniques were split into two broad categories: search and sequential. Search techniques such as grid or random search establish a list of settings to test and then those were evaluated until a certain amount of trials or time limit was reached. While these tested settings were typically spread-out over the design space, search techniques were not utilizing any information after each trial to test settings that were more beneficial to finding optimal performance. In contrast, Sequential Model-Based Optimization (SMBO) techniques determined the next setting to test after each trial is complete [32]. The purpose of most SMBO techniques is to find a balance between exploitation and exploration. Exploitation involves testing settings in areas that are currently high in performance to find the true global optima, while exploration is to ensure that model is not stuck in a local optima. The SMBO techniques evaluated include Bayesian optimization, an infinite-armed bandit, reinforcement learning, and sequential random search.

## 1.4 Dissertation Overview

The following is the overview of this dissertation. The next chapter is the literature review that details CNNs, image segmentation approaches, and hyperparameter optimization techniques. The third chapter introduces the three different biomedical image segmentation projects used to evaluate the optimization techniques. The fourth chapter explains the methodology of the experimental setup, as well as, discusses the optimization techniques used in this study. Two of these techniques were novel approaches designed specifically to address issues apparent with biomedical datasets or the application of other techniques. Within the three segmentation projects emerge seven total datasets of images and annotations that were used to produce the results found in the fifth chapter. Finally, the last chapter discusses the conclusions drawn from the results and suggests areas for future work.

This dissertation provided contributions for many field to include systems engineering, data science, machine learning, deep learning, computer vision, artificial intelligence, image segmentation, medical image analysis, hyperparameter optimization, and more. The following is a list of some of the major contributions:

- Demonstrated through the experimentation that Bayesian optimization, specifically with a Gaussian process surrogate function and a upper-confidence bound acquisition function, can achieve high performance in limited-time scenarios.
- Showed the relationship of hyperparameter optimization with allowable runtime. Particular techniques, such as Bayesian optimization, were useful when few trials can be run, but eventually most techniques will converge to similar performance as then number of trials or time grows.
- Introduced the Random Search with Statistical Reduction (RSSR) approach to sequentially reduce the random search space by considering and factoring out trials runs that produce very low performance. This approach also incorporates non-parametric statistical testing. RSSR consistently produces some of the highest validation accuracies in the 8 and 16 hour scenarios.
- Introduced the Gaussian Mixture with Epsilon Greedy (GMEG) approach that was designed to force more exploration into long SMBO runs, because the Bayesian approaches would occasionally test the same settings for many trials. GMEG showed success as one of the most efficient methods for some datasets regarding finding high performance results in a quick manner.
- Displayed strong evidence that certain hyperparameter setting work well with image segmentation models including some settings that aligned with the particular project. There was not "one-size-fits-all" answer for hyperparameter settings. Each dataset must be tuned separately to discover what settings work best.

- Evaluated the Pareto efficiency of the optimizations for finding high validation accuracy in a short amount of runtime. The Gaussian process Bayesian approach was the most efficient of the SMBO techniques, so an obvious recommendation is to build a stopping criterion into this approach. A near-optimal solution is typically found quickly and so the research can save on time and computation by not continuing the optimization for the remainder of the time scenario.
- Improved the detection performance of three different biomedical image segmentation projects that all have critical clinical importance.

# Chapter 2

# Literature Review

This chapter discusses literature and related works regarding the many aspects of this dissertation. The first section discusses CNNs and various details about the concept, architecture, and model training that can impact hyperparameter optimization. The second section is specifically about image segmentation, the different models, and ways to train and evaluate them. The third section discusses various types of hyperparameter optimization and the purpose of each.

## 2.1 Convolutional Neural Networks

Deep feedforward, fully-connected artificial neural networks or multi-layer perceptrons are capable of performing image prediction, but are not efficient at this task [27]. Images with even a moderate size or resolution would require many neural nodes in the first layer to connect with each pixel. This typically results in fully-connected networks being infeasible to train due to the enormous network size or the unreasonable resources needed to be possible. The network may also end up spatially-biased, depending on the data, since learning is still traced back to the original pixel location of objects. CNNs eliminate these training concerns by utilizing convolutional layers to generate important image-level features before reaching fully-connected decision layers [42]. Convolutional layers consist of a set of image filters that are applied to the input image or the current tensor by a mathematical operation known as a convolution [27]. The spatial-bias is removed, because the filters are scanned the through entire image to create a "feature map" that represent where important textures, patterns, or objects, are located on the tensor [27].

The filters can be of any kernel size but are typically $3 \times 3$ pixels. The number of filters in each convolutional layer is also adjustable. The kernel size and the number of filters can both be tunable parameters in a hyperparameter optimization or neural architecture search. Convolutional layers are often featured within a convolutional block and then the CNN architecture can contain several or many blocks, but these attributes are all flexible. The basic convolutional block consists of a convolutional layer and a pooling layer [27]. One

example of a pooling layer is a $2 \times 2$ max-pooling layer that finds that returns the maximum value within every $2 \times 2$ kernel on the tensor to greatly reduce the size of the tensor. Studies have shown that max-pooling can still retain important information, while increasing the efficiency of model training [42]. The number of convolutional layers or blocks in a model are typically optimized through neural architecture search.

While the width and depth of CNNs can be customized for each dataset, the computer vision community typically utilizes existing, pre-trained models by researchers participating in the ImageNet conference [18]. In 2012, the AlexNet CNN architecture utilized a series of convolutional blocks and was trained on a Graphics Processing Unit (GPU) to greatly increase prediction performance on the ImageNet benchmark test set than had been seen before [40]. This sparked a influx of deep learning computer vision research that resulted in development of other popular CNN architectures such as VGG16 [68], ResNet [29], and Inception [75]. Because there are many existing CNN architectures that have shown high-performance, hyperparameter optimization typically avoids little or no architectural aspects of these models when applied. In some cases, a researcher may want to evaluate different ImageNet models such as ResNet18 versus ResNet50 [29]. Although the image input size can be another tunable hyperparameter.

CNNs are trained similarly to multi-layer perceptrons, so they will have similar training and generalization focused hyperparameters. There were some additions to neural networks and the training prcoess that prevent over-fitting. Batch normalization is a technique that standardizes the activations or inputs from one neural layer to the next [33]. This stabilizes the learning process, especially in deeper networks, so larger learning rates can be used. Models with batch normalization can reach optimal parameters in much less epochs and it also provides some regularization benefit. For hyperparameter tuning, the use of batch normalization is typically considered active or not for the entire network. Dropout layers are another generalization technique. Depending on the dropout rate percentage, a random set of activations are dropped or zeroed-out from one layer to the next [72]. This prevents certain nodes from co-adapting together and greatly reduces over-fitting. A single dropout rate for all dropout layers is typically tuned during hyperparameter optimization. It is believed that batch normalization and dropout may not need to be combined in the same network, so the optimization can determine if this is true [44].

Much like multi-layer perceptrons, CNNs utilize an optimizer during training to dictate how the parameters (filter weights and biases) are determined to minimize training and validation loss. The most popular optimizers are stochastic gradient decent [11], RMSProp [31], and Adam [38]. Which optimizer to use can be treated as a categorical hyperparameter. Each optimizer has a learning rate that controls how much the parameters are updated after each batch is processed. The learning rates can be held constant, decreased according to a schedule, or set to decrease exponentially according to the epoch [69]. Learning rate schedulers can be setup for hyperparameter tuning, but the simplest is to tune a single constant learning rate. Learning rate as a continuous hyperparameter can either be modeled as a uniformly-distributed value within a certain range or as a log-normal variable. Finally, the batch size is very important aspect of model training. It coincides with learning rate to

determine how quickly and closely the optimizer reaches a global optimum. For hyperparameter tuning, batch size must be an integer, is usually a power of two, and is constrained by the amount of available processor memory. Therefore, batch size is typically tuned as a categorical variable.

## 2.2 Deep Image Segmentation

Deep image segmentation models are also supervised learning approaches that utilize CNNs. Instead of classifying at the image-level, semantic segmentation models classify at the pixel-level [41]. An image is inputted into the segmentation model and the output is a segmentation mask of the same pixel height and width as the input image. The segmentation mask contains a class label at each pixel that represents what object is present at that location. For segmentation problems in this dissertation, each pixel is a binary classification of whether the specific object of interest is present of not. A positive detection is made if the prediction probability exceeds a selected threshold (typically 0.5 for binary problems). The segmentation masks can then be analyzed or post-processed for certain results depending on the problem. An example: for an image segmentation model that segments a tumor, a biomedical researcher might be interested in tumor size to describe severity, so they would calculate the area or the total number of pixels that classified as positive for tumor in the segmentation mask.

One of the most popular image segmentation models is the U-Net [63]. The U-Net was original designed for biomedical image cell segmentation and can find adequate fits without a substantial amount of labeled data. One major aspect of U-Nets and other similar segmentation models is that they are fully-convolutional networks [49]. This means that the entire model consists of convolutional blocks and there no dense or decision layers at the back-end. For the U-Net, there are four sets of convolutional blocks that contract the image into a feature vector (bottom of the "U") and then more convolutional blocks up-samples the image back to the original dimensions. There are also skip connections that can bypass the contracting path and send minor info to the expansion side of the "U" to ensure that the final mask has adequate resolution [63]. Many attributes of the U-Net architecture can be customized and tuned with hyperparameter optimization or neural architecture search. However, typically only the number of filters in the first layer is adjusted. The number of filters in subsequent layers are based on the initial number of filters.

There have been many modifications and alternate designs of the original U-Net model. Residual blocks have been incorporated into the U-Net architecture to more efficiently train deeper networks on more complex imagery [82, 19, 45]. Although residual models may require more labeled data to take advantage of the residual skip connections than a traditional U-Net. Attention gates were added to the U-Net structure to help the model focus on more critical areas of images and concentrate less on irrelevant areas [57, 1]. Residual blocks and attention gates have been combined in the same U-Net architecture [14, 55]. For certain segmentation problems, it is debatable whether certain areas of the image are more important than others,

so this could diminish the benefit of self-attention. A recurrent residual convolutional layers were added to the U-Net architecture to be able to accumulate important features and improve performance [5]. The U-Net has also been adapted fro 3D image problems [15, 52].

There are other types of non-U-Net based semantic segmentation models. Most of these other models were designed to segment complex imagery into high-resolution, multi-class segmentation masks. The most popular benchmark datasets for these types of tasks are Cityscapes [16] and Microsoft Common Objects in COntext (COCO) [47]. These other semantic segmentation approaches include SegNet [6], DeepLab [13], and Gated-SCNN [76]. While these other semantic models are very powerful, they are more difficult to implement on custom datasets than the U-Net. Further, what works to improve Cityscapes detection may not translate to biomedical engineering. Instance segmentation is another alternative to semantic segmentation and is similar to object detection approaches. Rather than just classifying objects at the pixel-level, instance segmentation models returns bounding-boxes that describe where each individual object is located. One of the most popular instance segmentation approaches is the Mask R-CNN which outputs both segmentation masks and bounding boxes [30]. Detecting individual objects is useful for most biomedical image segmentation models, but instance segmentation is not applied as much as U-net, because they are harder to implement and are believed to require more labeled data.

To train semantic segmentation models, using binary or categorical cross entropy is possible, but is ineffective if there is a large class imbalance [73, 21]. There are many biomedical scenarios where an object, such as a nuclei cell, is much less prevalent than other sections of biopsy tissue samples (background, cytoplasm, stroma, connective tissue). Therefore, region-based loss functions or intersection-over-union approaches are typically preferred for many image segmentation tasks [50]. The most well-known region-based metrics are the Jaccard index [59, 61], Tversky [78, 65, 1], and Dice coefficient [71, 52, 20]. There are also approaches that combine different types of loss functions together such as merging cross entropy and Dice loss [34].

## 2.3 Hyperparameter Optimization Techniques

### Grid and Random Search

Grid search is as exhaustive search method that tests the hyperparameter space in evenly-spaced intervals [9]. The intervals only apply to continuous variables, because categorical variables must be tested for all combinations. The researcher can specify how many trials will be performed and this dictates the interval spacing on the grid. Besides from the computational rigor required to complete a full grid search, the main issue with grid search is low-fidelity sampling for the continuous variables. For example, if there were two continuous hyperparameters and nine trials were to be performed, a grid search intervals would only lead to three different settings tested for each hyperparameter. To solve this problem, random search is preferred over grid search when the continuous variables can be modeled with a

probability distribution [9]. In the same example as above, a random search using a uniform distribution for each continuous variable would produce nine different settings to test for each variable. This ensures that an optimal value can be found for each setting, while maintaining efficiency in the number of trials required.

In design of experiments terms, grid and random search are similar to a space-filling design [53]. Space-filling designs are typically used to test non-linear design spaces when the experiments are relatively quick computer simulations. The difference here is that each trial is a full training run of a deep learning model which could take hours to complete. Before experimentation, a researcher could produce a list of what settings will be tested in a grid or random search and the optimization will not deviate from that list. Therefore, if certain settings appear to produce much lower or higher validation loss performance, the search plan does not change for the entirety of the optimization run. An alternative to grid and random search are Sequential Model-Based Optimization (SMBO) techniques which will determine what settings to test before each trial based on previously collected information from prior experiments within the optimization run [32].

## Bayesian Optimization

Bayesian optimization is a form of SMBO that works well optimizing black-box functions that are expense-to-evaluate like the deep learning model training scenario [25]. Bayesian optimization involves updating a probabilistic posterior distribution $p(y|x)$ after every trial. In this case, the optimization goal (validation loss) is the response $y$ and the hyperparameter settings are the independent variables $x$. This function is known as the surrogate function, because it acts as a substitute or estimation of the true function that maps hyperparameters to optimal loss [25]. The second aspect to Bayesian optimization is the acquisition function. This is the technique used to determine what settings will be tested on the next trial.

Bayesian optimization strategies typically differ in what types of surrogate functions are used. Spearmint was developed by Snoek et al. [70] and uses a Gaussian process as the surrogate function. The Tree-structured Parzen Estimator (TPE) models $p(x|y)$ and $p(y)$ instead of modeling $p(y|x)$ directly [9]. TPE models $p(x|y)$ with two distributions, $l(x)$ and $g(x)$, which are created by splitting the trial results into two groups based on a response cutoff or a fraction of trials. A separate Gaussian Mixture Model (GMM) is fit to each distribution and then the next trial is chosen by finding the settings $x$ that maximizes the ratio of $l(x)/g(x)$ [8]. TPE also requires a warm-up period. It weights later trials more than previous trials and adjusts the fraction of trials in each distribution as the optimization continues. The SMAC technique fits a random forest model as a surrogate function that predicts the validation loss response using the hyperparameter settings from the previously obtained trial results [32]. The possible acquisition functions for Bayesian optimization techniques include probability of improvement, expected improvement, and upper-confidence bound [70].

## Multi-armed Bandit

The multi-armed bandit can be considered within the realm of reinforcement learning and is focused on exploration and exploitation like most other methods. One popular example of a multi-armed bandit is a gambler trying to determine which slot machine has the highest payout [79]. If one slot machine appears to be doing the best, a gambler is inclined to keep exploiting that machine, because maybe it has the best return and thus is more profitable to keep testing it. Although, perhaps it is not the best slot machine and the gambler should keep exploring the other options. The Hyperband approach applies a similar strategy to hyperparameter optimization [43]. Buckets are created that contain random hyperparameters settings and these are screened at a low number of epochs. After a round of this screening process, the settings that did not do well are removed and the process continues, but the number of epochs also increases [43]. Eventually a full model is trained on the settings believed to be the global optimal. Hyperband is not a full-SMBO, so some processes can be run more quickly with parallel processing. The Bayesian Optimization HyperBand (BOHB) approach was designed to combine the beneficial aspects of both strategies and has shown improved performance [23]. The TPE technique replaced the random sampling of configuration within each evaluation round.

## Reinforcement Learning

Reinforcement Learning (RL) is another aspect of machine learning that includes supervised and unsupervised learning. A Markov decision process is the foundation of RL where there are states, actions, transition probabilities, and rewards [74]. In hyperparameter optimization context, the states are hyperparameter settings, the actions change these settings, and the rewards are the validation set accuracy. There are a few methods that apply RL to hyperparameter optimization with Deep Q-Learning [80] [36]. A Long Short-Term Memory (LSTM) recurrent neural network is used as a state representation to store the settings and performance results as the trials were tested. There are also methods for utilizing the Proximal Policy Optimization (PPO) technique to perform parts of hyperparameter optimization or neural architecture search[66].

# Chapter 3

# Segmentation Datasets

In this chapter, the three projects used to evaluate the hyperparameter optimization techniques are presented. The clinical or medical research impact of each image segmentation project are also explained.

## 3.1 Eosinophilic Esophagitis Biopsy Images

Eosinophilic Esophagitis (EoE) is an inflammatory disease of the esophagus and is increasing in prevalence. The inflammation, caused by eosinophil (a type of white blood cell) response is believed to be an allergic reaction to dietary factors in patients [12]. The symptoms can include swallowing issues, food impaction, regurgitation, and decreased appetite [64]. To diagnosis EoE, patients undergo an endoscopy where tissue samples were retrieved from locations within the patient's esophagus. These biopsy tissue samples were colorized with Hematoxylin and Eosin (H&E) staining in order to visualize the cellular features. Histopathologists can review the tissue samples under a microscope or with digital imagery. For the latter, the stained biopsy tissue samples were digitized and stored as very large-resolution Whole-Slide Images (WSI). The gold-standard diagnostic criteria is to find at least 15 eosinophils within at least one 400× magnification high-power field of esophageal tissue [26].

Due to the large size of WSI, it can be quite difficult for pathologists to accurately describe EoE severity and extent. It would require significant time and effort to count each eosinophil contained in the tissue samples. An automated eosinophil detection model could quickly and accurately summarize EoE characteristics of each patient so that they can receive personalized disease management plans. With aspects of EoE medical research still in its infancy, this tool could also assist with computer-aided diagnosis, discovery of eosinophil trends with other cellular features, and finding potential linkages of eosinophil presence with clinical and treatment phenotypes. Adorno et al. [4] presented the first technique to apply deep learning computer vision to quantify eosinophils with EoE patients.

WSI cannot be feasibly processed through deep learning models due to memory constraints [24]. Annotating entire WSI was unreasonable for annotators to handle and could

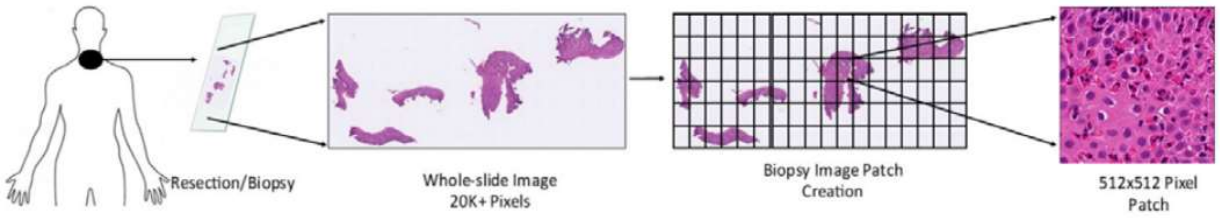



Figure 3.1: This figure shows the process of converting a biopsy sample into digitized whole-slide image and then split into many 512 × 512 pixel patches.

reduce patient-level diversity if not enough were labeled. Therefore, the WSI were patched into smaller images of 512 × 512 pixels for annotation and for model input [24]. Figure 3.1 shows a visual depiction of the WSI digitization and patching process. The results of a segmentation model at the patch size can be scaled to the high-power field size using a sliding window approach. After a few rounds of labeling there were 483 patches annotated at the pixel-level indicating whether an eosinophil was located there. These patches were derived from WSIs of 29 different patients. Figure 3.2 shows nine examples of eosinophil annotations on esophageal biopsy image patches of patients diagnosed with EoE.

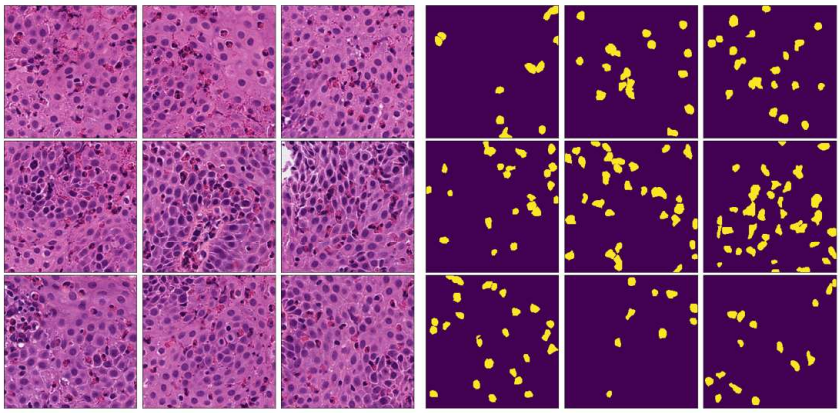


Figure 3.2: This figure shows examples of patch-level annotations of eosinophils on EoE biopsy patches. The eosinophils were outlined by trained experts on specialized software and then converted into ground-truth segmentation masks.

Javaid et al. [35] trained a U-Net model on the EoE dataset using a 20% training/validation split. The hyperparameters were user-established at:

- Batch normalization = True
- Number of filters = 32

- Learning rate = $2 \times 10^{-4}$
- Dropout Rate = 0.25
- Batch size = 4

These hyperparameter settings led to a validation accuracy of **0.671** for the Dice coefficient. This model was trained with similar settings as discussed later in the Experimental Setup chapter. The segmentation model was applied to generate eosinophil counts at the high-power field level. The eosinophil counts were aggregated to create patient-level statistics to include: maximum eosinophil count within an HPF, percentage of high-power fields with greater than 15 eosinophils, and average eosinophil size. These stats and others were used to describe EoE severity and extent in patients at their initial biopsy. Ultimately, the end-goal is to determine if there is any linkages between initial biopsy eosinophil stats and patient's clinical or treatment phenotype. If linkages are discovered, then it could be possible to design a disease management plan to reach remission sooner and improves patient outcomes. Figure 3.3 shows examples of comparing the eosinophil stats with patient phenotypes [4].

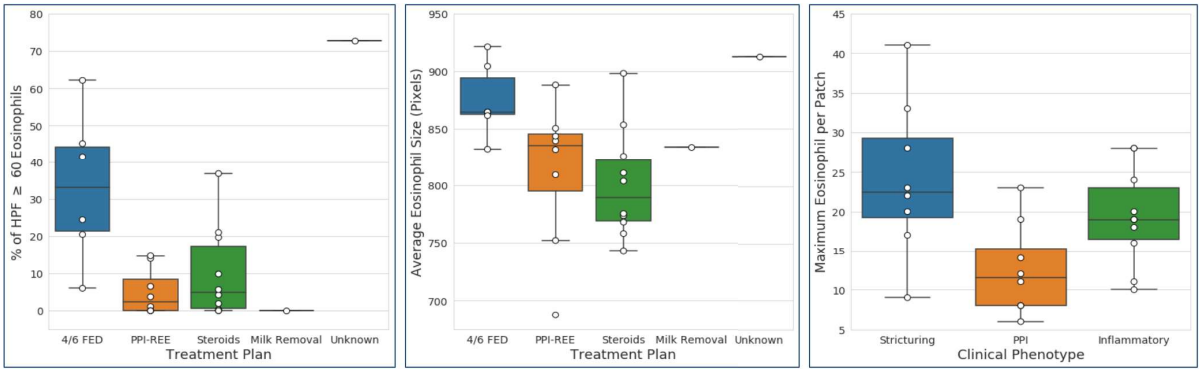



Figure 3.3: This figure shows three different examples of examining linkages between patient-level eosinophil count statistics at initial biopsy with a patients' phenotypes. For instance, the left-most chart shows the the 4 to 6 Food Elimination Diet (Diet

treatment appeared to have a linkage with patients who have a large percentage of high-power fields containing 15 or more eosinophils.

The performance of segmentation model directly affects the accuracy of counting eosinophils on biopsy images. Improving the current performance through hyperparameter optimization can reduce false positives and negatives in diagnosing patients and result in better characterization of severity and extent. It also ensures that the data used to assess phenotype linkages are reliable.

## 3.2 Cardiovascular Immunofluorescent 3D Images

Heart attack and stroke are leading causes of death in the world. [7] Each are caused by rupture of atherosclerotic plaques that release debris that can block vessels in the heart or brain. Studies show that the composition of the atherosclerotic plaque is more of indicator of heart attack or stroke than the size of the lesion. [46, 58]. Atherosclerotic composition can be described by quantifying the number of certain cell types within the lesion. This cell quantification can be performed with image segmentation of 3D atherosclerotic plaque immunofluorescent images. Adorno et al. [2] developed an approach to accommodate multiple annotation strategies to accuracy quantify the number of cells for three different cell types.

Cardiovascular immunofluorescent images are quite different from the biopsy images discussed in the EoE dataset. These images are $2,048 \times 2,048$ pixels in size for height and width, but also contain five different color channels and a depth ($Z$-axis) of varying size. The $Z$ can cause difficulty when annotating cells and when processed through image segmentation models, so a maximum intensity projection was applied to eliminate the $Z$ dimension [54]. The five color channels correspond to four different antibody stains and confocal scanning microscopy. Three of the antibody stain color channels, 4',6-diamidino-2-phenylindole (DAPI), Actin alpha 2 (ACTA2), and Galectin-3 (LGALS3), are used to detect nuclei, Smooth Muscle Cells (SMC), and macrophages, respectively, in atherosclerotic plaque lesions. The fourth stain antibody, the Yellow Fluorescent Protein (YFP) lineage marker, did not have a large enough sample size to develop a cell detection model. Each stain antibody coincides with a particular fluorescent color: nuclei = blue, SMC = red, macrophages = magenta, and YFP = green.

One major challenge for this problem was developing an annotation strategy to efficiently label a substantial number of cells per image. Normally, annotations are created that precisely outline the entire cell structure. Due to limitations, only the nuclei cells were fully annotated in this manner, but the other cells were annotated with basic shapes that similarly described the location of the cells. Figure 3.4 shows an example of the two different annotation strategies for a single crop of a immunofluorescent image.

Adorno et al. [2] evaluated each of the annotations strategies to determine which was most accurate at cell quantification. The ground-truth total cell counts were obtained directly from the original annotations and compared to the total counts predicted by U-Net models trained on each cell type and annotation strategy. The results showed that different annotation strategies performed best for each cell type. The nuclei cells achieved accurate cell quantification with a "dots" technique for the segmentation masks that represents each cell at a small circle centered at the annotation centroid. The SMC cells were best quantified by using the ellipses shapes annotations for the segmentation masks. Lastly, the macrophages were best quantified by linking the two annotating strategies by replacing the circle annotations with nuclei polygons where they intersected. Every cell type must include a nuclei cell, so this combination utilizes the outlined nuclei cell annotations to bolster prediction of macrophages. Figure 3.5 shows an example of one immunofluorescent image patch and the optimal annotation strategy for each cell type.

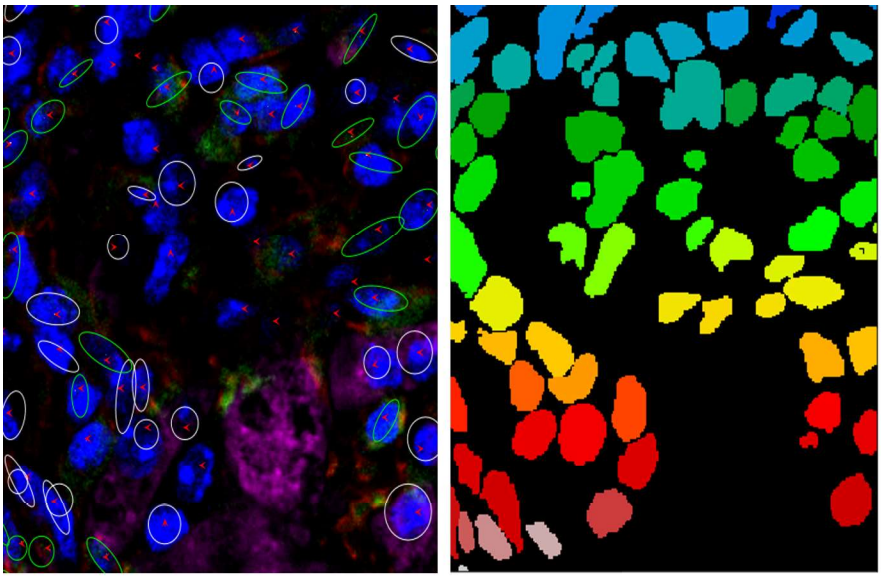


Figure 3.4: This figure shows an example at a single $Z$-level of the two different annotation strategies used to represent cell types within the cardiovascular immunofluorescent images. On the left, the nuclei cells are marked with red arrows, the SMCs are annotated with ellipses, and the macrophages are annotated with circles. On the right, only nuclei cells are outlined in the traditional annotation format. These annotations correspond with the blue fluorescent antibody stain and the red arrow markings on the left image.

The original images and masks were patched to $512 \times 512$ pixel images for input into the segmentation model. Each image set was allocated into training and validation sets with roughly a 20% split. Certain images had to be selected for the validation set to avoid data leakages issues, because in some cases there was an overlap of atherosclerotic plaque lesions between multiple images from the same patient. Table 3.1 shows the image counts for training and validation sets across the three cell types.

Table 3.1: Training and validation set patch counts for each cell type.

| **Cell Type** | **Training** | **Validation** |
|---|---|---|
| DAPI (Nuclei) | 851 | 156 |
| ACTA2 (SMC) | 702 | 145 |
| LGALS3 (Macrophage) | 710 | 147 |

U-Net models were trained using these training and validation sets and then evaluated on the validation set Dice coefficient. The segmentation masks from the best annotation strategies were used for ground-truth. The validation coefficient results were **0.683**, **0.451**,

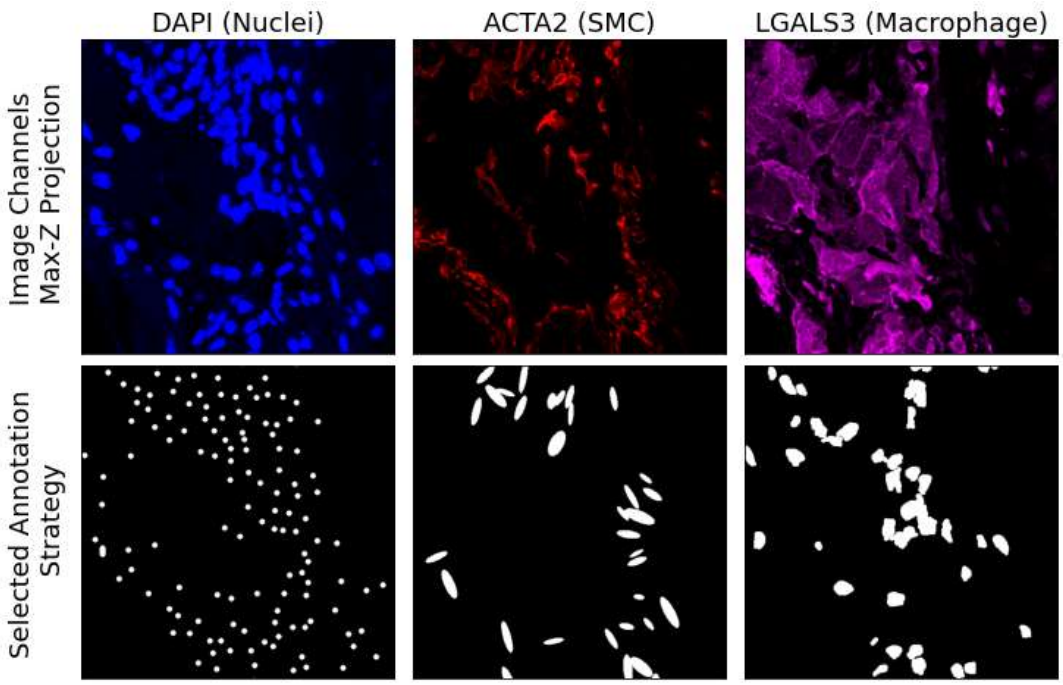



Figure 3.5: This figure shows an example of one patch of a cardiovascular immunofluorescent image. The top row shows each of cell type channels with maximum intensity projection of the $Z$-axis applied. The colors refer to each antibody stain. The bottom row shows segmentation masks for annotation strategy that most accurately counted the total number of cells.

and **0.710** for nuclei, SMC, and macrophage detection, respectively. The following user-established hyperparameters were used to train each of the three models for the three cell types:

- Batch normalization = True
- Number of filters = 32
- Learning rate = $2 \times 10^{-4}$
- Dropout Rate = 0.25
- Batch size = 4

With hyperparameter optimization applied to optimize the performance of these three models, the cell quantification of each type will be more accurate. This will lead to a better estimation of atherosclerotic plaque composition and hopefully result in better understanding and prevention of heart attacks and strokes.

## 3.3 Vital Signs Graph from Surgical Flowsheet

Analyzing perioperative data is critical in understanding why certain patients died from surgery or had long post-surgery hospital stays. In low-to-middle income countries, this analysis is very difficult, because perioperative data is often not collected digitally during surgery. Perioperative data is hand-written onto paper surgical flowsheets, but there was not an efficient process to digitize this data. To solve this problem, Rho et al. [62] designed a scanning apparatus that can be used to effectively and consistently convert the paper flowsheets into high-quality digital images. As digital images, image processing or deep learning computer vision approaches can be applied to extract the important information. The vital signs graph section of the flowsheet, in particular, could be digitized with an image segmentation approach. Figure 3.6 shows an example of the surgical flowsheet with the vital signs graph outlined in red.

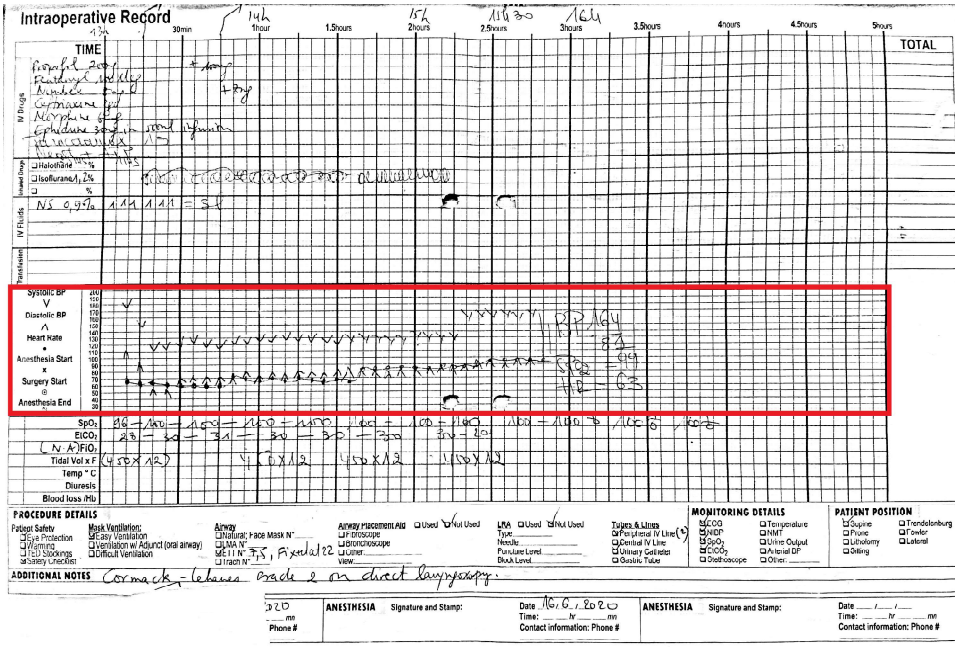
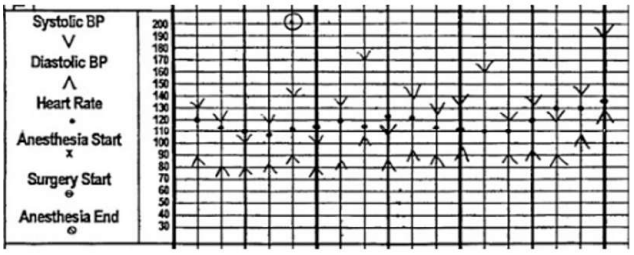



Figure 3.6: This figure shows an example of a paper flowsheet that includes handwritten perioperative data. The flowsheet includes sections such as the medications administered, surgical info checkboxes, and the graph to track vital signs. The vital signs graph is outlined in red.

The vital signs graph contains recorded Blood Pressure (BP) and heart rate readings at 5-minute intervals. Blood pressure in particular is believed to be closely linked with post-operative mortality. These vital signs are captured using three different symbols on the graph at each time interval. The heart rate is represented as small circles, the systolic

BP is represented as downward arrows, and diastolic BP is represented as upward arrows. The goal of digitization is to detect the presence and location of each symbol on the graph and then convert that into the true numerical values at the appropriate time point. Figure 3.7 shows an example of a vital signs graph and the resultant digitized numerical values in spreadsheet form.




| Time (Minutes) | Heart Rate | Systolic BP | Diastolic BP |
|---|---|---|---|
| 5 | 120 | 130 | 90 |
| 10 | 115 | 120 | 80 |
| 15 | 110 | 100 | 80 |
| 20 | 108 | 118 | 82 |
| 25 | 112 | 140 | 90 |
| 30 | 115 | 100 | 80 |
| 35 | 120 | 135 | 85 |
| 40 | 115 | 170 | 105 |
| 45 | 123 | 110 | 85 |
| 50 | 121 | 140 | 95 |
| 55 | 112 | 125 | 90 |
| 60 | 110 | 135 | 98 |
| 65 | 110 | 160 | |
| 70 | 110 | 120 | 90 |
| 75 | 120 | 135 | 92 |
| 80 | 130 | 120 | 90 |
| 85 | 130 | 142 | 108 |

Figure 3.7: This figure shows an example of a vital signs graph that has been converted into digitized numerical values. On the left, is a section of the vital signs graph from a perioperative flowsheet with the symbols manually recorded by hand. On the right, is an example of the final end result in converting of all the symbols on the graph into tabular data for further analysis.

Since the location of the detected symbols on the graph relate to the numerical value of each, image segmentation can be applied to produce a segmentation mask with all detected symbols. The segmentation mask can be post-processed to obtain time and value of each symbol. Adorno et al. [3] developed a process to annotate symbols, train U-Net segmentation models, and post-process the symbols. There were 29 vital sign graphs annotated of size $164 \times 990$ pixels. The heart rates were annotated with a single coordinate, while the BPs were annotated by outlining the arrows with a rectangle. The heart rates were converted to small circles with a radius of three pixels and centered at the annotated coordinate in the segmentation masks. Some of the these symbols could overlap or be near each other, so the segmentation masks were kept separate and models were trained to predict each symbol. Figure 3.8 shows an example of the raw annotations on a vital signs graph image.

The 29 total images were divided into training and validation sets with 23 and 6 images, respectively, which is roughly a 20% split. The images are black-and-white, so they are converted into a one channel gray-scale image before model training. The original input size did not not work well as a U-Net input, so the original image is placed in the top-right corner of a $256 \times 1024$ pixel image and then zero-padded elsewhere. This ensure that the image can process through a U-Net and preserves the original resolution. This may seem like a small number of images for CNN training, but these images and objects are not complex

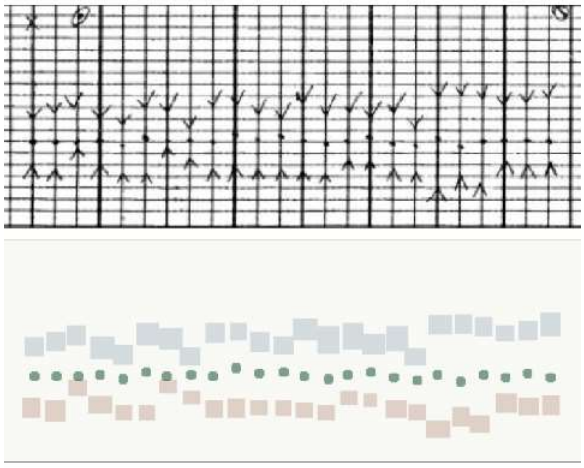


Figure 3.8: This figure shows an example of the annotations performed on vital signs graphs. The top image is an example vital signs graph. The bottom image is annotations for each of the three symbols. The systolic BPs are blue squares, the heart rates are green dots, and the diastolic BPs are red squares. Each symbol is separated into different segmentation masks to train separate models.

and all contain images contain a similar background setting which facilitates training. Three separate U-Net models were trained for symbol and resulted in the validation Dice coefficients **0.835**, **0.889**, and **0.879** for heart rate, systolic BP, and diastolic BP, respectively. The following user-established hyperparameters were used to train each of the three models for the three cell types:

- Batch normalization = True
- Number of filters = 20
- Learning rate = $2 \times 10^{-3}$
- Dropout Rate = 0.25
- Batch size = 2

Improving image segmentation performance through hyperparameter optimization could both reduce false positive and negative symbol detections, as well as, decrease the error between ground-truth and predicted digitized numerical values on true positives. The post-processing steps rely on a segmentation mask that is accurate in order to search the space and obtain numerical values for each detected symbol. Improved digitization in these areas will ensure that datasets used to analyze linkages between perioperative vital signs and post-operative mortality and hospital stay length are reliable and worthy of finding significant clinical conclusions.

# Chapter 4

# Methodology

This chapter explains the details of how the hyperparameter optimization evaluated experiments were designed and executed. The experimental setup and scenarios are discussed first. The image segmentation model and architecture was presented. The tested hyperparameters are listed and explained. Finally, the evaluated hyperparameter optimization techniques are listed to include two novel approaches introduced in this dissertation.

## 4.1 Experimental Setup

The following sections explain all of the details of how these experiments were coded and performed.

### Scenarios

To evaluate hyperparameter optimizations, timed scenarios are used. Using time limits was better to fairly compare methods than the number of trials performed, because then number of trials can vary based on the design of each technique. With a high enough time-limit, most optimization techniques should converge to similar results. Therefore, scenarios are performed at multiple time limits, so it can found which techniques do best under strict time constraints. We can also see how runtime affects optimization and about how long it will take to achieve near optimal performance. The time scenarios executed in this study are: 2, 4, 8, 16, and 24 hours.

### U-Net Architecture

There are many different semantic and instance segmentation architectures. Model selection of the architecture itself usually comes before hyperparameter optimization. Models can be compared based off of validation results obtained via hyperparameter optimization, but this just increases the amount of computation required and was not a good way of comparing

optimization techniques. Therefore, in this study, a single architecture was utilized for all experiments. The base U-Net model trains quickly, was easy to implement, and does not require a substantial amount of data to find a fit [63]. Table 4.1 shows the exact U-Net architecture used in this research. The input image size (height × width × channels) changes based on what dataset was used, so this was kept as a variable in this depiction of the architecture. Also, the initial number of filters (F) can be adjusted as a hyperparameter, so this remains flexible in the architecture.

Table 4.1: This table shows the U-Net architecture used throughout all the experiments. The left side of the table was the contracting side of the U-Net (processes downwards). The right side was the expansion side (processes upwards in the table). The variables in the tensor output size refer to: H = image height, W = image width, C = number of input channel, and F = number of initial filters.

| **Contracting Side** | | | | **Expansion Side** | | |
|---|---|---|---|---|---|---|
| Block | Layer Type | Output Tensor Size | Skip Connections | Layer Type | Output Tensor Size | Block |
| Biomedical Image | Input | H x W x C | | Conv | H x W | Output Mask |
| | | | | Batch Norm | H x W x F | |
| | Conv | | | Conv | | |
| | Batch Norm | | | Batch Norm | | |
| Contracting Block 1 | Conv | H x W x F | | Conv | H x W x 2F | Expansion Block 4 |
| | Batch Norm | | Concatenate | Dropout | | |
| | Max Pooling | | | Conv Transpose | H x W x F | |
| | Dropout | H/2 x W/2 x F | | Batch Norm | H/2 x W/2 x 2F | |
| | Conv | | | Conv | | |
| | Batch Norm | | | Batch Norm | | |
| Contracting Block 2 | Conv | H/2 x W/2 x 2F | | Conv | H/2 x W/2 x 4F | Expansion Block 3 |
| | Batch Norm | | Concatenate | Dropout | | |
| | Max Pooling | | | Conv Transpose | H/2 x W/2 x 2F | |
| | Dropout | H/4 x W/4 x 2F | | Batch Norm | H/4 x W/4 x 4F | |
| | Conv | | | Conv | | |
| | Batch Norm | | | Batch Norm | | |
| Contracting Block 3 | Conv | H/4 x W/4 x 4F | | Conv | H/4 x W/4 x 8F | Expansion Block 2 |
| | Batch Norm | | Concatenate | Dropout | | |
| | Max Pooling | | | Conv Transpose | H/4 x W/4 x 4F | |
| | Dropout | H/8 x W/8 x 4F | | Batch Norm | | |
| | Conv | | | Conv | | |
| | Batch Norm | | | Batch Norm | H/8 x W/8 x 8F | |
| Contracting Block 4 | Conv | H/8 x W/8 x 8F | | Conv | | Expansion Block 1 |
| | Batch Norm | | Concatenate | Dropout | H/8 x W/8 x 16F | |
| | Max Pooling | | | Conv Transpose | H/8 x W/8 x 8F | |
| | Dropout | H/16 x W/16 x 8F | | Batch Norm | | |
| | Conv | | | Conv | H/16 x W/16 x 16F | |
| Bottom of U | Batch Norm | H/16 x W/16 x 16F | | | | Bottom of U |

## Experiment Details

All experiments were performed on Rivanna, the University of Virginia's high-performance computing cluster. All code was written in Python 3.7 and used the deep learning framework TensorFlow 2.1 with Keras 2.2.4. Rivanna contained several different options for GPU, so allowing it to select whatever was available could lead to unfair comparisons across opti-

mization techniques. Therefore, each dataset used a particular GPU for every experiment. The following was the GPU list across the datasets:

- EoE eosinophils: NVIDIA GeForce RTX 2080 Ti
- Cardiovascular Nuclei: NVIDIA Tesla P100
- Cardiovascular SMC: NVIDIA Tesla P100
- Cardiovascular Macrophage: NVIDIA Tesla P100
- Graph reading heart rates: NVIDIA Tesla V100
- Graph reading systolic BP: NVIDIA Tesla V100
- Graph reading diastolic BP: NVIDIA Tesla V100

Using different GPUs across the datasets also allowed for more experiments to be running simultaneously. Multiple GPUs can be utilized to run jobs in parallel. Parallel computing does not add much benefit to SMBO techniques since trials must be run one at a time, but random search can be performed on multiple nodes to execute more trials within the time limit. Only one GPU was used for every experiment in this study. The effect of multiple GPUs can still be estimated with the generated timed-scenario results. For example, if a 2-hour random search was performed on 2 GPUs, then one could assume that the results are similar to a 4-hour random search performed on 1 GPU. Each model was trained for up to 100 epochs. An early stopping criteria was implemented that ended training when validation loss did not improve after ten consecutive epochs. A negative Dice coefficient was used to measure loss for training and validation since loss was minimized by default in Keras. The Adam optimizer was used to optimize the model parameters during training. No data augmentation methods were applied to the images and masks in these experiments to ensure consistency across experiments.

## 4.2 Tested Hyperparameters

The following list shows the hyperparameters that were tuned in the experiments:

- Batch normalization - categorical {True, False}
- Batch size - categorical {2, 4, 8}
- Dropout Rate - continuous as Uniform(0, 0.5)
- Learning rate - categorical $\{2 \times 10^{-1}, 2 \times 10^{-2}, 2 \times 10^{-3}, 2 \times 10^{-4}, 2 \times 10^{-5}\}$
- Number of filters - categorical {16, 20, 24, 28, 32}

There were batch normalization layers throughout the architecture, so "True" means that all will stay active, while "False" means that batch normalization was completely removed. The dropout rate was a single value that applies to all dropout layers. Although the very first dropout rate in the contracting path was divided by two. Dropout rates above 0.5 are uncommon. If the optimal dropout rate was found to be zero, then this would be equivalent to not having any dropout layers. The number of filters must be an integer and divisible by 2, so it was easiest to make this a categorical variable. A max of 32 was used due to memory constraints. The batch size must also be an integer and is commonly a power of two, so it was also treated as categorical. Batch sizes above 8 were likely to lead to Out-Of-Memory (OOM) errors. The learning rate could be modeled as a continuous variable with a log-normal distribution. However, not all optimization techniques have the ability to represent learning rate this way, so it was modeled as categorical with five of the most common rates used.

It was important that any combination of these hyperparameter settings results in a trial that does not generate an OOM error. For example, it may be possible to run a batch size of 16 if the number of filters is 16, but 32 initial filters would result in an OOM error. To create an orthogonal hyperparameter search space, the settings are restricted to where all possibilities can be successfully trialed. Hyperparameter optimizations can be run with many more tunable hyperparameters, but these five are the most critical to training a U-Net and sufficiently compare techniques in a reasonable time frame.

## 4.3 Existing Optimizations

### Random Search (RS)

The random search implementation from the *Keras Tuner* Python package was utilized [56]. Typically a certain number of trials are established before executing random search, but it would be difficult to compare all optimizations fairly using trials since some techniques run trials with a low maximum number of epochs. The number of random trial is set to a large number so that the search continues for the duration of the timed-scenario. Each trial run is random and independent from the next, so completing as many trials as possible within a time-limit should be equivalent to running a random search with that many trials specified at the start.

### Gaussian Process - Upper Confidence Bound (GP-UCB)

The Bayesian optimization implementation from the *Keras Tuner* Python package was utilized [56]. This implementation uses a Gaussian Process (GP) as the surrogate function and Upper-Confidence Bound (UCB) as the acquisition function. The maximum number of trials is set to a a large number so that the optimization will run for the entirety of the timed-scenario, if needed. It is possible that the optimization converges to an optimal solu-

tion and stops testing before the time limit expires, but this scenario never occurred in this study. This SMBO requires a warm-up period to establish the Gaussian process and start using the UCB acquisition function. The provided rule-of-thumb for warm-up trials was to use three times the number of the hyperparameters settings which resulted in 15 trials used in this study. In shorter time-scenarios, it is possible that only warm-up runs are executed and the actual Bayesian optimization never begins. It appears the warm-up trial runs in *Keras Tuner* are similar to a grid search and start off testing the boundaries of the space. This techniques also contains two of its own hyperparameters: $\alpha$ and *beta*. $\alpha$ represents the expected amount of noise in the observed performances and $\beta$ is a balancing factor between exploration and exploitation. These settings are left to their default values of $1 \times 10{-}4$ and 2.6 for $\alpha$ and $\beta$, respectively.

### Hyperband (HB)

The Hyperband implementation from the *Keras Tuner* Python package was utilized [56]. The number of maximum trials was also established as a very large number for Hyperband, but this number needs to be even greater than the other approaches, because Hyperband tends to run many trials at a low number of maximum epochs. The *Keras Tuner* implementation does not have a minimum number of epochs hyperparameter, so it is possible to do very short trial runs that do not gain much of a fit on the training or validation data. If that setting was available, it would still be difficult to determine a number that would balance exploration and exploitation. The reduction factor was left to the default value of 3. The number of Hyperband iterations was also set to a high enough value that the algorithm would continue to run for the entire timed-scenario.

### Tree-structured Parzen Estimator (TPE)

The TPE implementation from the *HyperOpt* Python package was utilized [8]. The maximum number of trials was set to a large number to ensure that TPE ran for the entirety of the time-scenario. All other TPE hyperparameters were left to the default values established within HyperOpt.

## 4.4 Random Search with Statistical Reduction (RSSR)

All SMBO techniques intend to reduce the search space and efficiently balance exploration and exploitation. However, existing approaches can have difficulties during execution of certain datasets and problems. For example, the Bayesian optimization technique acquisition function may not reliable due to many trials with low validation accuracy. Certain settings can lead to slow training that stops prematurely due to the established early-stopping criteria. When this occurs, the resulting validation losses of all previous trials appear to be a bimodal

distributed. This cause certain settings to have unusually high uncertainty and directly affect an SMBO's acquisition function. The method Random Search with Statistical Reduction (RSSR) was developed to reduce the search space and evaluate settings in more beneficial area, while providing more flexibility to handle highly non-linear and non-parametric results.

The first step in RSSR is to run a random search optimization to produce enough results to be able to statistically analyze differences in all hyperparameter setting changes. The amount of runtime needed is dependent on each dataset, how many hyperparameters are being optimized, and how many bins the categorical variables contain. After the random search is complete, the next step is to split the trial results into two separate categories based on the final validation accuracy: good runs or bad runs. The goal is to analyze performance of settings using all trials and only-good trials. While it may seem worth it to consider only-good trials, it is useful to assess the trends of bad trials to determine if certain settings consistently do not train well. To assign each trial as good or bad run, a validation Dice threshold is determined by finding the intersection of two Gaussian distributions that were fit to the validation results using a Gaussian Mixture Model (GMM) [77]. The GMM is fit using the *scikit-learn* package in Python [60]. The GMM portion does assume that the good and bad trials are each normally-distributed. This may not always hold true, but it is still effective at modeling the separation each and to provide a probabilistic method for finding the threshold. The threshold is determined by setting the two fitted Gaussian distributions equal to each other and then solving for the validation Dice variable.

The rest of this explanation of the RSSR method will use results obtained from the hyperparameter optimization of systolic BP. A 2-hour random search was executed a produced 147 trials worth of setting combinations and validation Dice results. The graph reading data produces many trials in a short period of time due to the smaller dataset size. The EoE and cardiovascular data need at least a 4-hour scenario to produce enough runs for statistical comparison. With the 2-hour SBP results, the first step is determining the validation Dice threshold that will split good and bad trials. Figure 4.1 shows the validation Dice results in a few different formats. The boxplot and histogram confirm that validation loss not bell-shaped and is bimodal distributed. A GMM with $k = 2$ component was fit on all of the validation Dice results to produce a $\mu$ and $\sigma$ for each distribution. The chart on the right of Figure 4.1 shows the Gaussian Probability Density Function (PDF) for each Gaussian. The green dot, the solved intersection of the two PDFs, is located at 0.344.

The next step is to assess each hyperparameter to determine if certain settings can be eliminated for another random search optimization. If the next random search exploits the most beneficial settings, then it may have a higher likelihood of finding the global optimal than continuing searching the full hyperparameter. The first round of random search acts as the exploration and factor-screening phase and then the next round is targeting the settings of most promise. The following sections explain how RSSR is applied to binary, multi-category, and continuous hyperparameters. To statistically evaluate the hyperparameter settings, a Mann-Whitney U test is utilized . The Mann-Whitney U test is a non-parametric equivalent to the parametric t-test of independently sampled groups [51]. The limitation here is that the hyperparameters are assessed independently, because it would be difficult to

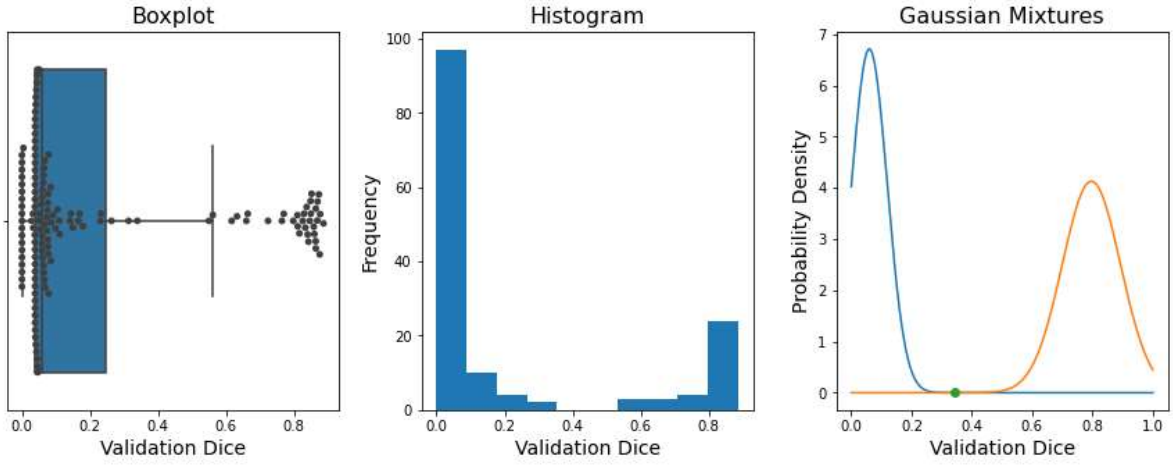



Figure 4.1: This figure shows the validation Dice results for systolic blood pressure symbol detection. A total of 147 trial results were obtained in a 2-hour random search optimization. The left chart shows a boxplot of the validation Dice. The middle chart shows a histogram of the same response. It is clear from these two charts the data is bimodal. The chart on the right shows the two Gaussian distributions that were fit to this data and the intersection between them. This intersection splits good trial runs from bad trials.

execute statistical reduction based on hyperparameter interaction results.

## Binary Hyperparameter

Batch normalization (True or False) is a binary hyperparameter optimized in this study and is used here to show an example of applying RSSR. Figure 4.2 shows the boxplots of validation Dice for both batch normalization as "True" and "False" for all trials and only good trials. After the validation Dice cutoff is applied, there are only 34 results in the good trial category. This indicates that there were many hyperparameter settings that lead to slow training and early stoppage. From both charts, it is evident that using batch normalization tends to produce higher validation Dice results. Batch normalization appears to result in less bad trials and good trials are also higher performing. The maximum values for validation Dice also come during trials where batch normalization is active. This is all confirmed in the Mann-Whitney U tests results which produced p-values of $7.4 \times 10^{-6}$ and 0.010 for all trials and only good trials, respectively. To automate this process, the user can establish that both test results must reject the null hypothesis to remove one of the settings or just one test results can be enough. Typically, an $\alpha = 0.05$ or 95% confidence is sufficient to make this test determination. For this example, batch normalization will be set to "True" for the entire second random search based on these findings.

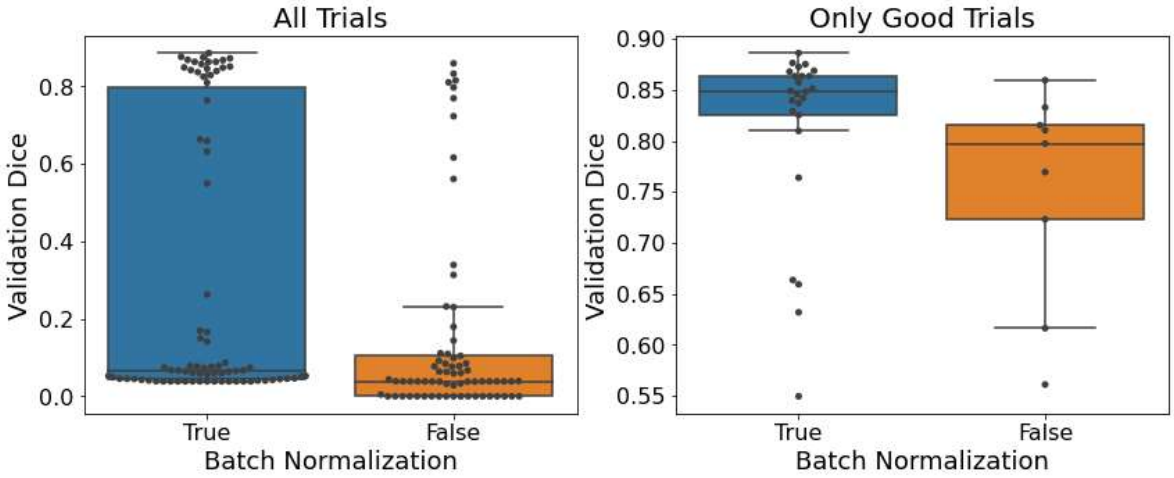



Figure 4.2: This figure shows the validation Dice results for systolic blood pressure symbol detection by batch normalization. The left chart contains all 147 trials, while the right chart contains the 34 good trials. It appears that batch normalization contributes to higher validation Dice and future random search trials should always contain batch normalization.

## Multi-category Hyperparameter

Learning rate for the Adam optimizer is one of the multi-category hyperparameters optimized in this study and is here to show an example of applying RSSR. Figure 4.3 shows two sets of boxplots for validation Dice split by learning rate. The chart on the left shows all trials, while the chart on right shows only-good trials. It is apparent that learning rates of $2 \times 10^{-3}$ and $2 \times 10^{-4}$ stand out when looking at all trials, while $2 \times 10^{-2}$ and $2 \times 10^{-3}$ appear to perform best on only-good trials. There is always a possibility of differences between all trial versus only good trial, so it is important to include both sets when considering to reduce the search space.

The process to evaluating these settings is similar as mentioned above for binary comparisons, but we must run a Mann-Whitney U test pairwise for all setting combinations. These statistical tests are performed separately for all trials and for only-good trials. An $\alpha = 0.5$ is used to reject or fail to reject the null hypothesis that each distribution is equal. If the null hypothesis is rejected, the test combination receives a 1 if $\mu_i > \mu_j$ and a $-1$ if $\mu_i < \mu_j$ where $i$ and $j$ are each categorical setting. Table 4.2 shows these results for the statistical comparisons of all trials. If column has many $-1$ values, then it means that this setting is dominated by the other settings and should be considered for removal.

The same statistical comparison strategy is performed on the good trials, as well. Table 4.3 shows the results for the only the good trials.

After completing these tables for all trials and only good trials, the results can be combined to have an overall assessment of what variables should remain or be removed. Table

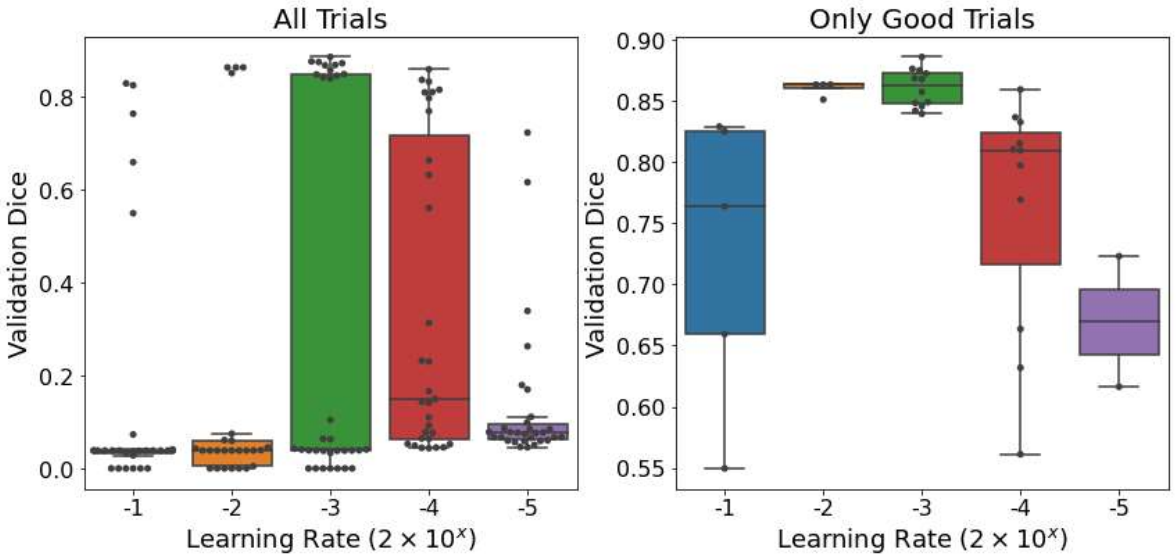



Figure 4.3: This figure shows the validation Dice results for systolic blood pressure symbol detection by Adam optimizer learning rate. The left chart contains all 147 trials, while the right chart contains the 34 good trials. It appears that learning rates of $2 \times 10^{-2}$, $2 \times 10^{-3}$, and $2 \times 10^{-4}$ contributes to higher validation Dice and future random search trials should consider only testing these settings

Table 4.2: This table shows the statistical comparison results of learning rate for the five possible settings on all trials. An entry receives a -1 or 1 if the distributions are statistically different according to a Mann-Whitney U test. A -1 or 1 is assigned based on if the mean of the row entry is less than or greater than the column entry. A learning rate of $2 \times 10^{-5}$ was significantly lower in validation Dice performance than three other settings.

| | $2 \times 10^{-1}$ | $2 \times 10^{-2}$ | $2 \times 10^{-3}$ | $2 \times 10^{-4}$ | $2 \times 10^{-5}$ |
|---|---|---|---|---|---|
| $2 \times 10^{-1}$ | | 0 | 1 | 1 | -1 |
| $2 \times 10^{-2}$ | 0 | | 0 | 1 | -1 |
| $2 \times 10^{-3}$ | -1 | 0 | | 0 | 0 |
| $2 \times 10^{-4}$ | -1 | -1 | 0 | | -1 |
| $2 \times 10^{-5}$ | 1 | 1 | 0 | 1 | |

Table 4.3: This table shows the statistical comparison results of learning rate for the five possible settings on only good trials. A learning rate of $2 \times 10^{-3}$ was significantly higher in validation Dice performance than three other settings.

| | $2 \times 10^{-1}$ | $2 \times 10^{-2}$ | $2 \times 10^{-3}$ | $2 \times 10^{-4}$ | $2 \times 10^{-5}$ |
|---|---|---|---|---|---|
| $2 \times 10^{-1}$ | | 1 | 1 | 0 | 0 |
| $2 \times 10^{-2}$ | -1 | | 0 | -1 | 0 |
| $2 \times 10^{-3}$ | -1 | 0 | | -1 | -1 |
| $2 \times 10^{-4}$ | 0 | 1 | 1 | | 0 |
| $2 \times 10^{-5}$ | 0 | 0 | 1 | 0 | |

4.4 shows the sum of Tables 4.2 and 4.3. Again, negative values across a column indicate that the setting is not useful and positive values indicate that the setting is useful.

Table 4.4: This table combines the all trials and only good trial results for learning rate setting comparison by adding the two previous tables together. Now, both $2 \times 10^{-1}$, $2 \times 10^{-5}$ appear dominated by the other settings.

| | $2 \times 10^{-1}$ | $2 \times 10^{-2}$ | $2 \times 10^{-3}$ | $2 \times 10^{-4}$ | $2 \times 10^{-5}$ |
|---|---|---|---|---|---|
| $2 \times 10^{-1}$ | | 1 | 2 | 1 | -1 |
| $2 \times 10^{-2}$ | -1 | | 0 | 0 | -1 |
| $2 \times 10^{-3}$ | -2 | 0 | | -1 | -1 |
| $2 \times 10^{-4}$ | -1 | 0 | 1 | | -1 |
| $2 \times 10^{-5}$ | 1 | 1 | 1 | 1 | |

The final step is to sum down the columns to obtain a single score for each setting. Table 4.5 shows the final results of summing down the columns of Table 4.4. This is where a researcher can apply their own thresholds to determine what settings to keep or not. A good rule-of-thumb is to remove variables that are $\leq -S + 2$ where $S$ is the number of settings in that multi-category hyperparameter. In this case, the threshold is $\leq -3$, so learning rates of $2 \times 10^{-1}$ and $2 \times 10^{-5}$ would be removed from the next random search round. The researcher can adjust this threshold depending on how conservative or aggressive they want to be as they manage their time-limit and computational budgets.

## Continuous Hyperparameter

The dropout rate is a continuous hyperparameter optimized in this study and is used here to show an example of applying RSSR to a continuous variable. Applying a similar methodology as above directly to continuous settings is more challenging. Figure 4.4 shows the dropout

Table 4.5: This table is the final step in the statistical comparison process. It sums down the columns of the previous table to create a metric for each setting that can be used to determine which should remain or be removed. A learning rate of $2 \times 10^{-3}$ appears to perform the best and should definitely remain, while $2 \times 10^{-1}$ and $2 \times 10^{-5}$ should be considered for removal.

| | $2 \times 10^{-1}$ | $2 \times 10^{-2}$ | $2 \times 10^{-3}$ | $2 \times 10^{-4}$ | $2 \times 10^{-5}$ |
|---|---|---|---|---|---|
| Column Sum | -3 | 2 | 4 | 1 | -4 |

rate versus validation Dice for all trials and only-good trials. Visually, there appears to be a trend where higher dropout rates lead to higher validation Dice scores for only-good trials. However, there is no straightforward statistical approach to justify changing the dropout rate range from this visual inspection. To conform to the established strategy, the dropout rate was converted into a categorical variable with five bins and each of a size of 0.1.

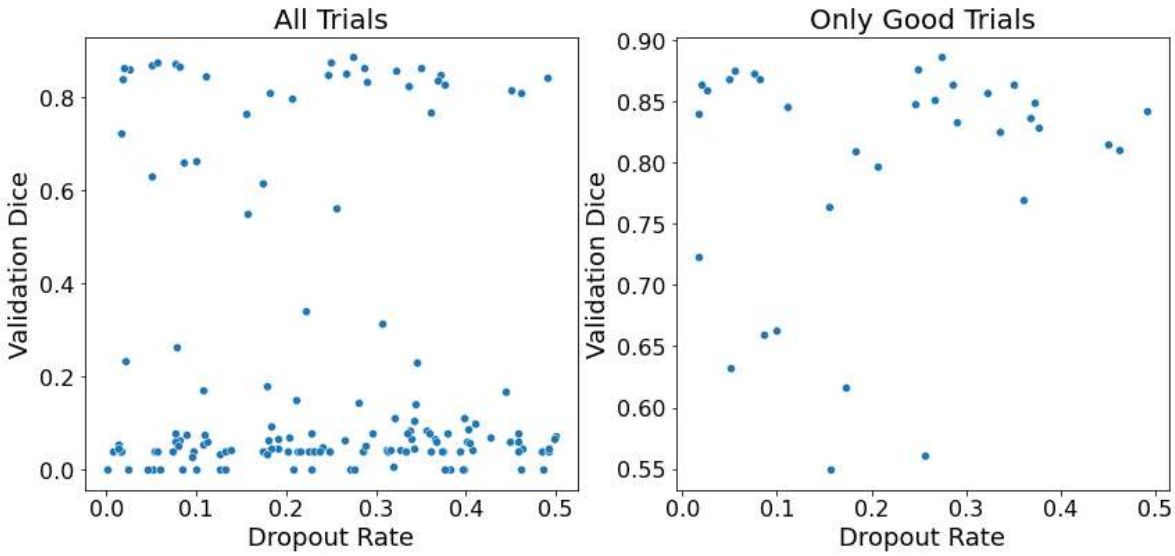



Figure 4.4: This figure shows the validation Dice results for systolic blood pressure symbol detection by continuous dropout rate. The chart on the left shows all 147 trials, while the chart on the right is only the 34 good trials. The right chart shows a minor correlation of higher dropout rates producing higher validation Dice results.

Figure 4.5 shows the updated charts after dropout rate is converted to a categorical variable. It still appears that dropout rates above 0.2 achieve high validation Dice scores on only-good trials, but now the sample size can be incorporated to statistically compare these settings as was done before.

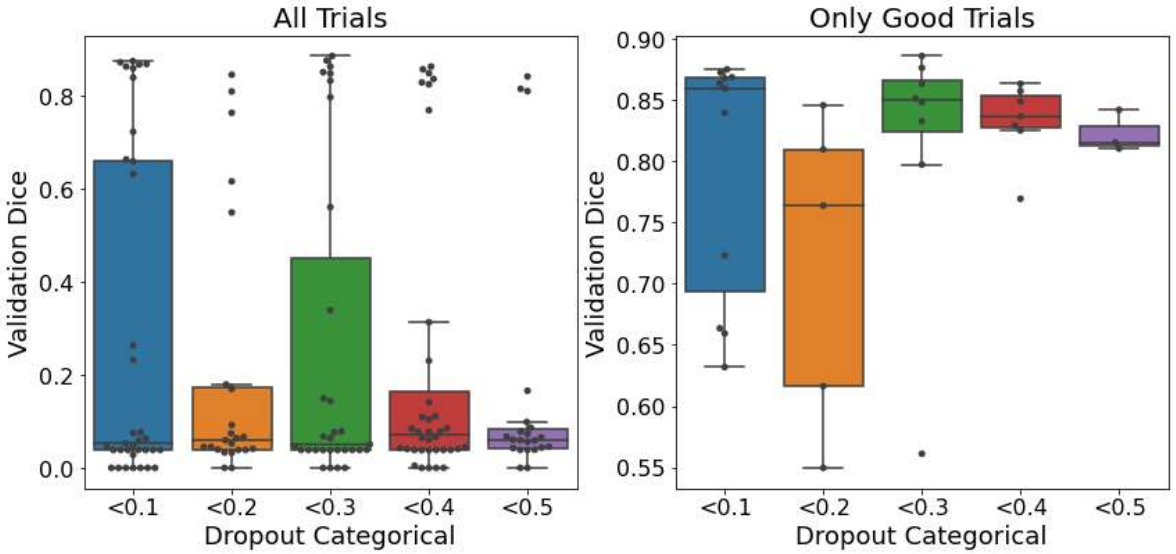



Figure 4.5: This figure shows the validation Dice results for systolic blood pressure symbol detection by categorical dropout rate. By converting into categorical, the established statistical comparison methods can be implemented. The right chart still shows a trend with higher dropout rate, but the sample sizes can be incorporated to determine if this trend is statistically significant.

Table 4.6 shows the final results of the categorical dropout rate comparison. Even though higher dropout rates showed some benefit, there was not enough statistical evidence to remove dropout rates below 0.2 from the search space. The 0.1 to 0.2 bin did reach close to the threshold, so it could be considered for removal, but overall the consensus is that there are no dropout rates that appear to maximize validation Dice at this point in this example.

Table 4.6: This tables shows the final results of the statistical comparison between the five categorical dropout rates. Even though there did appear to be a minor dropout rate and validation Dice trend, it was not a strong of difference to warrant eliminating dropout settings for future random search testing. The safest conclusion is to continue to search the entire dropout rate range.

| | $2 \times 10^{-1}$ | $2 \times 10^{-2}$ | $2 \times 10^{-3}$ | $2 \times 10^{-4}$ | $2 \times 10^{-5}$ |
|---|---|---|---|---|---|
| Column Sum | 0 | -2 | 1 | 1 | 0 |

## 4.5 Gaussian Mixture with Epsilon-Greedy (GMEG)

Another issue with existing SMBO techniques is when the acquisition function becomes stuck at testing the same area repeatedly and rarely exploring. Each approach is designed to have some exploration and exploitation balance, but occasionally the previous trial results contain outliers or is skewed in such a way to create this scenario. For instance, GP-UCB selects the settings that have the highest UCB, but still tests same setting for many trials. Perhaps because one of the settings that produced the highest achieved validation Dices also had many bad trials at similar settings. The UCB is high because of some examples that did well, but it also very uncertain due to the bad trials in the vicinity. There is some evidence that these techniques perform almost the same trial over 100 times in a row which results in the model just trying to optimize the random initial weights of the model.

One way to ensure that the approach continues to explore as trial results accumulate is to implement an $\epsilon$-greedy approach. The $\epsilon$-greedy method has origins in multi-armed bandit and reinforcement learning problems [74]. The $\epsilon$ part refers to a probability that the agent will take a random action next. A random number is drawn and if this number is less than $\epsilon$, then a completely random action will be taken to force the agent to potentially explore new or rarer states. If the random number exceeds $\epsilon$, then the agent is takes the exploitation route and acts "greedily" by selecting the action that currently maximizes reward. In hyperparameter optimization context, exploration opens the possibility to test any combination of settings at a rate of $\epsilon$ and exploitation tests the setting(s) that have the best current validation Dice results.

A first glance, it seems simple and straightforward implementing $\epsilon$-greedy for hyperparameter optimization, but there are important considerations to make sure it does not have the same issues as other SMBO techniques. As $\epsilon$ increases, the technique approaches random search, so epsilon should be adjusted to create a good balance. When exploiting, the method should do more than just re-testing the current best setting, because then the technique is just repeating trials again which is what this method is trying to avoid. In a reinforcement learning approach, a state space is developed so that rewards can be assigned to states. Actions are the agents way of moving from state to state. It is possible to construct a tabular state space from hyperparameter settings if continuous variables are converted into categorical, but this could lead to over 1,000 different states. Completing the tabular entire state space with reward value is basically a fully-exhaustive grid search which is an infeasible approach. Therefore, it is best to model the state space as a function and then improve the approximation of this function as trial results are obtained.

Linear models and neural networks were considered for mapping hyperparameters to the validation Dice reward, because these models are differentiable and the extracted gradient could dictate the next setting to test. However, hyperparameter optimizations start from scratch with no trials and are non-linear search spaces, so modeling the state space with linear models or neural networks do not seem plausible at the moment. The workaround that was developed was to treat the validation Dice coefficient as another independent variable and train a GMM (unsupervised approach) onto validation Dice and the five hyperparameter

settings to create a combined state space and reward representation [77]. The multivariate Gaussians in this mixture model each be described by mean of the validation Dice scores of the trials that labeled within each cluster. During the exploitation phase, a multivariate random setting is drawn from the cluster that has the highest mean validation Dice. The random validation Dice pulled during this is step is ignored. This setup ensures that the exploitation phase is produced settings in the most beneficial areas, but is very unlikely to continually choose the same setting. Figure 4.6 shows an example of a GMM ($k = 7$) fit to validation Dice and dropout rate.

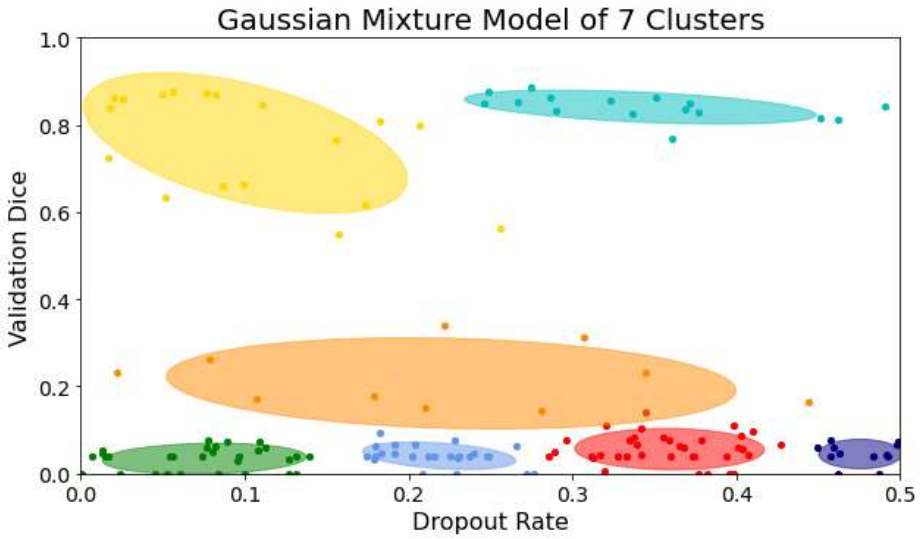



Figure 4.6: This figure shows the validation Dice results for systolic blood pressure symbol detection by continuous dropout rate. The colored ellipses show where the seven Gaussian clusters are located with results from all completed trial runs. The top-right cluster has the highest mean validation Dice, so the greedy action is to sample from this distribution. The exploitation dropout rate will likely be in the 0.3 to 0.4 range using a multi-variate normal random number generator. This methodology was also flexible for handling bad runs as about 5 clusters formed in the low response area.

The main advantage of this technique was user-flexibility to control the rate of exploration and exploitation. However, this technique has many of its own hyperparameters that need to be considered. With traditional $\epsilon$-greedy, the $\epsilon$ value must be determined. There are many other $\epsilon$-greedy based techniques such as Greedy in the Limit of Infinite Exploration (GLIE) where $\epsilon$ reduces as the number of completed trials increases [67]. The number of Gaussians or clusters to use was also adjustable. This number should be a function of the number of trials completed in order to not over or under fit the data. The following function was a conservative approach to determine the number of clusters:

$$k = \lfloor \frac{\sqrt{n}}{2} + 1 \rfloor \tag{4.1}$$

where $k$ was the number of GMM clusters to fit and $n$ was the current number of completed trials. Another more aggressive approach was created that increases the number of clusters at a higher rate:

$$k = \lceil \sqrt{n} - 1 \rceil \tag{4.2}$$

where $k$ and $n$ are defined the same. It seemed that the number of clusters could impact the performance of the Gaussian Mixture with Epsilon-Greedy (GMEG) the most, so the number clusters from both equations are tested. Figure 4.7 shows an example of modeling dropout rate and validation Dice with 12 clusters (Equation 4.2) instead of 7. Finally, the major disadvantage with GMEG was that all hyperparameters must be convertable into continuous values to be fit by a GMM. Even though hyperparameters like batch size or number of filters are typically restrictive integers, they can be treated as continuous within the GMM and then converted back to categorical once the random setting was obtained from $\epsilon$-greedy. A category was assigned based on which value of the category was closest to the continuous random sample. The learning rate was converted into integers from $-1$ to $-5$, so that all categories are evenly spaced. The batch normalization was converted into 1 and 0 for "True" and "False", respectively. Unfortunately, a hyperparameter that cannot be converted into a continuous variable, such as which optimizer to apply, would have to be optimized outside of the GMEG construct. Unlike the RSSR technique, the GMEG can account for interaction effects between hyperparameters.

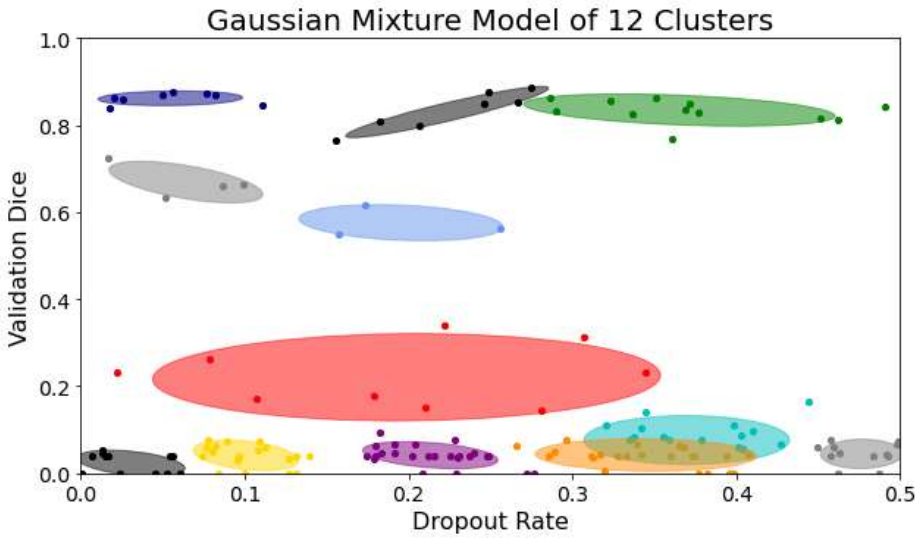



Figure 4.7: This figure shows the validation Dice results for systolic blood pressure symbol detection by continuous dropout rate. The colored ellipses show where the 12 Gaussian clusters were located with results from all completed trial runs. it appears the top-left cluster was now the greedy option.

# Chapter 5

# Results

This chapter presents the results from the experiments to optimize the validation dataset Dice coefficient over the seven total segmentation models. First, the optimization techniques were compared by plotting the maximum validation Dice found by the time scenario for each dataset. Next, this comparison was converted into a weighted rankings system to provide a scoring approach to see which optimization techniques perform better with limited time and on certain datasets. Then, we will assess which hyperparameter settings were selected as optimal and determine if there were any trends by dataset. Finally, the runtime length elapsed to find the maximum validation Dice will be used to evaluate the Pareto efficiency of the various optimizations.

## 5.1 Comparison of Optimization Techniques

In this section, maximum validation Dice coefficient was utilized to compare the optimize techniques across time scenarios and datasets. The scenarios with a longer runtime were expected to achieve a higher performance than shorter scenarios since more trials can be performed to find better settings. The main interest was which techniques reach near-optimal accuracy as earlier as possible, because this was an indicator of efficiency of each approach. For the optimization techniques, GMEG 1 refers to using Equation 4.1 to generate the number of clusters per trial, while GMEG 2 was applying Equation 4.2. All other techniques used the same acronyms as displayed in Chapter 4. GMEG 1 was not tested at the 24-hour scenario due to computational constraints. GMEG 2 was only tested for one scenario and performed poorly compared to GMEG 1. RSSR requires enough random search samples to being the statistical comparisons. For each RSSR method tested, a pure random search was executed until the halfway point in the time limit. The settings were reduced as necessary and then the remainder half of the scenario was executed with this reduced settings. RSSR was only tested up until 16 hours, because there was not a 12-hour random search scenario to use as halfway assessment for the 24-hour scenario.

## Performance by Dataset

The following charts plot the best validation Dice coefficient obtained from each individual experiment by dataset and time scenario. There were seven total datasets and models evaluated and following was the notation used:

- Eosinophilic Esophagitis (EoE) eosinophil detection
- Cardiovascular (CV) nuclei detection
- CV Smooth Muscle Cell (SMC) detection
- CV macrophage detection
- Graph Reading (GR) Heart Rate (HR) detection
- GR Diastolic Blood Pressure (DBP) detection
- GR Systolic Blood Pressure (SBP) detection

Figure 5.1 shows the hyperparameter optimization results for the EoE eosinophil detection dataset. The best validation Dice found was 0.705 during the 4-hour scenario with the GP-UCB technique. The GP-UCB approach was able to reach a high accuracy in short time with this dataset. GMEG 1 also performed well at the 4 and 8-hour scenarios. All techniques appear to approach a validation Dice in the 0.67 to 0.71 range as time progress.

Figure 5.2 shows the hyperparameter optimization results for the CV dataset for detecting nuclei. The best validation Dice found was 0.724 during the 16-hour scenario with the Hyperband technique. Hyperband performed poorly in the 2 and 4-hour scenarios, but did well from 8 to 16-hours. The RSSR technique also performed well at 8 and 16-hour scenarios.

Figure 5.3 shows the hyperparameter optimization results for the CV dataset for detecting SMC. The best validation Dice found was 0.537 during the 24-hour scenario with the GP-UCB technique. Random search performed best during the shorter length scenarios.

Figure 5.4 shows the hyperparameter optimization results for the CV dataset for detecting macrophages. The best validation Dice found was 0.727 during the 24-hour scenario with the RS technique. RSSR had success at the 8 and 16-hour scenarios and RS did well throughout all scenarios.

Figure 5.5 shows the hyperparameter optimization results for the vital signs graph dataset for detecting heart rates. The best validation Dice found was 0.867 during the 8-hour scenario with the RSSR technique. RSSR had success at the 8 and 16-hour scenarios, but GP-UCB also did well in the short time scenarios. The GMEG 1 approach did very poorly at the 2 and 4-hour scenarios.

Figure 5.6 shows the hyperparameter optimization results for the vital signs graph dataset for detecting diastolic blood pressures. The best validation Dice found was 0.869 during the 24-hour scenario with the random search technique. GP-UCB did well at the 2-hour scenario, but RSSR and RS perform better at the other time scenarios.

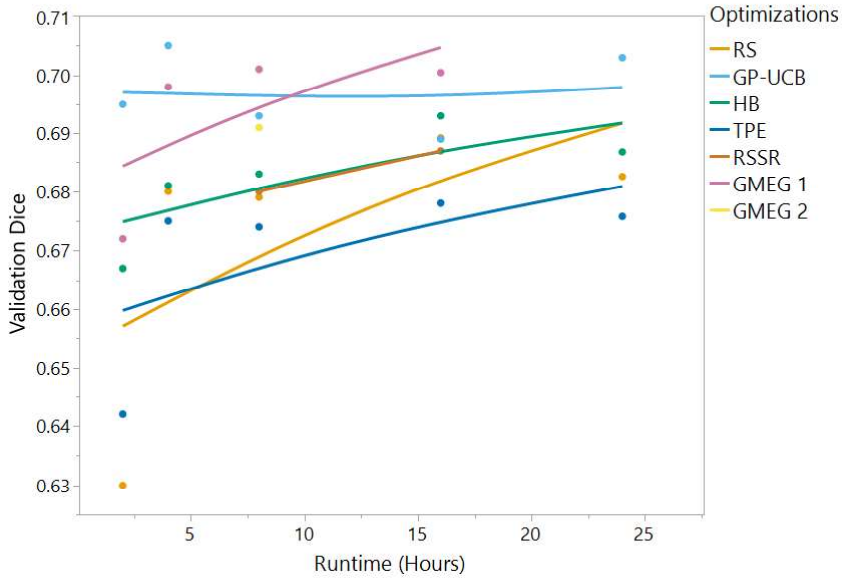



Figure 5.1: Performance Results by EoE Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that GP-UCB and GMEG 1 were best-suited for the EoE dataset.

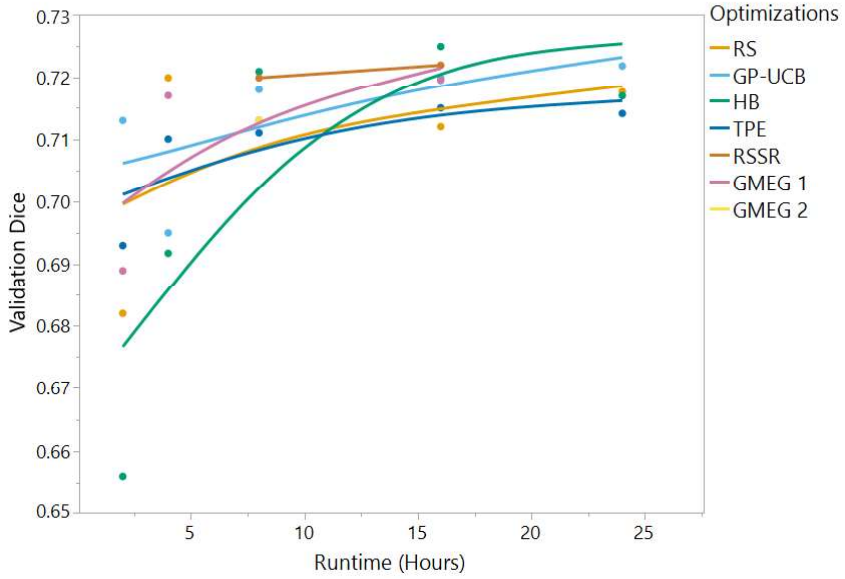



Figure 5.2: Performance Results by Cardiovascular Nuclei Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that Hyperband and RSSR were best-suited for the CV Nuclei dataset.

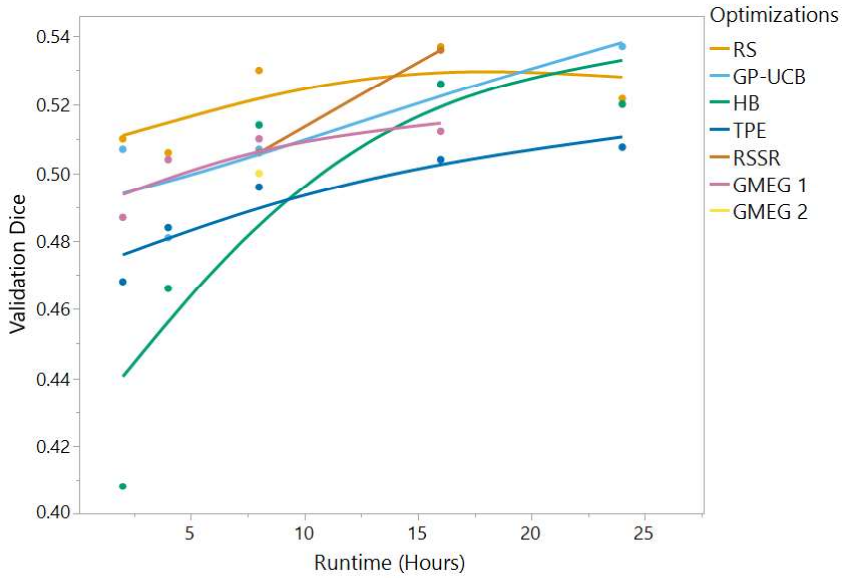



Figure 5.3: Performance Results by Cardiovascular SMC Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that random search and RSSR were best-suited for the CV SMC dataset.

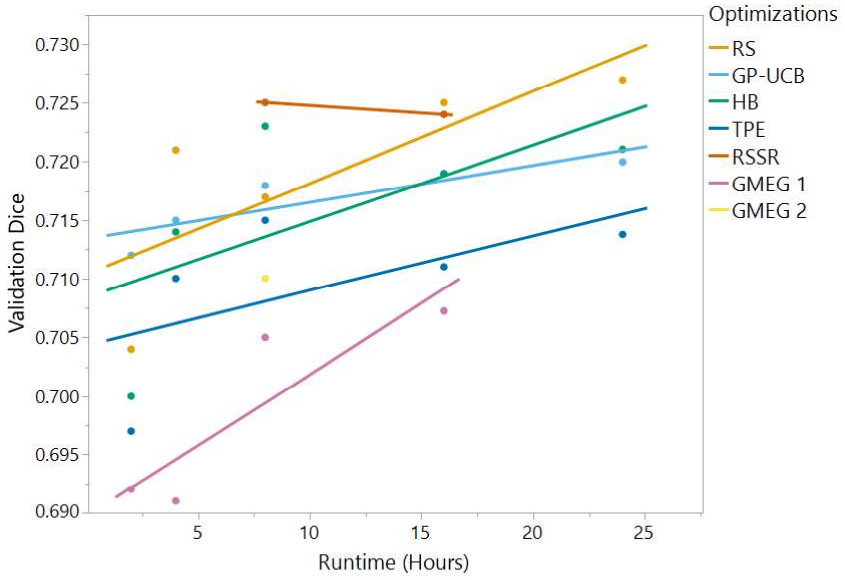



Figure 5.4: Performance Results by Cardiovascular Macrophage Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that random search and RSSR were best-suited for the CV macrophage dataset.

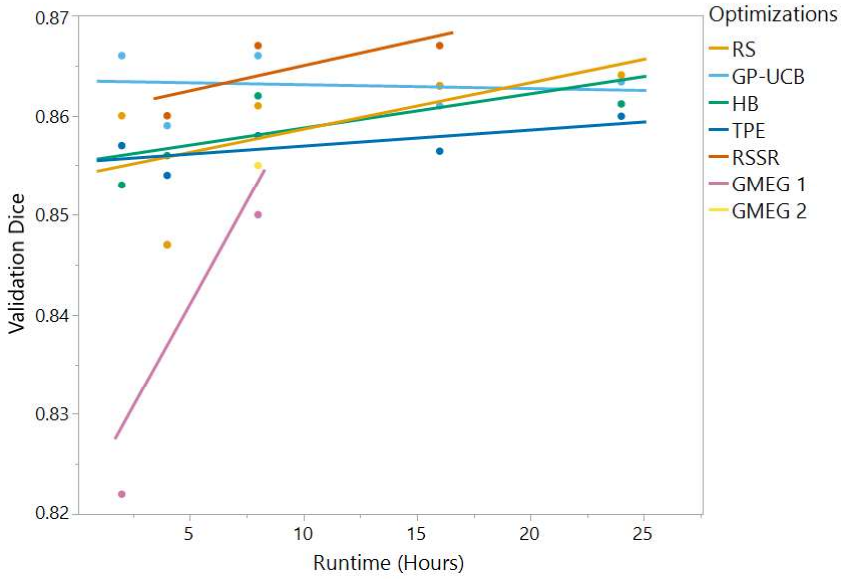



Figure 5.5: Performance Results by Vital Signs Graph Heart Rate Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that RSSR and GP-UCB were best-suited for the GR HR dataset.

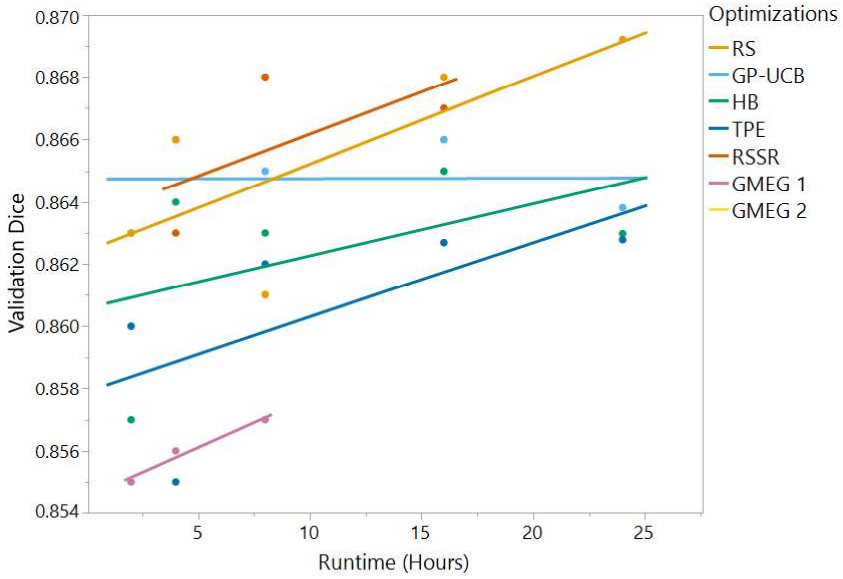



Figure 5.6: Performance Results by Vital Signs Graph Diastolic BP Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that RSSR and RS were best-suited for the GR DBP dataset.

Figure 5.6 shows the hyperparameter optimization results for the vital signs graph dataset for detecting systolic blood pressures. The best validation Dice found was 0.892 during the 8-hour scenario with the RSSR technique. RSSR performed very well across 4 to 15-hour scenarios. All techniques had comparable performance except for HB and GMEG 1 at earlier scenarios.

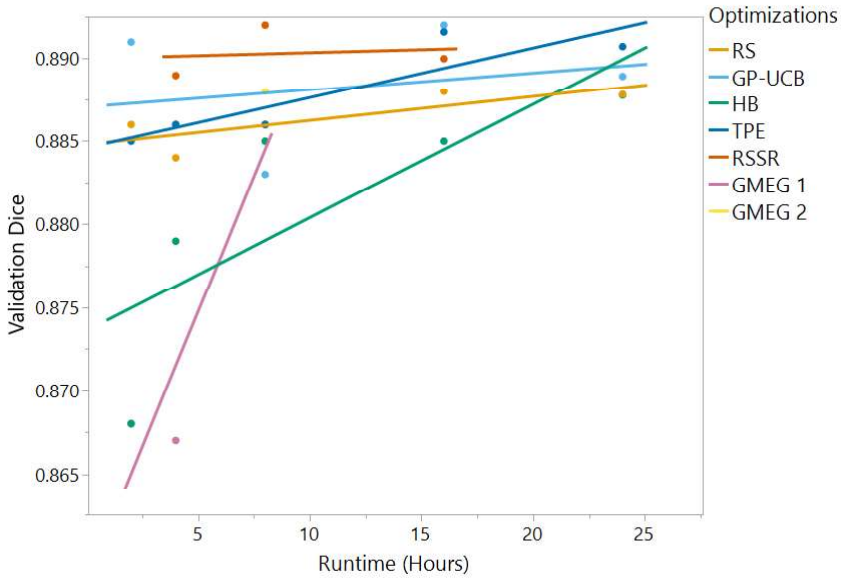



Figure 5.7: Performance Results by Vital Signs Graph Systolic BP Dataset. This figure compares of the performance of the optimization techniques at each time scenario. It appears that RSSR was best-suited for the GR DBP dataset.

## Time Scenario Rankings

The performance results were just presented visually and by dataset, but it was still questionable which optimizations were best for particular time constraints and datasets. Therefore, a methodology was created to score each technique based on the ranking of the maximum validation Dice found when using each technique. The following tables show the validation Dice rankings for each optimization technique across the dataset. For example, a ranking of 5 for RS technique on the EoE dataset means that the RS technique produced the fifth highest validation Dice coefficient on the EoE dataset. A weighting vector was applied with a sumproduct calculation to create an overall technique score for each optimization. Each root dataset (EoE, CV, and GR) were equally worth 1/3 and then the subset datasets within CV and GR were also equally weighted with 1/3 (1/9 overall). The technique with the lowest score using this methodology was considered was considered the best overall approach at that specific time scenario. If the validation Dice coefficients were equivalent at the 3-

decimal place level, then the techniques were considered tied and then average ranking was distributed between the ties.

Table 5.1 shows the weighted ranking scores for the 2-hour time scenario. GP-UCB clearly dominates this scenario with five #1 rankings and never had a ranking worst than second place. It was surprising that GP-UCB does so well at the shortest time scenario, because it was still within the warm-up period ($\leq$ 15 trials) for the EoE and CV datasets.

Table 5.1: 2-Hour Scenario Performance Comparison. This table shows the weighted ranking scores for each optimization technique. GP-UCB clearly dominates the other optimizations at the 2-hour scenario.

| **Dataset** | **RS** | **GP-UCB** | **HB** | **TPE** | **GMEG 1** | **Weight** |
|---|---|---|---|---|---|---|
| EoE | 5 | 1 | 3 | 4 | 2 | 0.333 |
| CV Nuclei | 4 | 1 | 5 | 2 | 3 | 0.111 |
| CV SMC | 1 | 2 | 5 | 4 | 3 | 0.111 |
| CV Macrophage | 2 | 1 | 3 | 4 | 5 | 0.111 |
| GR HBR | 2 | 1 | 4 | 3 | 5 | 0.111 |
| GR DBP | 1.5 | 1.5 | 4 | 3 | 5 | 0.111 |
| GR SBP | 2 | 1 | 4.5 | 3 | 4.5 | 0.111 |
| Sumproduct | 3.06 | **1.17** | 3.83 | 3.44 | 3.50 | 1 |

Table 5.2 shows the weighted ranking scores for the 4-hour time scenario. GP-UCB performs the best overall again, but did not dominate all datasets. Random search was actually ranked first for all of the CV datasets which closed the overall scoring gap between these two techniques.

Table 5.2: 4-Hour Scenario Performance Comparison. This table shows the weighted ranking scores for each optimization technique. GP-UCB scored the best overall, but RS dominates the entire CV dataset.

| **Dataset** | **RS** | **GP-UCB** | **HB** | **TPE** | **GMEG (#1)** | **Weight** |
|---|---|---|---|---|---|---|
| EoE | 4 | 1 | 3 | 5 | 2 | 0.333 |
| CV Nuclei | 1 | 4 | 5 | 3 | 2 | 0.111 |
| CV SMC | 1 | 4 | 5 | 3 | 2 | 0.111 |
| CV Macrophage | 1 | 2 | 3 | 4 | 5 | 0.111 |
| GR HBR | 4.5 | 1 | 2 | 3 | 4.5 | 0.111 |
| GR DBP | 3 | 1.5 | 4 | 1.5 | 5 | 0.111 |
| GR SBP | 1.5 | 1.5 | 3 | 5 | 4 | 0.111 |
| Sumproduct | 2.67 | **1.89** | 3.44 | 3.83 | 3.17 | 1 |

Table 5.3 shows the weighted ranking scores for the 8-hour time scenario. As the amount of runtime increases, it becomes harder to find techniques that stand out above the rest. Overall, RSSR achieved the best weighted score and dominated the entire GR dataset, but also was ranked 5th for EoE and CV SMC. GP-UCB and Hyperband were close in overall score, but only had one first place ranking between them.

Table 5.3: 8-Hour Scenario Performance Comparison. This table shows the weighted ranking scores for each optimization technique. RSSR scored the best overall, but the performance was very inconsistent across all datasets.

| Dataset | RS | RSSR | GP-UCB | HB | TPE | GMEG 1 | GMEG 2 | Weight |
|---|---|---|---|---|---|---|---|---|
| EoE | 6 | 5 | 2 | 4 | 7 | 1 | 3 | 0.333 |
| CV Nuclei | 5 | 2 | 3 | 1 | 7 | 5 | 5 | 0.111 |
| CV SMC | 1 | 5 | 4 | 2 | 7 | 3 | 6 | 0.111 |
| CV Macrophage | 4 | 1 | 3 | 2 | 5 | 7 | 6 | 0.111 |
| GR HBR | 4 | 1 | 2 | 3 | 5 | 7 | 6 | 0.111 |
| GR DBP | 6 | 1 | 2 | 3 | 4 | 7 | 6 | 0.111 |
| GR SBP | 4 | 1 | 7 | 6 | 4 | 4 | 2 | 0.111 |
| Sumproduct | 4.67 | **2.89** | 3.00 | 3.22 | 5.89 | 4.00 | 4.44 | 1 |

Table 5.4 shows the weighted ranking scores for the 16-hour time scenario. RS was now the top overall score and this makes sense, because at higher time-limits RS was able to explore many hyperparameter setting possibilities. It was possible that success of GP-UCB success tapers off as time increases, because it can get stuck exploiting the same setting rather than utilizing the large number of trials for exploration. RSSR and HB also did well at 16-hours and each had a first-place ranking.

Table 5.5 shows the weighted ranking scores for the 24-hour time scenario. GP-UCB did best on shorter scenarios and now was the top scorer on the longest scenario. Perhaps having an additional 8 to 16 hours worth of additional trials will eventually lead to some exploration for the GP-UCB. Depending on the situation, GP-UCB either continues to search the entire hyperparameter space or becomes stuck exploiting the same area, so that could explain the inconsistency.

## 5.2 Analysis by Hyperparameter

In this section, we assessed which settings resulted in the maximum validation Dice for each dataset. Without hyperparameter optimization, a researcher may have a default set of hyperparameters that they apply every time they use a U-Net regardless of the dataset. This analysis revealed whether certain datasets preferred certain hyperparameter settings which can confirm or deny the need for hyperparameter optimization. Each optimization

Table 5.4: 16-Hour Scenario Performance Comparison. This table shows the weighted ranking scores for each optimization technique. RSSR scored the best overall, but the performance was inconsistent across all datasets.

| **Dataset** | **RS** | **RSSR** | **GP-UCB** | **HB** | **TPE** | **Weight** |
|---|---|---|---|---|---|---|
| EoE | 2.5 | 4 | 2.5 | 1 | 5 | 0.333 |
| CV Nuclei | 4 | 1 | 2 | 5 | 3 | 0.111 |
| CV SMC | 1 | 2 | 3 | 4 | 5 | 0.111 |
| CV Macrophage | 1 | 2 | 4 | 3 | 5 | 0.111 |
| GR HBR | 2.5 | 1 | 4 | 2.5 | 5 | 0.111 |
| GR DBP | 1 | 2 | 3 | 4 | 5 | 0.111 |
| GR SBP | 4 | 3 | 1.5 | 5 | 1.5 | 0.111 |
| Sumproduct | **2.33** | 2.56 | 3.00 | 2.94 | 4.39 | 1 |

Table 5.5: 24-Hour Scenario Performance Comparison. This table shows the weighted ranking scores for each optimization technique. GP-UCB scored the best overall, but RS was also the best ranking for three of the seven datasets.

| **Dataset** | **RS** | **GP-UCB** | **HB** | **TPE** | **Weight** |
|---|---|---|---|---|---|
| EoE | 3 | 1 | 2 | 4 | 0.333 |
| CV Nuclei | 2 | 1 | 3 | 4 | 0.111 |
| CV SMC | 2 | 1 | 3 | 4 | 0.111 |
| CV Macrophage | 1 | 3 | 2 | 4 | 0.111 |
| GR HBR | 1 | 2 | 3 | 4 | 0.111 |
| GR DBP | 1 | 2 | 3.5 | 3.5 | 0.111 |
| GR SBP | 3.5 | 2 | 3.5 | 1 | 0.111 |
| Sumproduct | 2.17 | **1.56** | 2.67 | 3.61 | 1 |

run results in a different maximum validation Dice, but it was assumed that those varying results were not a major factor in the following charts. Although settings obtained through longer scenarios that produced the maximum validation Dice coefficients would have the most reliable settings.

Figure 5.8 shows in how many total optimization runs were "True" or "False" batch normalization selected as the setting that produced the maximum validation Dice coefficient. Across the board, batch normalization was beneficial to producing higher validation accuracy.

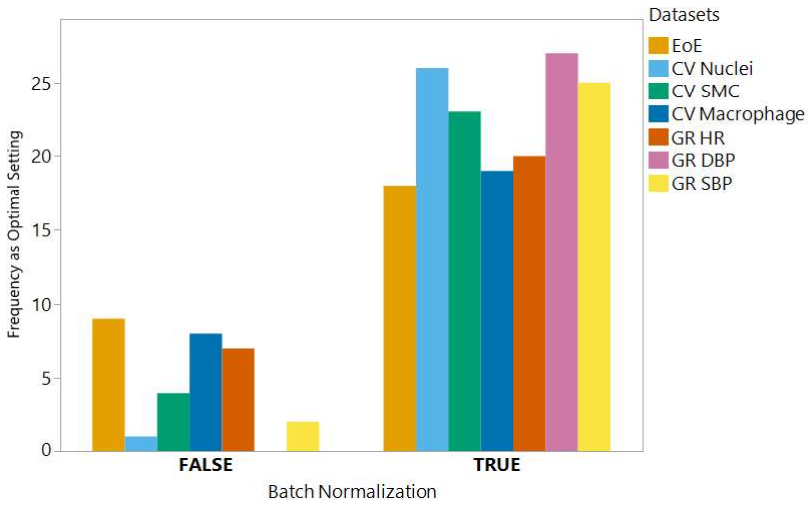



Figure 5.8: Batch Normalization Selection as the Best Setting. It was clear that batch normalization should be set to True and remain active throughout the neural network architecture for all datasets.

Figure 5.9 shows in how many total optimization runs were each learning rate option selected as the setting that produced the maximum validation Dice coefficient. A learning rates of $2 \times 10^{-3}$ and $2 \times 10^{-4}$ appear to be the most popular selections, but there were some trends per dataset. EoE greatly prefers a $2 \times 10^{-4}$ learning rate, while GR datasets mainly lead to learning rates of $2 \times 10^{-3}$. The extreme learning rate on both sides were rarely ever selected as the highest performing.

Figure 5.10 shows in how many total optimization runs were each initial number of filters option selected as the setting that produced the maximum validation Dice coefficient. These results were actually spread-out fairly evenly besides some trends within the datasets. The results here debunks any notion that more complex models with more filters or parameters will always perform better. It was possible that a model of lesser complexity will generalize better to outside data.

Figure 5.11 shows in how many total optimization runs were each batch size option selected as the setting that produced the maximum validation Dice coefficient. It was exceedingly clear that the GR datasets produce higher validation accuracies when batch size equalled two. Sometimes, the inclination was to set the batch size as high a possible to

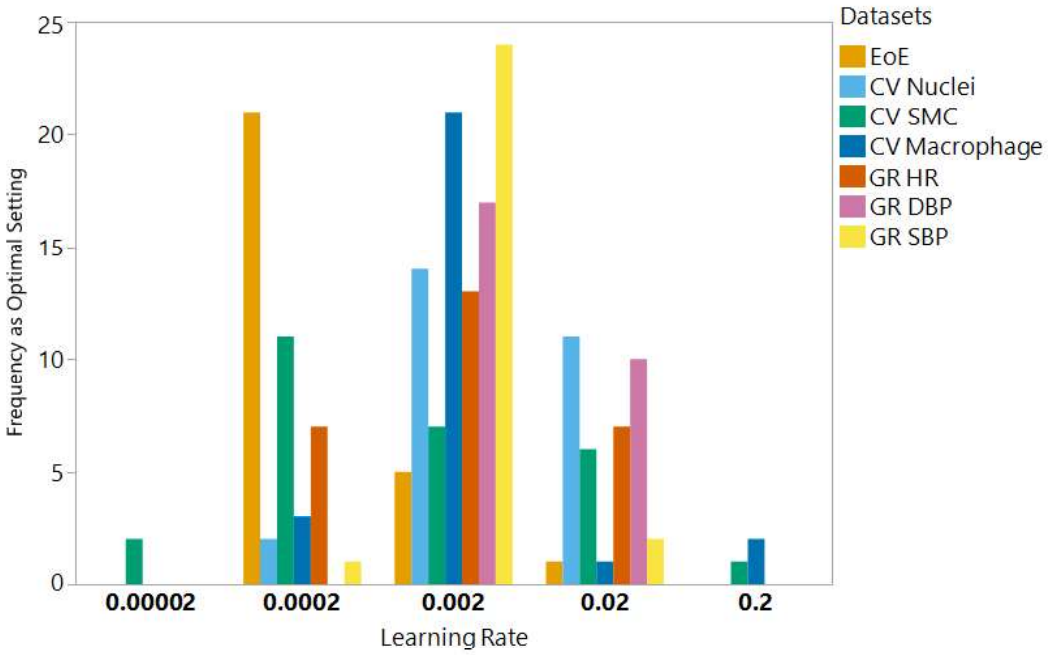



Figure 5.9: Learning Rates Selection as the Best Setting. Learning rates of $2\times10^{-2}$, $2\times10^{-3}$, and $2\times10^{43}$ were the most popular, but there were some trends for which of these the datasets prefer.

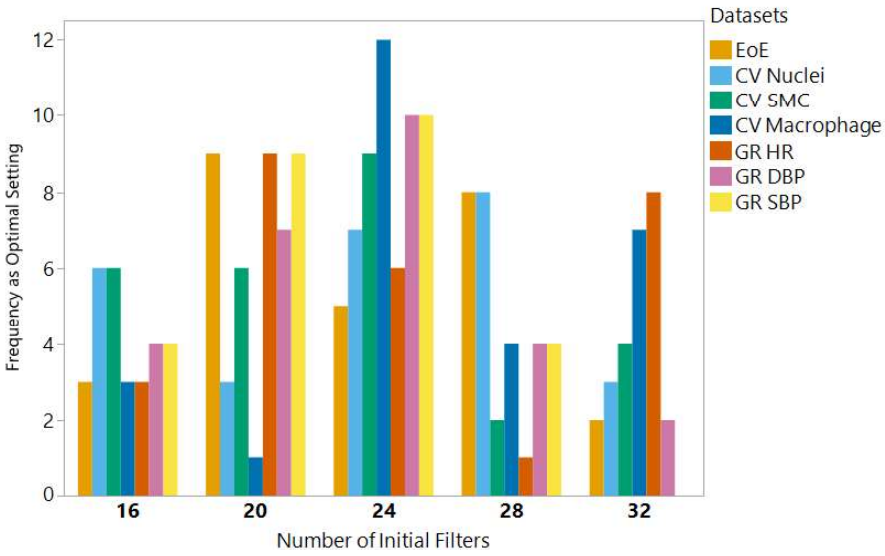



Figure 5.10: Number of Initial Filters Selection as the Best Setting. The middle number of filter option, 24, was selected the most, but the overall distribution of selected setting was fairly flat. This does show that more filters was not always better for model generalization.

quicken model training, but this shows that lower batch sizes can also be beneficial. The GR datasets have a very small sample size, so maybe that was the reason why smaller batches were preferred.

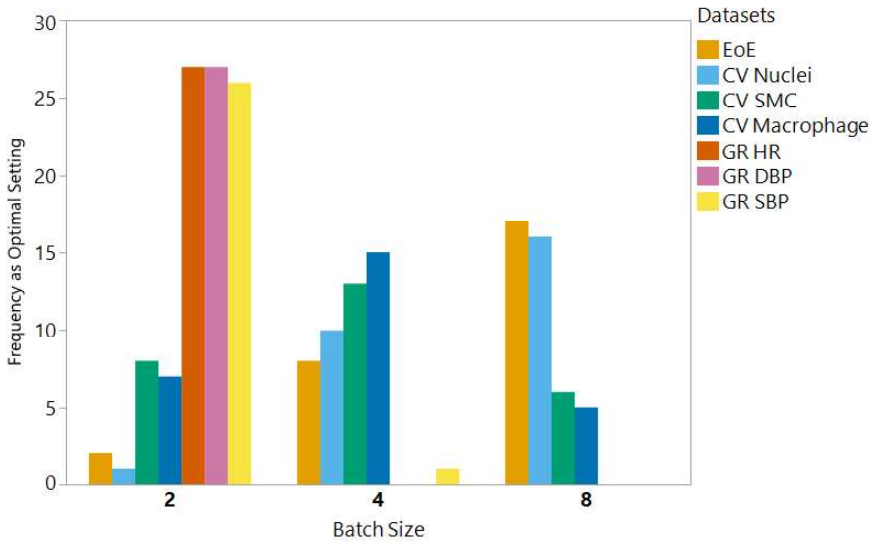



Figure 5.11: Batch Size Selection as the Best Setting. Batch size appeared highly dependent on which dataset was being used. The GR data greatly preferred a batch size of 2, while the others fluctuated between 4 or 8.

Lastly, Figure 5.12 shows the distribution of dropout rates selected as the setting that produced the maximum validation Dice coefficient. Dropout rate was shown across datasets, as well as, optimization technique. The dropout rate selections were very wide-ranging and rarely did any particular dropout rate seem to align with a dataset or optimization technique. There were a couple datasets, CV nuclei and GR DBP, that may associate with higher dropout rates. There were some studies that claim that a both dropout rate and batch normalization needed in the same network and could actually decrease performance [44]. Since batch normalization was almost always useful according to the results from Figure 5.8, perhaps this negated the effects of dropout rate.

## 5.3 Pareto Efficiency of Optimizations

This final section of this chapter examines the Pareto efficiency of each optimization technique [17]. Hyperparameter optimization techniques can be compared based on how quickly each reached a high validation Dice coefficient. The Pareto frontier represented a balance between quickness and lowered computation with validation accuracy performance. The quicker a near-optimal solution was found, the less computation resources were needed if the optimization was halted shortly after finding this value. The Section 5.1 compared per-

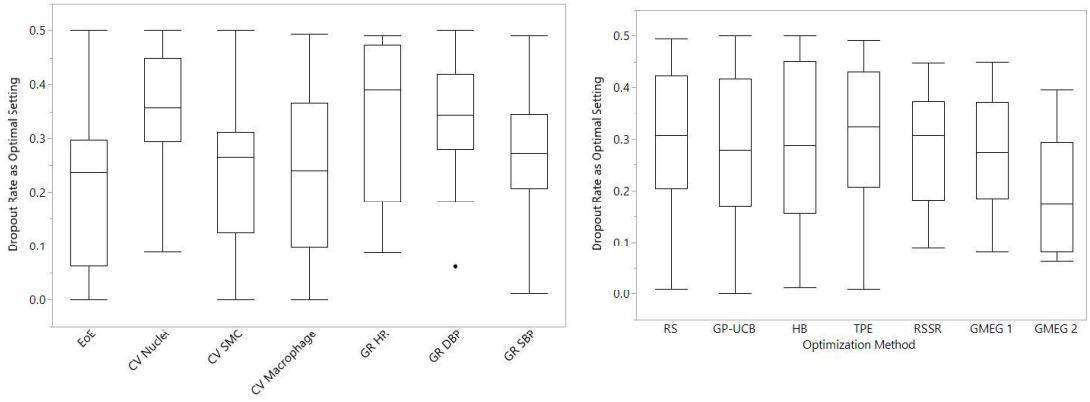



Figure 5.12: Dropout Rate Select as the Best Setting. No particular dropout rate appeared to correspond with higher performance, but there were a few datasets that seemed to prefer higher dropout rates.

formance of the optimizations by the time-limit of each scenario, but it was likely that the eventual maximum Validation Dice was found much sooner during the optimization. For SMBO techniques, it was easy to monitor performance per trial and determine when accuracy has leveled-off. For random search, the maximum validation Dice could occur at any moment, so it was harder to be certain when the optimization has plateaued. For the following charts, the elapsed optimization time when the maximum validation Dice was found was extracted from the data. This time was compared with Dice to from a Pareto frontier of options. Longer run times typically produced higher validation accuracies, but there should still be a sweet-spot for achieving performance while staying computationally efficient.

Figure 5.13 shows the Pareto efficiency of the optimization techniques on the EoE dataset. The GP-UCB and GMEG techniques were mostly located in the top-left of the chart which was the Pareto frontier in this case. While these techniques were set to run from 2 to 24 hours depending on the scenario, the near-optimal solution can be obtained quickly due to a fortunate random trial during the warm-up period or due to the benefits of the sequential process.

Figure 5.14 shows the Pareto efficiency of the optimization techniques on the CV nuclei dataset. The GMEG 1 optimization was also Pareto efficient on the CV nuclei dataset.

Figure 5.15 shows the Pareto efficiency of the optimization techniques on the CV SMC dataset. The RS optimization has three optimizations near the Pareto frontier. It was possible that a RS finds a high-performance at an early trial, but random search is never considered an efficient approach. If an sufficient accuracy was found early on, the random search could be terminated early, but otherwise the search would continue for the entire time scenario.

Figure 5.16 shows the Pareto efficiency of the optimization techniques on the CV macrophage dataset. RS also did well with Pareto efficiency on the macrophage data, but there were also

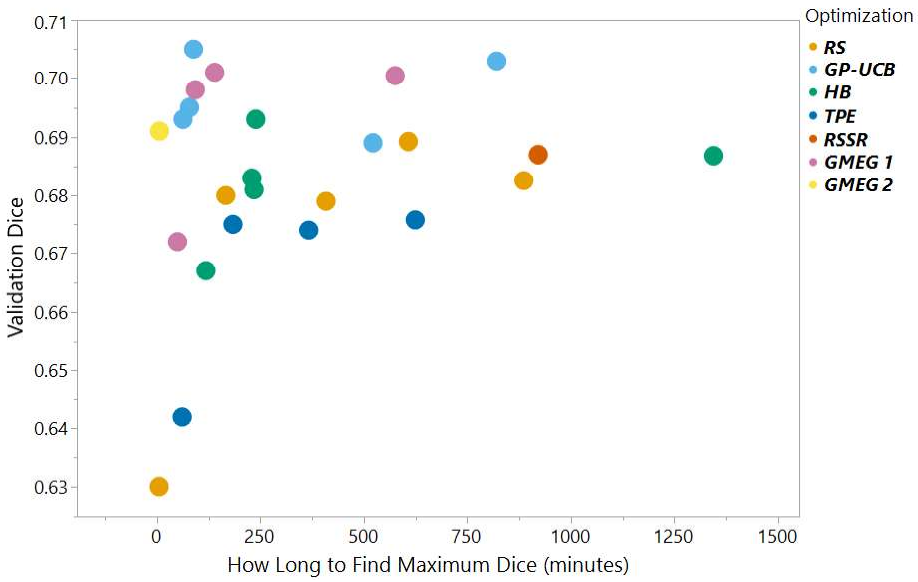



Figure 5.13: EoE Dataset Pareto Efficiency of Optimizations. This figure shows a scatterplot of the maximum validation Dice scores versus the time these scores occurred during each each optimization. The points in the upper-left were the most Pareto efficient. GP-UCB and GMEG 1 were both efficient on the EoE data.

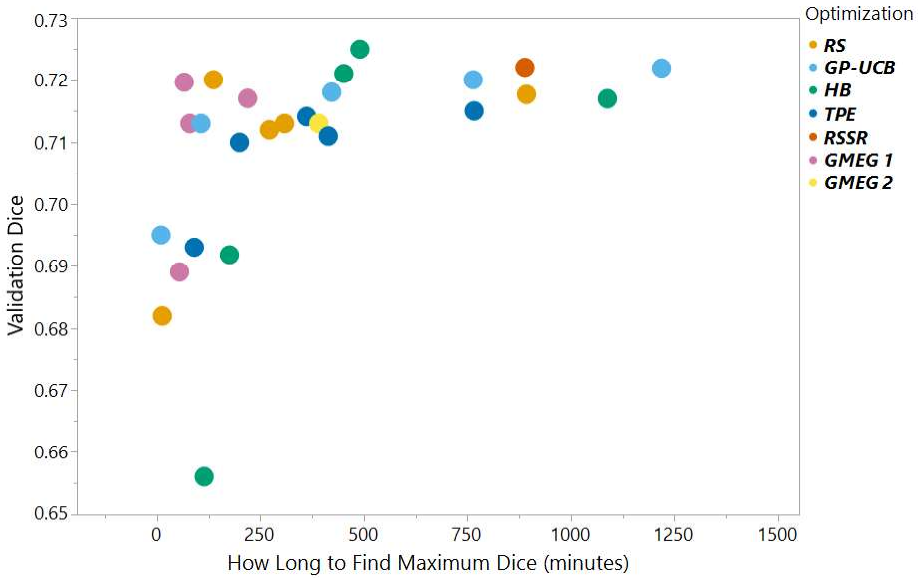



Figure 5.14: CV Nuclei Dataset Pareto Efficiency of Optimizations. This figure shows a scatterplot of the maximum validation Dice scores versus the time these scores occurred during each each optimization. The points in the upper-left were the most Pareto efficient. GMEG 1 appeared most efficient on the CV nuclei data.

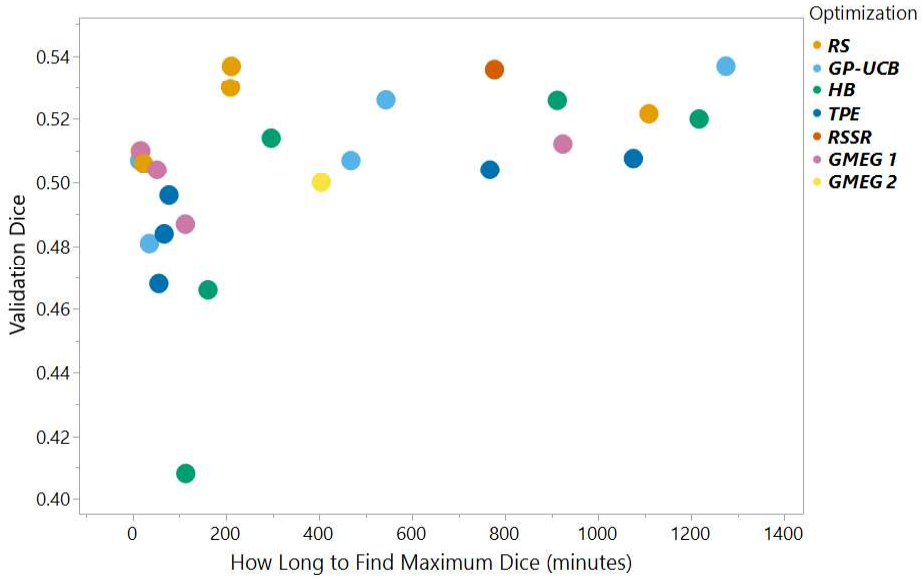



Figure 5.15: CV SMC Dataset Pareto Efficiency of Optimizations. The RS optimization appears efficient on this dataset, but this was likely due to random chance of testing a near-optimal setting as one of the early trials.

some GP-UCB runs that quickly found high validation Dice coefficients.

Figure 5.17 shows the Pareto efficiency of the optimization techniques on the graph reading heart rate detection dataset. For this dataset, GP-UCB was again the most efficient optimization in finding high validation accuracies early using a sequential approach.

Figure 5.18 shows the Pareto efficiency of the optimization techniques on the graph reading diastolic BP detection dataset. Both RS and GP-UCB were the most Pareto efficient on this dataset.

Figure 5.19 shows the Pareto efficiency of the optimization techniques on the graph reading systolic BP detection dataset. GP-UCB was the most Pareto efficient optimization on this dataset.

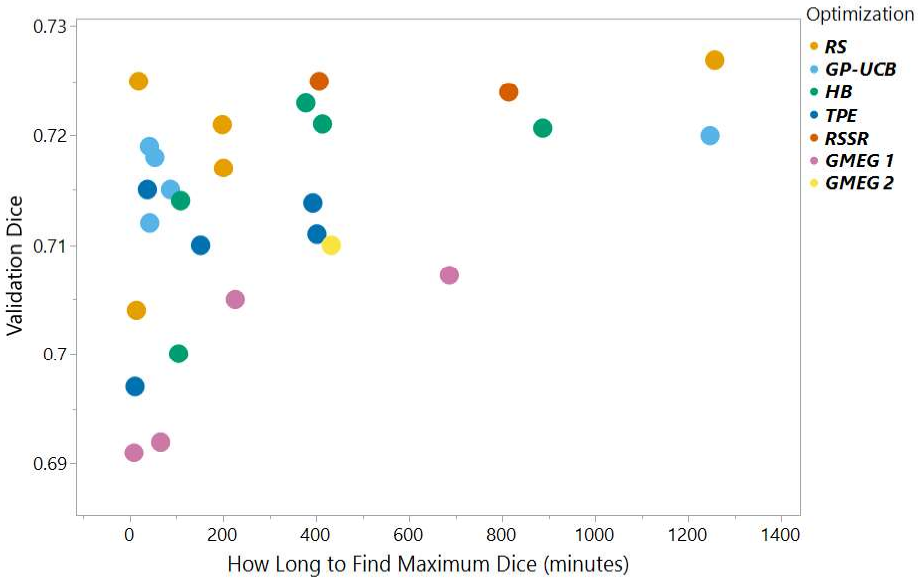



Figure 5.16: CV Macrophage Dataset Pareto Efficiency of Optimizations. The RS optimization appears efficient on this dataset, but this was likely due to random chance of testing a near-optimal setting as one of the early trials.

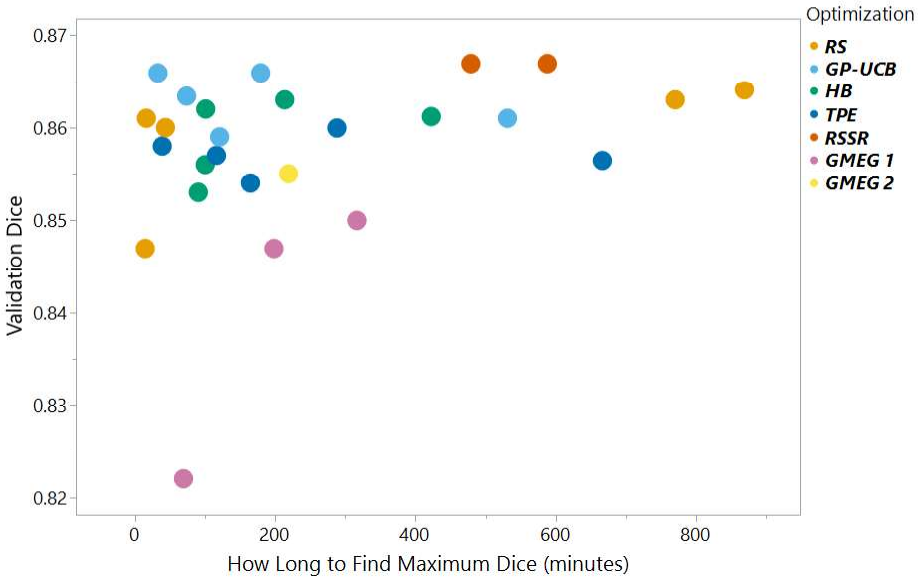



Figure 5.17: Vital Signs HR Dataset Pareto Efficiency of Optimizations. The GP-UCB appeared as the most efficient on the graph reading HR dataset.

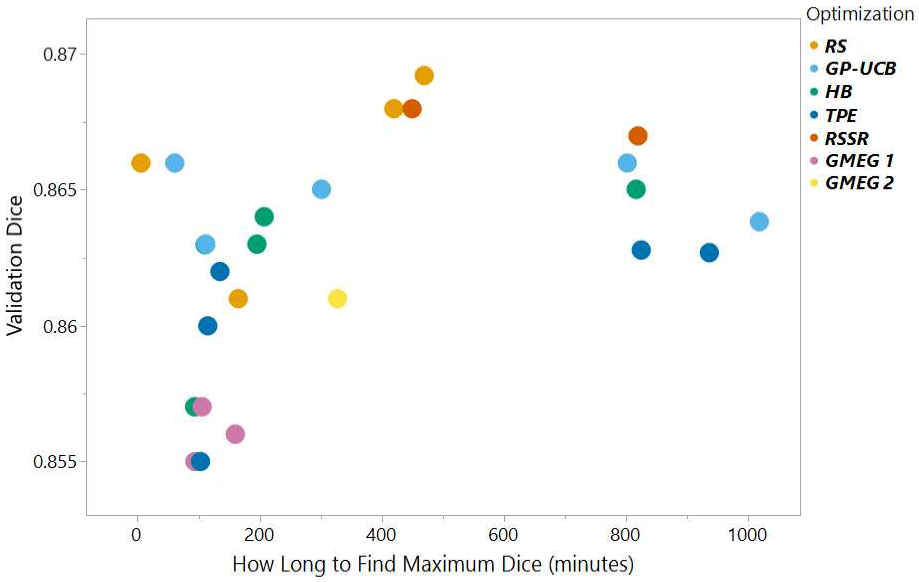



Figure 5.18: Vital Signs Diastolic BP Dataset Pareto Efficiency of Optimizations. The RS and GP-UCB optimizations appear as the most Pareto efficient on this dataset.

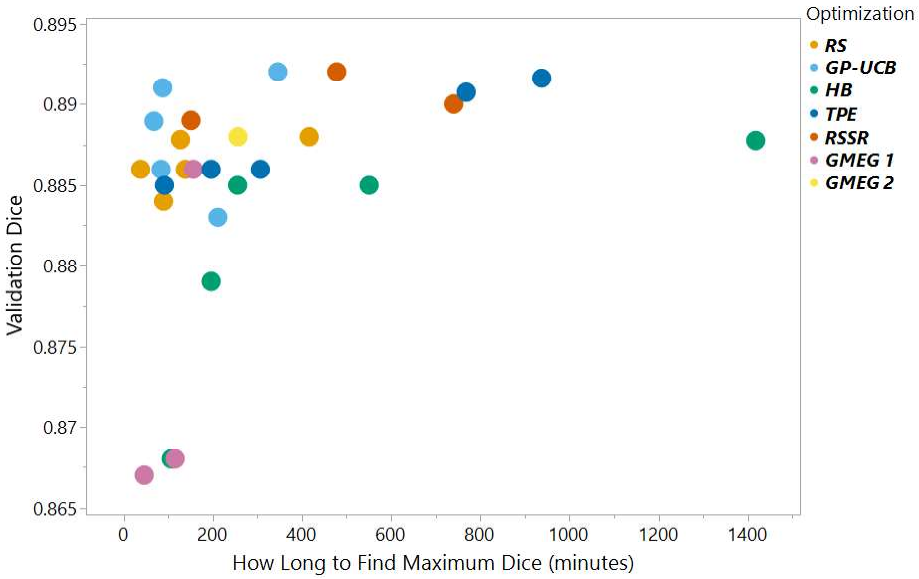



Figure 5.19: Vital Signs Systolic BP Dataset Pareto Efficiency of Optimizations. The GP-UCB optimization was the most Pareto efficient on this dataset.

# Chapter 6

# Conclusion and Future Work

In this dissertation, different hyperparameter optimization techniques were evaluated on three biomedical image segmentation projects. The evaluations were performed at five different timed-scenarios to learn how runtime or the number of executed trials affects model performance and the sufficiency of each technique. The results showed that the Bayesian optimization approach, GP-UCB, performed well in the scenarios where time was most limited. This makes sense because GP-UCB is a SMBO technique that starts exploiting and exploring immediately after the warm-up period. In the 2-hour setting, the GP-UCB performed better than the other techniques, while still in the warm-up period (15 trials) for the EoE and CV data. Perhaps, the way the software package *Keras Tuner* allocated the warm-up period was more robust than a random search of 15 trials. GP-UCB does fairly well across all time scenarios, but was outperformed at the 8 and 16-hour scenarios.

The RSSR approach, when applied, usually achieved the highest validation accuracies of all methods. RSSR was mainly designed to overcome the large skew between good and bad trial runs. The implementation only reduced settings once at the timed-scenario halfway point, but the same approach could be applied after every trial much like a traditional SMBO technique. In that case, it could be possible to remove a setting and then reinstate the setting later after more data is gathered. The cutoffs used to remove or retain variables allowed for some user-flexibility based on how aggressive they would like to be or for computation limitation purposes.

The GMEG approaches did not have as much success at finding the highest validation accuracy. This technique did show some promise as an efficient method for finding a decent model fit in a short amount of runtime. It was the ultimate approach for user-flexibility as there were many different customizations including the desired number of filters, how to model the variables, and how often to do exploration. Hoeever, this can also be seen as a downside that a optimization technique has so many of its own hyperparameters to adjust. The main purpose of this approach was to ensure there was adequate exploration since some of the Bayesian options did repetitive testing for many trials. Therefore, the $\epsilon$-greedy aspect should remain with this approach, but different $\epsilon$-greedy methods can be tested in the fture. The use of GMM as the surrogate could change to function that maps hyperparameters to

the validation accuracy response rather than treat the response as an independent variable. A similar GMM strategy is applied in TPE, but perhaps a regression tree approach, similar to SMAC, would perform better.

When researchers start an image segmentation project, it was often easy to re-use hyperparameters that excelled on previous projects. This was done for convenience, but researchers likely do not realize how big of an impact the hyperparameters have on model performance. The results in this dissertation showed that certain hyperparameters can lead to much higher validation Dice coefficients than others. Settings for batch normalization, batch size, and learning rate increased prediction performance on certain datasets. Therefore, it was critical to perform hyperparameter optimization on each specific image segmentation project to ensure that the models were tuned exclusively to each problem. In the Datasets chapter, each dataset had hyperparameter settings and results based on the prior knowledge-based user-established values. Table 6.1 shows those results before and after hyperparameter optimization. The best validation Dice coefficients found in all optimizations and the corresponding settings were shown for each dataset as the upper-bound of what optimization can do for performance. In all but one dataset, the performance improved and did greatly in most cases.

Table 6.1: Comparison of User-established Hyperparameters with Best Hyperparameter Optimization Settings

| Dataset | Optimization | Batch Norm. | No. of filters | Learning Rate | Dropout Rate | Batch Size | Val. Dice | Change |
|---|---|---|---|---|---|---|---|---|
| EoE | Before | TRUE | 32 | $2 \times 10^{-4}$ | 0.250 | 4 | 0.671 | +0.034 |
| | After | TRUE | 28 | $2 \times 10^{-4}$ | 0.279 | 4 | 0.705 | |
| CV Nuclei | Before | TRUE | 32 | $2 \times 10^{-4}$ | 0.250 | 4 | 0.683 | +0.042 |
| | After | TRUE | 24 | $2 \times 10^{-3}$ | 0.452 | 8 | 0.725 | |
| CV SMC | Before | TRUE | 32 | $2 \times 10^{-4}$ | 0.250 | 4 | 0.451 | +0.086 |
| | After | TRUE | 24 | $2 \times 10^{-4}$ | 0.266 | 2 | 0.537 | |
| CV Macrophage | Before | TRUE | 32 | $2 \times 10^{-4}$ | 0.250 | 4 | 0.710 | +0.017 |
| | After | TRUE | 24 | $2 \times 10^{-3}$ | 0.493 | 4 | 0.727 | |
| GR HR | Before | TRUE | 20 | $2 \times 10^{-3}$ | 0.250 | 2 | 0.835 | +0.032 |
| | After | TRUE | 24 | $2 \times 10^{-2}$ | 0.390 | 2 | 0.867 | |
| GR DBP | Before | TRUE | 20 | $2 \times 10^{-3}$ | 0.250 | 2 | 0.879 | -0.010 |
| | After | TRUE | 16 | $2 \times 10^{-2}$ | 0.422 | 2 | 0.869 | |
| GR SBP | Before | TRUE | 20 | $2 \times 10^{-3}$ | 0.250 | 2 | 0.889 | +0.003 |
| | After | TRUE | 24 | $2 \times 10^{-3}$ | 0.351 | 2 | 0.892 | |

When grid or random searches were performed, every trial can be considered an independent sample, so the optimal result could be found at any trial. The order of search runs was random, so it was appropriate to let the search execute the entire time scenario. Alternatively, SMBO techniques do control the order of runs, so even though there was an exploitation-exploitation balance, it should be unlikely that a much higher validation accuracy will surprisingly occur late in the SMBO process. Some SMBO techniques have a stopping criterion when it appears that optimality has been reached, but this seems like a very rare occurrence from experience. The Pareto efficiency results showed that an SMBO like GP-UCB typically found the best solution in about the first 5 to 6 hours, but the opti-

mizations were still running for 3 to 19 hours in the 8 to 24-hour scenarios. Even the best scores that were found at those times might only be minimally better than options found much sooner. Therefore, the results from this dissertation recommend that better stopping criterion be applied to SMBO techniques that monitor validation improvements and cease optimization if no improvement is found after $x$ trials.

There were other existing optimization techniques that were not evaluated in this study and more will be developed over time. An obvious future work recommendation is to expand this research and methodology to cover more techniques. There has been a growth of programming packages that handle hyperparameter optimization and feature numerous approaches. For example, the Neural Network Intelligence package [81] contained many different hyperparameter optimization and neural architecture search algorithms that could be evaluated, but the code was not implementable on the high-performance computing cluster at the time. There was also a lack of shared code from many reinforcement learning implementations of hyperparameter optimization or even value function approximation. When these two areas become more accessible we can evaluate a true reinforcement learning approach against the other methods or utilize a value function approximation to improve the GMEG technique.

# Bibliography


[1] Nabila Abraham and Naimul Mefraz Khan. "A novel focal tversky loss function with improved attention u-net for lesion segmentation". In: *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*. IEEE. 2019, pp. 683–687.

[2] William Adorno, Laura Shankman, and Donald Brown. "Combining Multiple Annotations to Count Cells in 3D Cardiovascular Immunofluorescent Images". In: *2021 IEEE EMBS International Conference on Biomedical and Health Informatics (BHI) (IEEE BHI 2021)*. Athens, Greece, July 2021.

[3] William Adorno et al. "Hand-drawn Symbol Recognition of Surgical Flowsheet Graphs with Deep Image Segmentation". In: *2020 IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*. IEEE. 2020, pp. 295–302.

[4] William Adorno III et al. "Advancing Eosinophilic Esophagitis Diagnosis and Phenotype Assessment with Deep Learning Computer Vision". In: *In Proceedings of the 14th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2021) - Volume 2: BIOIMAGING*. SciTePress. 2021, pp. 44–55.

[5] Md Zahangir Alom et al. "Recurrent residual U-Net for medical image segmentation". In: *Journal of Medical Imaging* 6.1 (2019), p. 014006.

[6] 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.

[7] Emelia J Benjamin et al. "Heart disease and stroke statistics—2019 update: a report from the American Heart Association". In: *Circulation* 139.10 (2019), e56–e528.

[8] James Bergstra, Daniel Yamins, and David Cox. "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures". In: *International conference on machine learning*. PMLR. 2013, pp. 115–123.

[9] 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.

[10] Paarth Bir and Valentina E Balas. "A Review on Medical Image Analysis with Convolutional Neural Networks". In: *2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*. IEEE. 2020, pp. 870–876.

[11] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[12] Stuart Carr, Edmond S Chan, and Wade Watson. "Eosinophilic esophagitis". In: *Allergy, Asthma & Clinical Immunology* 14.2 (2018), pp. 1–11.

[13] Liang-Chieh Chen et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[14] Xiaocong Chen, Lina Yao, and Yu Zhang. "Residual attention u-net for automated multi-class segmentation of covid-19 chest ct images". In: *arXiv preprint arXiv:2004.05645* (2020).

[15] 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.

[16] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[17] Nuno Ricardo Costa and João Alves Lourenço. "Exploring pareto frontiers in the response surface methodology". In: *Transactions on Engineering Technologies*. Springer, 2015, pp. 399–412.

[18] 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.

[19] Sripad Krishna Devalla et al. "DRUNET: a dilated-residual U-Net deep learning network to segment optic nerve head tissues in optical coherence tomography images". In: *Biomedical optics express* 9.7 (2018), pp. 3244–3265.

[20] Michal Drozdzal et al. "The importance of skip connections in biomedical image segmentation". In: *Deep learning and data labeling for medical applications*. Springer, 2016, pp. 179–187.

[21] Tom Eelbode et al. "Optimization for medical image segmentation: theory and practice when evaluating with Dice score or Jaccard index". In: *IEEE Transactions on Medical Imaging* 39.11 (2020), pp. 3679–3690.

[22] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural architecture search: A survey". In: *The Journal of Machine Learning Research* 20.1 (2019), pp. 1997–2017.

[23] Stefan Falkner, Aaron Klein, and Frank Hutter. "Combining hyperband and bayesian optimization". In: *NIPS 2017 Bayesian Optimization Workshop (Dec 2017)*. 2017.

[24] Navid Farahani, Anil V Parwani, and Liron Pantanowitz. "Whole slide imaging in pathology: advantages, limitations, and emerging perspectives". In: *Pathology and Laboratory Medicine International* 7 (2015), pp. 23–33.

[25] Peter I Frazier. "A tutorial on Bayesian optimization". In: *arXiv preprint arXiv:1807.02811* (2018).

[26] Glenn T Furuta et al. "Eosinophilic esophagitis in children and adults: A systematic review and consensus recommendations for diagnosis and treatment: Sponsored by the American Gastroenterological Association (AGA) Institute and North American Society of Pediatric Gastroenterology, Hepatology, and Nutrition". In: *Gastroenterology* 133.4 (2007), pp. 1342–1363.

[27] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[28] James H Harrison et al. "Introduction to Artificial Intelligence and Machine Learning for Pathology". In: *Archives of Pathology & Laboratory Medicine* (2021).

[29] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[30] Kaiming He et al. "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[31] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on* 14.8 (2012).

[32] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "Sequential model-based optimization for general algorithm configuration". In: *International conference on learning and intelligent optimization*. Springer. 2011, pp. 507–523.

[33] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[34] Fabian Isensee et al. "nnu-net: Self-adapting framework for u-net-based medical image segmentation". In: *arXiv preprint arXiv:1809.10486* (2018).

[35] Aamir Javaid et al. "Deep Learning Tissue Analysis Diagnoses and Predicts Treatment Response in Eosinophilic Esophagitis". In: *medRxiv* (2021).

[36] Hadi S Jomaa, Josif Grabocka, and Lars Schmidt-Thieme. "Hyp-rl: Hyperparameter optimization by reinforcement learning". In: *arXiv preprint arXiv:1906.11527* (2019).

[37] Baris Kayalibay, Grady Jensen, and Patrick van der Smagt. "CNN-based segmentation of medical imaging data". In: *arXiv preprint arXiv:1701.03056* (2017).

[38] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: `http://arxiv.org/abs/1412.6980`.

[39] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[41] Fahad Lateef and Yassine Ruichek. "Survey on semantic segmentation using deep learning techniques". In: *Neurocomputing* 338 (2019), pp. 321–348.

[42] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[43] 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.

[44] Xiang Li et al. "Understanding the disharmony between dropout and batch normalization by variance shift". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2682–2690.

[45] Sheng Lian et al. "A global and local enhanced residual u-net for accurate retinal vessel segmentation". In: *IEEE/ACM transactions on computational biology and bioinformatics* (2019).

[46] Peter Libby, Paul M Ridker, and Attilio Maseri. "Inflammation and atherosclerosis". In: *Circulation* 105.9 (2002), pp. 1135–1143.

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

[48] Geert Litjens et al. "A survey on deep learning in medical image analysis". In: *Medical image analysis* 42 (2017), pp. 60–88.

[49] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

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

[51] Patrick E McKnight and Julius Najab. "Mann-Whitney U Test". In: *The Corsini encyclopedia of psychology* (2010), pp. 1–1.

[52] 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.

[53] Douglas C Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

[54] Sandy Napel et al. "CT angiography with spiral CT and maximum intensity projection." In: *Radiology* 185.2 (1992), pp. 607–610.

[55] Zhen-Liang Ni et al. "Raunet: Residual attention u-net for semantic segmentation of cataract surgical instruments". In: *International Conference on Neural Information Processing*. Springer. 2019, pp. 139–149.

[56] Tom O'Malley et al. *Keras Tuner*. `https://github.com/keras-team/keras-tuner`. 2019.

[57] Ozan Oktay et al. "Attention u-net: Learning where to look for the pancreas". In: *arXiv preprint arXiv:1804.03999* (2018).

[58] Gerard Pasterkamp, Hester M Den Ruijter, and Peter Libby. "Temporal shifts in clinical presentation and underlying mechanisms of atherosclerotic disease". In: *Nature Reviews Cardiology* 14.1 (2017), pp. 21–29.

[59] Jaccard Paul. "The distribution of the flora in the alpine zone. 1". In: *New phytologist* 11.2 (1912), pp. 37–50.

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

[61] Md Atiqur Rahman and Yang Wang. "Optimizing intersection-over-union in deep neural networks for image segmentation". In: *International symposium on visual computing*. Springer. 2016, pp. 234–244.

[62] Victoria Rho et al. "Digitization of perioperative surgical flowsheets". In: *2020 Systems and Information Engineering Design Symposium (SIEDS)*. IEEE. 2020, pp. 1–6.

[63] 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.

[64] Thomas M Runge et al. "Causes and outcomes of esophageal perforation in eosinophilic esophagitis". In: *Journal of Clinical Gastroenterology* 51.9 (2017), p. 805.

[65] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. "Tversky loss function for image segmentation using 3D fully convolutional deep networks". In: *International workshop on machine learning in medical imaging*. Springer. 2017, pp. 379–387.

[66] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[67] David Silver. *Lectures on Reinforcement Learning*. URL: `https://www.davidsilver.uk/teaching/`. 2015.

[68] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[69] Leslie N Smith. "Cyclical learning rates for training neural networks". In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2017, pp. 464–472.

[70] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *arXiv preprint arXiv:1206.2944* (2012).

[71] Th A Sorensen. "A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons". In: *Biol. Skar.* 5 (1948), pp. 1–34.

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

[73] 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.

[74] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[75] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[76] Towaki Takikawa et al. "Gated-scnn: Gated shape cnns for semantic segmentation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 5229–5238.

[77] Sergios Theodoridis. *Machine learning: a Bayesian and optimization perspective*. Academic press, 2015.

[78] Amos Tversky. "Features of similarity." In: *Psychological review* 84.4 (1977), p. 327.

[79] Richard Weber. "On the Gittins index for multiarmed bandits". In: *The Annals of Applied Probability* (1992), pp. 1024–1033.

[80] Jia Wu, SenPeng Chen, and XiYuan Liu. "Efficient hyperparameter optimization through model-based reinforcement learning". In: *Neurocomputing* 409 (2020), pp. 381–393.

[81] Quanlu Zhang et al. "Retiarii: A Deep Learning Exploratory-Training Framework". In: *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 2020, pp. 919–936.

[82] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. "Road extraction by deep residual u-net". In: *IEEE Geoscience and Remote Sensing Letters* 15.5 (2018), pp. 749–753.