Estimating Predictive Uncertainty in Gastrointestinal Image Segmentation

Felicia Jacobsen

Thesis submitted for the degree of
Master in Biological and Medical Physics
60 credits

Department of Physics
The Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2022

Estimating Predictive Uncertainty in Gastrointestinal Image Segmentation

Felicia Jacobsen

Estimating Predictive Uncertainty in Gastrointestinal Image Segmentation

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Deep learning models are known to achieve state-of-the-art performance in numerous applications. Applying these models to the medical field can relieve demanding workloads and decrease the number of observational oversights in diagnostics. Despite this, deep learning models are considered to exhibit a "black box" nature due to their complex structure and lack of transparency, interpretability, and explainability to how they arrived at a decision. These attributes are crucial to establish trust and reliability between the users and model.

We investigated the use of computer-aided detection of polyps in the gastrointestinal tract using segmentation models. To increase explainability behind the model prediction, we adopted two existing uncertainty estimation methods, Monte Carlo (MC) dropout and deep ensembles. We further explored these using two state-of-the-art deep learning architectures, U-Net and ResUNet++, trained with two different loss metrics, binary cross-entropy and dice similarity coefficient (DSC). The uncertainty estimates were visualized as heatmaps, showing spatial uncertainties for the predicted segmentation mask. Our results show that deep ensembles provide informative uncertainty representations that connect large uncertainties to misclassified pixels. Additionally, we established a correlation between the large uncertainty estimates and the corresponding pixels that are likely to be misclassified. MC dropout was insufficient at providing such information to its uncertainty representations.

Out of all the combinations tested, our results show that using the U-Net based deep ensemble trained with the DSC metric gave the overall highest score in terms of the mean DSC during test time. Deep ensembles were also able to increase this test score when increasing the ensemble size, giving a DSC score equal to 0.8172 using an ensemble size of 16. We conclude that uncertainty estimation, using deep ensembles, can improve the understanding of deep learning models that can aid users to make informed decisions on whether to trust a model's predictions.

Acknowledgments

I would like to express enormous gratitude towards my supervisors Pål Halvorsen, Michael Riegler and Steven Hicks. Thank you for giving me expert supervision, guidance, and newfound academic confidence for the past year. I would especially thank Steven for being a huge inspiration and always bringing positive energy to our weekly meetings.

I would also like to thank my friends Simen, Jon and Frida for helping me with proofreading, and for always giving me support and encouragement.

To everyone mentioned here, thank you for believing in me.

Contents

List of Figures

List of Tables

Acronyms

AI artificial intelligence. 2, 15, 18, 44, 45, 49

ASPP atrous spatial pyramid pooling. 37, 38, 61–63

BCE binary cross-entropy. 6, 39, 40, 67–70, 72, 75, 76, 79, 80, 82–91, 93–95, 98–100, 103, 105–109

CAD computer-aided detection. 2, 9, 12–15, 53, 106, 109

CNN convolutional neural network. 7, 20, 27, 28, 30, 32–34, 36–39, 44, 46–48, 53, 55, 56, 60, 63, 64, 67, 70, 78, 85, 89–91, 103, 104, 106–109

CPU central processing unit. 55

CRC colorectal cancer. 1, 9, 11, 12, 53

DL deep learning. 1–4, 6, 7, 9, 13–15, 21, 22, 26, 31, 35, 39, 44–50, 53–57, 77–79, 104–106, 108, 109

DSC dice similarity coefficient. i, 6, 40–44, 52, 63, 67, 68, 70–72, 75–77, 79–96, 98–100, 102, 103, 105–108

ECG electrocardiogram. 47, 48

FCN fully convolutional neural network. 39

FN false negative. 14, 40, 92, 93, 96–99

FP false positive. 14, 40, 92–94, 96–99, 103

GD gradient descent. 21–24, 46, 58

GI gastrointestinal. 10, 12, 16–18

GPU graphics processing unit. 1, 55, 56, 64, 66, 70, 78, 101

IoU intersection over union. 41–44

MC Monte Carlo. i, 6, 7, 50, 51, 59, 70, 71, 73, 74, 76, 78–85, 87–108, 123

NN neural network. 18–22, 24–28, 31, 32, 44, 46, 48–53, 55, 56, 59, 60, 64, 72, 75

RAM random access memory. 56, 60

SE squeeze-and-excitation. 36, 37, 61–63, 71

SGD stochastic gradient descent. 23, 69

TN true negative. 93, 96, 98, 99

TP true positive. 40, 41, 93, 94, 96, 97, 99, 102, 103

VV Vestre Viken Health Trust. 16

XAI explainable artificial intelligence. 2, 3, 6, 7, 15, 45–47, 53, 105, 108

Chapter 1

Introduction

1.1 Motivation

Decision-making is imperative in modern medicine. Medical experts must not only rely on their knowledge, but also on their experience and clinical judgment to determine the best course of action that can benefit the patients. Despite this, human perception largely influences the decision-making process. Recognizing visual objects through human perception is not only something one can learn but occurs outside of one's own consciousness. Decision-making by deep learning (DL) models [76] is in contrast quite different, and their ability to recognize high-level features is based on a whole different aspect of factors. With the increasing number of large medical datasets [13, 18, 63, 68], the availability of the graphics processing units (GPUs), combined with the state-of-the-art DL architectures [47, 126, 133], DL models have gained large popularity in the medical field due to their ability to recognize medical data [44, 59], proving their clinical relevance.

As of today, colorectal cancer (CRC) is the third most prevalent cancer type worldwide and is the second most common cancer-related death [130]. CRC typically starts with a small benign growth inside the lining of the colon or rectum, referred to as polyps, which over time can become malignant. Polyps are often small and show few signs of or no symptoms. In many cases, patients with CRC are known to experience no symptoms in the earlier cancer stages [83]. Common symptoms like blood in stool, abdominal pain, diarrhea or change in bowel movements do not reveal themselves until later stages. Such medical scenarios can potentially delay treatment, meaning that the cancer can progress to lethal stages before the symptoms have been revealed. Early detection, identification, and removal of polyps at an early stage are thus highly beneficial for preventing mortality against CRC [124].

Today, colonoscopy is the gold standard for such screenings [124]. However, studies show that this technique can leave up to approximately 28% of polyps undetected [86, 95]. Computer-aided detection (CAD) is designed to decrease observational oversights and has been integrated into clinical workflows of diagnostic mammography more than two decades ago [111]. CAD systems can be utilized by medical experts for automatic polyp segmentation in colonoscopy screening. Segmentation is a technique used to highlight polyps from images or videos, which can act as a second tool in colonoscopy screenings. This second observer can potentially reduce the number of undetected polyps and thus provide improved prognosis to patients [10]. Besides the medical advantages, CAD can also reduce physician workloads, increase efficiency at clinics while being cost-effective, verify and assess the quality of earlier examinations, and train inexperienced physicians and medical students. Moreover, CAD is cost-efficient because the use of a computer rather than a second human observer has the advantage of reducing the demand for trained physicians.

With the increasing availability of large colonoscopy datasets [4, 5, 106, 125], several different DL approaches have shown to be highly effective and capable in detecting polyps. Segmentation models have been shown to be able to achieve impressive results in pixel-level detection of polyps [65, 66]. Other approaches include the use of bounding boxes to detect polyps [91, 109, 148]. These DL-based approaches were in all cases able to obtain a state-of-the-art performance. Despite this, these DL models have been considered "black boxes" due to their complex structure and their lack of explainability. They do not provide additional information on *why* they arrived at a specific decision [26, 54]. Because they lack the ability to explain their reasoning, they do not reveal their strengths or weaknesses in the decision-making process.

Explainability can assist developers in preventing future disasters provided by predictive models because strengthening the understanding of their predictions may suggest ways of improving them. Even though explainability is needed in many fields, it is an especially crucial property in high-risk predictions where the cost of making a wrong decision can determine the future of human life and health. In medicine, doctors are held accountable for their decisions by being required to explain and justify how they arrived at a specific diagnosis. Just as humans are held accountable for this reason, predictive models must also provide such explanations to obtain this accountability. CAD systems that miss these important aspects raise questions on how they can be accepted in the clinics.

Explainable artificial intelligence (XAI) aims at answering these questions. XAI methods are to bestow the understanding of why artificial intelligence (AI) models arrive at a particular decision by developing methods

that provide interpretability, transparency and explainability to these decisions. The benefits of XAI methods can help to integrate humans into the decision-making process of DL models, mend legal issues, foster trust, verify the diagnosis and provide complimentary information to help and assist physicians in treatment planning.

With the DL models' "black box" property, modeling their predictive uncertainty is hence a non-trivial task. Uncertainty provides explainability by adding additional information to the corresponding model decision and can be used by medical experts to gain insights into its prediction. Uncertainty estimates can be used to critically assess if the model gives overconfident predictions. Many researchers have attempted to quantify the uncertainty of DL models [100]. The result is a variety of approaches [37]. Two of these approaches propose methods of extracting the uncertainty using an ensemble of DL models [35, 73]. Additionally, ensembling resulted in a performance increase in terms of generalizability. This thesis contributes to the field of XAI by extending explainability to medical image segmentation by adapting the two existing state-of-the-art uncertainty estimation methods in a segmentation use-case. We focus on the use of polyp segmentation models to represent the uncertainty estimates as spatially distributed uncertainty representations to segmentation predictions.

1.2 Problem Statement

Despite the recent advances in achieving state-of-the-art performance by DL-models in detecting polyps by colonoscopy imaging, there still lacks research on explainability in these models. The work in this thesis is motivated by the lack of explainability in DL, and we attempt to contribute to XAI by providing predictive uncertainty estimates by answering the following research question:

RQ1 How can the predictive uncertainty estimates improve the understanding of the model prediction?

We aim to solve this research question with the following three research objectives:

Objective 1 Explore how uncertainty estimates connect the model prediction with the input features.

Objective 2 Explore how the use of different loss functions affect the model predictions and the corresponding uncertainty.

Objective 3 Use two different architectures to examine if the uncertainty estimation performance is affected by the model architecture.

These research objectives act as sub-tasks which help us to explore and extract the full potential from the uncertainty estimation methods adapted in this thesis.

1.3 Scope and Limitations

The scope of this thesis is limited to answering the research question defined in Section 1.2 by estimating the predictive uncertainties in automated polyp segmentation. The scope is divided into three main objectives. The first objective is to explore how uncertainty estimates connect the model prediction with the input features. The second is to explore the use of different loss functions to observe how and if the uncertainty estimates are affected by the different loss functions. The third and final objective is to examine how the performance of the uncertainty estimation method is affected by using different DL-based model architectures.

We consider several limitations for the scope of this thesis. The first is to limit the number of model architectures that are explored with. Some requirements that were set a priori, was that the trained models must be able to reach the state-of-the-art performance to prove their potential in clinical workflows. Showing what these models are capable of will support the motivations stated in Section 1.1. We limit the amount of hyperparameter tuning knowing that extensive tuning techniques are known to be both memory- and time-consuming. Even though extensive tuning methods are sometimes needed to achieve state-of-the-art performance, we argue that these types of achievements are beyond the scope of this thesis.

Memory constraints were another limiting factor in this thesis. This resulted in using smaller images as input to the models and utilize both half and single-precision during training, as explained in Section 3.1.4 and Section 3.3.4. The resources used were also shared with other researchers, meaning that the experiments could not be performed at any time. This was also a time-consuming factor because the shared resources were not always available.

Due to the lack of publicly available annotated datasets, the experiments conducted in this thesis were originally limited to three datasets and their corresponding sizes and quality. And because of time-constraints, we limited our experiments to one of the three datasets.

The last limitation is the time constraints, which limits the exploration of model architectures and loss functions, the number of training iterations, the number of trained models in an ensemble, hyperparameter tuning, and the number of uncertainty-extracting methods used. In the case of

less time restrictions, we explain potential future work in Chapter 6.

1.4 Research Methods

A research methodology is a set of specific procedures used to systematically and theoretically present, process and analyze the information provided in a research study. We have adapted Association for Computing Machinery's (ACM) research methodology [20] for the research presented in this thesis. This research methodology was presented in a specific report provided by a task force. The purpose of this report was to provide a detailed description of the discipline of computing and to propose a new teaching paradigm also known as the Computing Curricula model for computing disciplines.

This methodology argues that the discipline of computing should be split into three fundamental processes: theory, abstraction, and design. The way the research in this thesis is presented is consistent with each of these three processes of the computing discipline.

1.5 Ethical Considerations

Ethical considerations are a set of principles that should be followed in human research. These considerations include maintaining voluntary precipitation, informed consent, anonymity, confidentiality, integrity, and transparency, maintaining the minimum potential for harm and completely avoiding research misconduct.

Two of these were the key ethical considerations and the most relevant for the research conducted in this thesis:

Anonymity We do not know the identities of the participants from whom the data is collected. Personally identifiable data or metadata were neither collected nor presented in this research.

Research misconduct No data presented in this thesis were either falsified or made up. Manipulation of data analyses and misrepresentation of the results were in no way conducted in this thesis.

By following these key ethical considerations for human research, one can maintain scientific integrity and protect the rights of research participants.

1.6 Main Contributions

This thesis contributes to the field of XAI by testing and extending existing state-of-the-art uncertainty estimation methods to a segmentation use-case. We pose three research objectives given in Section 1.2, which aid us in solving **RQ1**. We summarize the main contributions behind this thesis by sequentially repeating and explaining how these were met. Lastly, we revisit and answer **RQ1**.

Objective 1: *Explore how uncertainty estimates connect the model prediction with the input features.*

This objective is supported by our exploration of two different uncertainty estimation methods, Monte Carlo (MC) dropout and deep ensembles. We extended these approaches to a segmentation use-case by testing and comparing these methods. By doing so, we were able to establish connections between the information given in the uncertainty estimates and their corresponding predictions given by the DL model. More specifically, we observed that the areas of high uncertainties had corresponding predictions that were more likely to be incorrectly classified. Only *one* of these methods was able to establish such connections.

Objective 2: *Explore how the use of different loss functions affect the model predictions and the corresponding uncertainty.*

Testing uncertainty estimation techniques on models trained with different loss functions allowed us to reach this objective. By doing so, we were able to explore uncertainty estimation techniques based on different model training characteristics. We observed that for all cases, using dice similarity coefficient (DSC) loss during training outperformed the corresponding models based on binary cross-entropy (BCE) loss, even though BCE is categorized as a proper scoring function. The purpose of this objective was to explore the uncertainty estimation techniques in different scenarios to investigate how to fully utilize these.

Objective 3: *Use two different architectures to examine if the uncertainty estimation performance is affected by the model architecture.*

Like the second objective, the purpose of this objective was to fully explore these uncertainty estimation techniques to fully utilize the adapted uncertainty estimation techniques in a segmentation use-case. We adapt two different state-of-the-art model architectures, U-Net and ResUNet++, to see how these affect the results of the uncertainty estimation methods. We found that the former architecture type outperformed the latter in all cases. Without the use of extensive and resource-demanding hyperpa-

rameter tuning techniques, we know that this objective leaves a topic for further investigation.

Reaching these research objectives helps us to explore and extend these existing uncertainty estimation techniques, which may motivate further development of these methods in the future. More importantly, meeting these objectives aids us in answering the following research question:

RQ1: *How can the predictive uncertainty estimates improve the understanding of the model prediction?*

We answer this question by testing each combination of two uncertainty estimation techniques, trained with different loss functions, and based on two different convolutional neural network (CNN) architectures. The resulting uncertainty representations were visualized as heatmaps, which shows spatially distributed predictive uncertainty estimates by a DL model. We were able to show that the uncertainty representations given by the deep ensemble method give important correlations between the predicted segmentation masks and the corresponding uncertainty representations. We know that these correlations are also consistent with the previous general understanding of uncertainty. Thus, to clearly answer our research question, we argue that the uncertainty representations given by the deep ensemble method are indeed able to improve the understanding of the model prediction.

The uncertainty representations given by the deep ensemble method show that the areas of large uncertainty estimates correspond to pixels in the predicted mask that are more likely to be misclassified. The uncertainty estimates given by MC dropout was unsuccessful at providing such correlations.

Meeting these research objectives and concluding our results by answering the research question lay the foundation for the main contributions given by this thesis. We claim that these contributions add value to the field of XAI which is consistent with our motivations stated in Section 1.1. Considering our main findings, we know that these explanations can help to improve the understanding of the model prediction that can be used to make more informed decisions on whether to trust a model's predictions. We hope that this work leaves motivation for further development for explainability- and uncertainty estimation methods.

1.7 Thesis Outline

This thesis is divided into six chapters. The following summary gives a brief introduction to the next five chapters.

Background: Chapter 2 presents the background, which provides a description of the theoretical framework that supports the methods used in this thesis.

Methods: Chapter 3 gives detailed descriptions of the experimental research process behind this thesis. This chapter supports the results in this thesis by acting as a roadmap to the reached research conclusions and gives explanations to why a specific approach was chosen, and how it led to the answers to the research questions. Discussion of obstacles and their solutions, descriptions of how results were collected, and how it was analyzed are also provided in this chapter.

Results: The main results from the experimental methods explained in Chapter 3 are given in Chapter 4. The results are provided with a description of the main observations, and the additional details and results can be found in the appendices.

Discussion: In Chapter 5, the results from the previous chapter are put into context, and conclusions are drawn. The results were interpreted considering the known information, and connections between research questions and the results are also provided. Finally, the results were also discussed in a broader context.

Conclusion: Chapter 6 provides the reader with the summarized findings. Finally, we discuss how our findings have laid a foundation for potential future work.

Chapter 2

Background

The necessary theoretical background behind the methods used in this thesis is presented in this chapter. Section 2.1 provides a medical perspective, which contains information about the digestive system, CRC, and screening methods of polyps. Section 2.2 gives an overview on the potential of CAD technologies today. Section 2.3 covers information about the datasets used in this thesis and descriptions on how they were collected. Sections 2.4, 2.5, 2.6 and 2.7 contain detailed descriptions of the algorithms used in this thesis. The theory on algorithms describes how they can make representations of and recognize the data. Section 2.8 and 2.9 contain details about the concepts of ethics and explainability in DL. The latter section explains why explainability is important, its relevance in the medical field, and summarizes the most relevant explainability methods of today. Section 2.10, introduces the relevant research and methods that were adapted and tested in this thesis. Finally, Section 2.11 provides a summary of this chapter.

2.1 Medical Background

Polyps are a central topic in this thesis. Therefore, this section is dedicated to the medical background behind CRC. An example image taken from the test dataset of the MediaEval challenge from 2021 can be viewed in Figure 2.1. Here, we see several polyps spread across a small area inside the colon. More information on this challenge can be found in the published research paper in Appendix B [61]. The medical background on polyps includes their hallmarks, treatment, occurrence, and clinical screening methods.

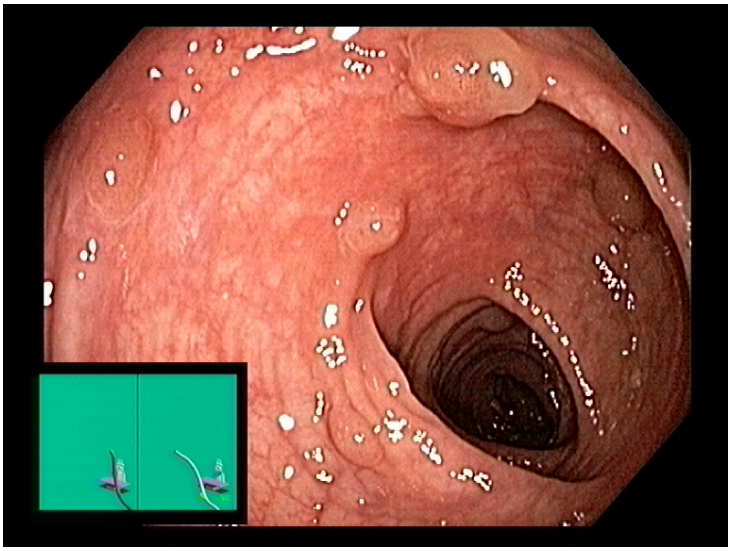

Figure 2.1: An example image taken from the test dataset of the MediaEval 2021 challenge (see Paper B.1 for further details).

2.1.1 The Gastrointestinal Tract

The gastrointestinal (GI) tract spans from the mouth to the anus and includes all organs of the digestive system: mouth, pharynx, esophagus, stomach, small intestine, large intestine, and rectum. The GI tract is responsible for four main processes: digestion, secretion, absorption, and motility [87].

Digestion starts at the mouth, where food is broken down into smaller compounds by the teeth and gets mixed with saliva [138, p. 372]. The food passes the pharynx, which is the organ that connects the mouth with the esophagus [138, p. 373]. The esophagus is a muscular tube that moves food from the pharynx to the stomach using muscle movements called peristalsis [138, p. 372]. The stomach is a J-shaped muscular sac that extends from the esophagus to the small intestine [138, p. 376]. It stores the food and breaks it down further. Peristaltic movements induce the release of a hormone called gastrin, which triggers the secretion of highly acidic digestive juices [138, p. 34]. These juices contain pepsin and acid, the former is an enzyme responsible for breaking down proteins of the food in the stomach, and the latter are responsible for killing bacteria and activating pepsin [138, p. 377].

The small intestine is a roughly 6 meter long tube that spans from the stomach to the large intestine [138, p. 378]. The lining of the small intestine has finger-like projections called villi, and their function is to increase the surface area of the small intestine [138, p. 51]. The villi are covered in microvilli which further increases the surface area for food absorption.

Nutrients from food pass into the bloodstream and lymph via the small intestine walls [138, p. 378]. Water from the digested food is absorbed in the large intestine. The large intestine is made up of an approximately 1.5-meter long muscular tube called the colon [138, p. 285]. Its function is to absorb water, minerals, vitamins and to eliminate indigestible material. No digestion takes place in the colon, only secretion of mucus to lubricate the passage of fecal material [138, p. 285]. Fecal material is formed from fiber and other undigested materials. The end of the colon connects to the rectum, which receives waste and pushes it out of the body through the anus.

2.1.2 Colorectal Cancer

Cancer cells divide uncontrollably in the cell cycle. In these cells, the DNA that is responsible for transcribing proteins responsible for cell cycle regulation have become damaged. These damages correspond to mutations, and if not repaired, the division mechanisms allow for cells to divide uncontrollably [138, p. 327]. Some cells with non-regulated cell cycles are categorized as cancerous, specifically the ones with uncontrollable cell division. Once these cells have become cancerous, their functioning cannot return to normal. The immune system will often destroy them, but sometimes it is not able to recognize them as cancerous, especially with age [138, p. 61].

Cancerous cells can form abnormal growths of tissue, called tumors. Tumors are either cancerous (malignant) or non-cancerous (benign). Benign tumors are known to not grow as rapidly as malignant ones. They do not spread to nearby tissues, nor do they grow back once removed from the body [138, p. 61]. In malignant tumors, cancer cells can break away from the tumor and travel through nearby blood or lymph vessels to distant regions of the body, allowing them to form new tumors at different sites [138, p. 61]. This is different from when cancer cells spread to nearby organs because the lymphatic system is a network that covers the entire body. Hence, cancer cells with access to the lymphatic system are referred to as metastasis or metastatic cancer and are regarded as an advanced stage of cancer. Depending on where the cancer spreads, metastatic lesions can become the deadliest form of secondary tumors. Metastatic cancer is the most fatal cancer stage and estimated to be responsible for about 90% of all cancer deaths [122, p 215].

A polyp is a benign cancer growth that specifically grows inside the colon. Specific types of benign tumors are known to later become malignant [122, p 211]. Hence, this also yields for polyps, and they are thus surgically removed once detected. Also, 95% of CRC cases are believed to arise from

polyps [88, p 678]. A typical medical scenario is that CRC occurs with the development of cancerous cells in a polyp. The polyp starts growing in the inner lining of the colon, the mucosa. Here, it can grow outward to the outer layers of the colon, which can connect the cancer cells to the blood or lymph vessels. When the cancer has advanced to these vessels, it is able to metastasize [122, p 222]. Early detection of polyps is therefore crucial for prognosis and survival [124].

Polyps can grow in several different areas along the GI tract, and they vary in size and shape, and look like protruding round lumps. Some grow on a stalk, and others protrude directly from the mucosa. Specific polyp types and characteristics increase their possibility of becoming cancerous. These characteristics include the size of the polyp or if there are three or more polyps located in the same area. Polyps are divided into four main groups: mucosal adenomatous polyps, mucosal serrated polyps, mucosal nonneoplastic polyps, and submucosal lesions. Of all polyps removed, over 70% are adenomatous, and the remainder are serrated polyps. They are removed because 95% of all CRC cases are believed to arise from these specific polyp types [88, p 678].

2.1.3 Screening of Colorectal Cancer

Due to the potential negative consequences of having polyps in the colon, screening and removal before they develop into CRC or having metastasized are crucial. Colonoscopy is currently the gold standard for detecting and removing polyps [88, p 679]. It is an invasive procedure that requires a long flexible tube with a camera and light source attached to the end. This tool is called a colonoscope. It is inserted through the mouth or anus and can be used to screen the entire colon and rectum. The polyp can be removed during the colonoscopy, using different instruments, such as a wire loop, which are inserted through the tube. Other instruments used to remove polyps are biopsy forceps or electric current. Patients in the US who do not have an increased risk of CRC are recommended to undergo this procedure at least every 10 years, and for patients that have an earlier incidence of polyps, or an increased hereditary risk of CRC, it is recommended to have a colonoscopy every 3 to 5 years [9]. This usually differs between different countries, often depending on infrastructure and availability of personnel. Colonoscopy devices and experts that can perform the procedure are sparse and often the bottle neck for national screening programs. With CAD, the doctors are supported in their task via automatic analysis to be more efficient, which is considered as a crucial factor to enable population wide screening.

2.2 Computer Aided Detection

CAD technologies have the potential to automate and relieve clinical workflows. To do so, the system must be able to detect diseases and reduce the number of false negatives. With this considered, this technology can relieve workflows by replacing a second human observer [10]. Thus, it can be able to decrease the demand for more physicians working with the same task.

2.2.1 History of Computer Aided Detection

Exploring with the use of computers to solve problems in medicine started in the 1950s. From around this time, two studies [78, 140] tested the use of patient symptom- and laboratory test results data for computers to generate a diagnostic outcome. In the first study, Ledley et al. [78] analyzed the reasoning process behind traditional medical diagnosis for this analytical process to be inherited in computers. In the second study, Weiss et al. [140] used the results from their disease classification to predict recommended treatment strategies, and this approach was later successfully integrated into a consultant program, which assisted diagnosis and long-term treatment of glaucoma.

In the 1960s, there were multiple studies on CAD in medical imaging. At the same time, it was expected that computers were able to replace human experts in detecting medical abnormalities. This is because it was assumed that computers outperformed humans at these tasks [25].

Today, CAD is considered as an assisting tool in the diagnostic process which acts as a second observer where human experts are included [142]. Most approaches in CAD of modern time are based on DL-models. Many of them are described in the following section.

2.2.2 Recent Developments of Computer Assisted Polyp Detection

Computer assisted polyp detection has been an active research topic over the last two decades [135]. Even to this day, there are many research papers on this topic that get published each year. The work listed below are only a few examples from the enormous amount of research studies on this topic. Even with these few examples, the listed work is still representative of this active research field. For example, Polyp-Alert [139] is a real-time visual feedback tool that detects polyps by recognizing their edges. This software is designed to be used during routine colonoscopy screenings. Other recent developments in polyp detection consist of the

recent progress in detecting bounding boxes containing polyps [91, 109, 148], as mentioned in Section 1.1. Other developments contain the classification of polyps. There is also recent research on DL-based polyp classification showing state-of-the-art performance [101, 147]. For example, Komeda et al. [71] were able distinguish between adenomatous and non-adenomatous polyps. Some of the results from these classification tasks showed that the model was able to outperform human experts. The number of appearances of popular colonoscopy datasets such as Kvasir-SEG and CVC-ClinicDB are also growing at an increasing pace each year.[1]

Other work focuses on the lack of annotated polyp datasets [67, 137]. For example, Thapa et al. [137] have developed an active learning approach to pixel level polyp annotation, whereas Ji et al. [67] have developed a per-frame annotated video polyp dataset consisting of over 158.690 image frames.

More recently, there is a growing interest in pixel-level classification known as semantic segmentation [64, 79, 143]. In these research problems, the model output is a pixel-wise annotated image where the classes represent objects of interest. In medicine, these annotations can for example correspond to lesions. New DL architectures specifically designed to segment these polyps [36, 65, 66, 85] have shown to provide near-exact predicted annotations of the polyps. These results show how DL models provide enormous value in a clinical scenario because these predictions can help to decrease the number of observational oversights. Even with these few examples of research on polyp segmentation mentioned here, there is still numerous works on this research topic today.

For DL-based CAD systems, the primary goal is to obtain a model ready to be put into clinical workflows to make clinically relevant predictions on unseen data. This means that one wants to obtain a model capable of making predictions that are on par with an experienced endoscopists. In addition to this, one must consider the fact that making false negative (FN) predictions is dangerous in diagnostics. Therefore, one must completely avoid such predictions by making the CAD system sensitive enough, but at the same time keeping the number of false positives (FPs) to a minimum. As mentioned in Section 1.1, this goal has already been reached where CAD has been successfully integrated in commercial diagnostic mammography imaging.

Despite the research achievements of CAD in colonoscopy, the lack of

[1]The number of times Kvasir-SEG have been used in different published research papers can be viewed on https://paperswithcode.com/dataset/kvasir-seg, and https://paperswithcode.com/dataset/cvc-clinicdb for CVC-ClinicDB

explainability and interpretation in DL-models are important limitations when evaluating their implementation in the clinical workflows where the typical end-users have little knowledge of AI. Their ability to outperform medical experts is not sufficient because their limitations make them untrustworthy to these end-users. There are also many potential ethical issues such as employment issues, privacy issues, racial bias in data, lack of trust and accountability which must be considered before real-world deployment. Racial bias, trust, and accountability are the three main ethical considerations discussed in Section 2.8.

XAI is a research field with the focus on developing methods that can extract explainability from DL-based CAD systems. Why is explainability important, one could ask. It can enable humans to understand predictions provided by AI, which efficiently enables humans to make informed decisions. Explainability can also provide confidence in the model prediction by providing clarity in the mechanics behind the model's reasoning. DL models have earlier shown to make predictions based on the inappropriate reasoning [75]. In such cases, explainability methods revealed how these mistakes were made, which can help in avoiding similar problems in the future. With this, explainability systems reduces the need to hire more AIdevelopers for guidance and assistance because these systems enable observers to understand decisions of DL models.

Explainability methods thus enable users to take advantage of what AI has to offer where trust between the AI system and the users has been established. Despite the enormous potential advantages of explainability, there is still much to be done. New explainability methods come with the challenging task of figuring out whether an explanation is correct or not. In addition, many of explanations are qualitative, which make them difficult to interpret. There is also no generic procedure to interpreting qualitative pixel-wise explainability representations, which is still an open research question. Therefore, there is still much to be done in the field of XAI.

2.3 Datasets

To train a model with the properties described in Section 2.2, one expects the training data to represent the typical characteristics of most data instances [39, p. 26]. With this considered, the trained model is expected to recognize unseen instances. In medicine, this corresponds to obtaining data that covers all general medical scenarios that can occur on a population level. Patient characteristics such as age, gender, and ethnicity, etc. should optimally also be considered.

For the sake of reproducibility, the work in this thesis includes the use

of a publicly available dataset to enable others to reproduce the experiments conducted. This thesis includes the use of one well known and commonly used dataset containing image frames and their corresponding segmentation masks collected from colonoscopy examinations. The following datasets used for model training and evaluation are: Kvasir-SEG [106]. Additionally, we mention two other datasets which are central in the research on segmentation in colonoscopy imaging, namely CVC-ClinicDB [4] and ETIS-Larib Polyp DB [125]. An overview of the mentioned datasets is given in Table 2.1.

Table 2.1: Table summarizes the main features of the segmentation datasets used in this thesis.

	Kvasir-SEG	CVC-ClinicDB	ETIS-Larib
Size	1000	612	196
Resolution	720×576–$1,920 \times 1,072$	384×288	$1,225 \times 966$
Contain annotations	Yes	Yes	Yes
Contain bounding boxes	Yes	No	No

2.3.1 Kvasir-SEG

Kvasir-SEG is a publicly available dataset consisting of 1,000 colonoscopy images containing annotated polyps provided by experienced endoscopists. The annotations are in the form of binary segmentation masks, which highlight the polyps. The dataset also includes *.json* files which correspond to the bounding boxes surrounding the polyps.

The dataset is an extension of the Kvasir dataset [106], which is also rooted in the largest GI dataset, Hyper-Kvasir [5], consisting of a collection of about one million image and video frames of the GI tract. The dataset was made in collaboration between Simula Research Laboratory, the Norwegian Cancer Registry and Vestre Viken Health Trust (VV) in Norway. The images are gathered from colonoscopy examinations at VV. The segmentation masks are annotated and verified by experienced endoscopists from VV and the Norwegian Cancer registry.

The images in the dataset vary from 720×576 up to $1,920 \times 1,072$ pixels. The ground truth segmentation masks consist of white and black pixels. The white pixels represent the polyp tissue, and the black pixels represent areas that are not part of the polyp, which is also called the background. The bounding boxes represent the region of interest, which defines the border pixels of the polyps. The region of interest coordinates in the *.json* files were used to make the ground truth segmentation masks.

The original images in the dataset, where some examples can be viewed in figure 2.2, contain a small green box also called an anchor object of the scope guide (used to determine where the scope is during the procedure) in the bottom left or right corner of the image as seen in Figure 2.1. These green anchor objects were later covered by a black corner object, as seen in Figure 2.2, to avoid the potential case where a prediction model learns to associate the anchor object with the target [51].

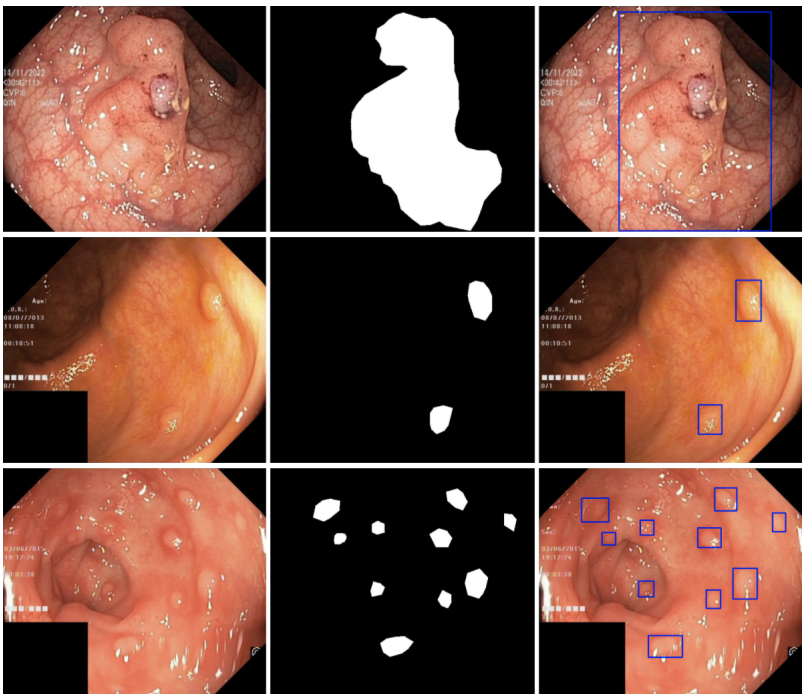

Figure 2.2: Three example images extracted from the Kvasir-SEG dataset [106]. From the left column: (1) represents the original images from the colonoscopy, (2) represents their corresponding ground truth annotations, and (3) represents their bounding boxes.

2.3.2 CVC-ClinicDB

The CVC-ClinicDB dataset consists of 612 image frames of size 384×288 and their corresponding ground truth segmentation masks. This dataset contains numerous examples of polyps. The database was made in collaboration with Bernal et al. [4], along with the Hospital Clinic of Barcelona. The database was introduced as the training dataset of the ISBI 2015 Challenge on Automatic Polyp Detection Challenge in Colonoscopy Videos.

The images in the dataset are from various parts of the GI tract. They were extracted from 25 different videos from standard colonoscopy examinations. When collecting the images, the objective was to collect different images of the same polyps but from different angles.

2.3.3 ETIS-Larib Polyp DB

The ETIS-Larib Polyp DB consists of 196 image frames of size $1,225 \times 966$ and their corresponding ground truth segmentation masks [125]. The images in this dataset were collected from colonoscopy videos using wireless capsule endoscopy, which utilizes a small wireless camera in the shape of a small pill. It is designed to take images of the GI tract and transmit data to a nearby receiver. The images in this dataset are extracted from a total of 34 different video sequences containing 44 different polyps.

There is minor variation to the polyps in this dataset, and many are of small size in contrast to the variety of polyps in the Kvasir-SEG dataset.

2.4 Neural Networks

Neural networks (NNs) are utilized in AI and machine learning problems to model complex relationships in data. NNs are commonly seen in computer vision [115], natural language processing [21] and weather forecasting [112] tasks.

NNs can recognize complex underlying relationships in large data of high-dimensional nature. Given some input x, and the corresponding output y, the goal is for the NN to produce the correct output given an input, with the following mapping function

$$f(x) = y. \tag{2.1}$$

Equation 2.1 represents the aim of all NNs, which is to approximate some unknown mapping function f. This section introduces one of the fundamental building blocks of NNs that can produce these mappings, which are the affine transformations and activation functions.

2.4.1 Feed-forward Neural Networks

Feed-forward NNs consist of multiple layers. One such layer consists of stacks of neurons. Each neuron has learnable weights and biases (i.e., parameters) and a non-linear activation function. The input along with the parameters are used in the computation of an affine transformation [145]. In this affine transformation, the linear transformation is represented by the weighted sum of the inputs and the translation is represented by the added bias. A non-linear activation function is then applied to the affine transformation. Non-linear relationships in the mapping 2.1 is represented by activation functions, which is further described in Section 2.4.2.

Figure 2.3 represents one example of a feed-forward NN architecture.

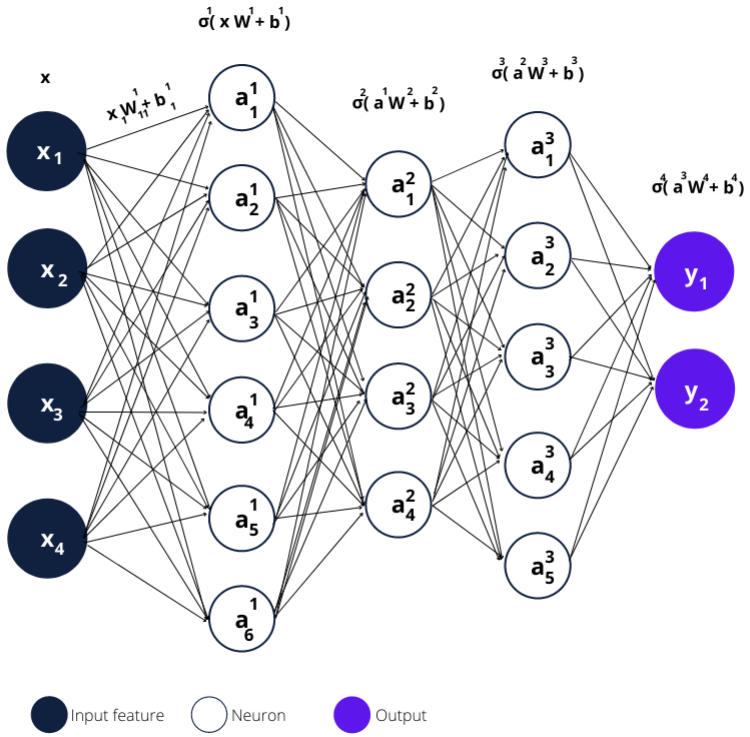

Figure 2.3: Computational graph of a fully connected feed-forward NN containing four input neurons, three hidden layers, and two output neurons.

The intermediate column stacks of neurons placed between the input layer (first layer) and output layer (last layer) are referred to as *hidden layer*. The input to each individual neuron in the first layer, as pictured in Figure 2.3, represents a feature from the input data. This can for example be an image pixel, height coordinate, or temperature.

Neurons in hidden layers that are not placed immediately after the input layer get their input from neurons from an neighboring upstream layer. In a *fully connected* NN architecture, the outputs from all the neurons in layer $l-1$ are passed to each neuron in layer l, such that each individual neuron in the two layers are connected.

The information passed from a stack of n neurons in layer $l-1$ to layer l containing m neurons in a fully connected network can be represented by [39, p. 283]

$$\boldsymbol{a}^l(\boldsymbol{X}) = \sigma^l(W^l\boldsymbol{X} + \boldsymbol{b}^l), \tag{2.2}$$

where $W^l \in \mathbb{R}^{m\times n}$ denotes the weight matrix, with m rows for each input neuron from layer l, and n columns for the number of neurons in layer $l-1$. The bias vector is represented by $\boldsymbol{b} \in \mathbb{R}^{m\times 1}$, such that one vector element gets added to each neuron in layer l. The raw input

column vector has one row per instance, which is represented by $\boldsymbol{X} \in \mathbb{R}^{n \times 1}$. The non-linear activation function in layer l, σ^l, is added after the affine transformation, which is applied elementwise, i.e., one instance at a time [145]. The output of layer l is represented by $\boldsymbol{a}^l(\boldsymbol{X})$, such that the output in the next layer $l+1$ is represented by

$$\boldsymbol{a}^{l+1}(\boldsymbol{a}^l) = \sigma^{l+1}(W^{l+1}\boldsymbol{a}^l + \boldsymbol{b}^{l+1}), \tag{2.3}$$

where the affine transformation (in the rightmost parenthesis) connects the neurons in layer l, represented by $\boldsymbol{a}^l$, with the neurons in the next layer, $\boldsymbol{a}^{l+1}$, and the activation function σ^{l+1} transforms $\boldsymbol{a}^l$ before being passed to layer $l+1$. In more general terms, the information that propagates through hidden layers, represented by Equation 2.3, is a chain of affine transformations and activations between previous layers.

In summary, what happens when information traverses a NN is the following: (i) a series of matrix-matrix multiplications, (ii) affine transformations, and (iii) non-linear activations between layers. This series of calculations that start at the first hidden layer and end at the output layer are referred to as the forward propagation, or the forward pass.

2.4.2 Activation Functions and Universal Approximators

Without the activation functions between each layer, the information that is passed forward in a NN will only be a series of affine transformations. Linearity is a strong assumption and is rarely the case in large high-dimensional datasets. The activation functions add non-linearity, which is a mechanism shown to work well for complex data. This is the reasoning behind why NNs are said to be *universal approximators*.

NNs have the ability to capture complex patterns in data with their hidden layers [72, 135]. The following studies show that for certain assumptions, such as a specific choice of activation function and number of layers, NNs can approximate any continuous function. Cybenko [14] shows that a NN with at least one hidden layer using the Sigmoid activation function can approximate any continuous function. The capability of approximating any continuous function has been proven universally for all non-polynomial activation functions [50], such as the Rectified Linear Unit (ReLU) function.

The property of universal approximation for continuous functions has also been proven for CNNs and recurrent neural networks [120]. For example, Zhou [149] argues that universality applies to deep CNNs, and can hence approximate any continuous function up to an arbitrary accuracy if the depth of the CNN is large enough.

2.4.3 Gradient Descent Optimization

Randomly initialized NNs will in most cases not be able to approximate the mapping in Equation 2.1. For it to do so, the parameters are iteratively adjusted with an optimization algorithm called gradient descent (GD) [114]. The end-goal for such an algorithm is to converge at a minimum relative to some scoring function. For problems in DL and machine learning, this scoring function is referred to as the loss function. Minimizing the loss function is the fundamental part to a training regimen for a NN, referred to as *backpropagation*. The name arrives from the fact that the output from a forward pass propagates backwards in the network relative to the trajectory of a forward pass. One forward- and backpropagation of data is referred to as one *epoch*. Note that there are several variations of GD, which all are described by the end of this section.

For some input, x, the output from the forward pass, $\hat{y}$, is compared to the true output y. The loss function, $C(\hat{y}, y)$, computes a measure of their dissimilarity. The goal is to let the GD algorithm converge at a minimum loss value. The gradients of the stored parameters from the forward pass with respect to C are computed in backpropagation. These gradients give information on how changing the model parameters will change C.

The partial derivatives of the stored parameters from the hidden layer l with respect to C are given by

$$\frac{\partial C}{\partial W_{jk}^{l}} \quad \text{and} \quad \frac{\partial C}{\partial b_{jk}^{l}}, \tag{2.4}$$

where W_{jk}^{l} denotes the weight element from the k-th neuron in the $(l-1)$-th layer to the j-th neuron in the l-th layer. W_{jk}^{l} is an element of the weight matrix W^{l}. This notation also applies to the bias element b_{jk}^{l}.

The values in Equation 2.4 can be vectorized to represent the gradients of an entire set of input data. Suppose $\boldsymbol{\theta}$ represents some column vector $[\theta_1, \theta_2, ..., \theta_n]^T$. Its corresponding multivariate gradient is represented by $\nabla C(\boldsymbol{\theta})$, such that

$$\nabla C(\boldsymbol{\theta}) = \left[\frac{\partial C(\boldsymbol{\theta})}{\theta_1}, \frac{\partial C(\boldsymbol{\theta})}{\theta_2}, ..., \frac{\partial C(\boldsymbol{\theta})}{\theta_n} \right]^T, \tag{2.5}$$

where $\nabla C(\boldsymbol{\theta})$ represents the entire loss gradient computed for the output vector $\hat{y}$ and the ground truth y.

If the gradients in Equation 2.4 are vectorized equivalent to that of Equation 2.5, then each element of the weight matrices and bias vectors of layer

l are updated at iteration t such that

$$W_{t+1}^l = W_t^l - \eta \cdot \nabla C(W_t^l), \tag{2.6a}$$

$$\boldsymbol{b}_{t+1}^l = b_t^l - \eta \cdot \nabla C(\boldsymbol{b}_t^l). \tag{2.6b}$$

Equations 2.6 can be derived from the Taylor expansion of $C(\theta)$.

For the case where $\hat{\boldsymbol{y}}$ represents the outputs for the entire dataset $\boldsymbol{x}$ passed to a NN, the GD algorithm in equations 2.6 is referred to as batch GD.

Note that many DL frameworks such as PyTorch [103, 104] and TensorFlow [1] handle gradient computation of NNs by automatic differentiation [3]. In short, automatic differentiation in these frameworks is done by compiling the network operations to a computational graph. The multiderivatives are obtained by computing the partial derivatives between each node in the graph, and then multiplying the connecting nodes together as an approximation to the multivariate chain rule.

The parameters are updated with respect to the loss, measuring the performance of the network at each iteration. Thus, the network parameters adapt more of the underlying patterns of the training data with each iteration. Therefore, the NN parameters are said to be learnable, as mentioned at the beginning of Section 2.4.1

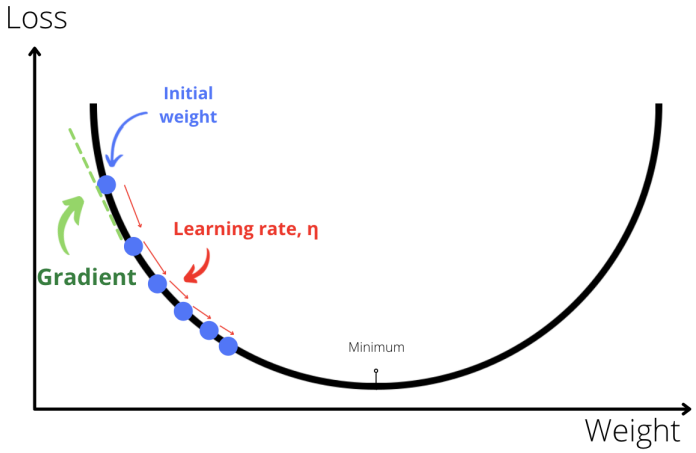

Figure 2.4: This figure illustrates the concept of a gradient descent algorithm for one dimension using an adaptive learning rate.

In Equations 2.6, η denotes the learning rate hyperparameter. Its function is illustrated in Figure 2.4. This defines the step size at each training iteration in the direction of the negative gradient. A too small learning rate requires many iterations before convergence but using a too large learning rate can cause the algorithm to overshoot and fluctuate around

the minimum, or in other cases cause the algorithm to diverge from the minimum. Introducing learning rate schedulers [15] to the training scheme is one way of avoiding problems of either a too large or too small learning rate. This method reduces the learning rate by some pre-defined schedule, e.g., by reducing the learning rate by a factor n for every m epochs if the loss has not improved by some threshold.

Mini-Batches and Stochastic Gradient Descent

Computing the gradients of the loss function for a substantial amount of input data is computationally expensive and time inefficient. Dividing the entire training data into mini-batches and updating the parameters with respect to one mini-batch reduces iteration time [145]. This is because it requires less computational capacity and hence increases time efficiency. An increasing convergence time is a negative consequence of using mini-batches. For the case of selecting instances randomly into the mini-batches, the optimization scheme is referred to as *stochastic mini-batch GD* or *mini-batch stochastic gradient descent (SGD)*. Using a different mini-batch with every iteration brings some stochastic variation to the computed loss because every mini-batch is a random combination of different inputs. Using mini-batch SGD comes with the benefit of computational efficiency.

Using mini-batches introduces the *batch size* as one additional hyperparameter to the optimization scheme. The batch size is the amount of training data for one batch. When choosing a batch size equal to a single training instance, the algorithm is known as SGD [69].

Momentum Optimization

There are several methods that can be added to the GD algorithm to shorten the computation time of one optimization step but can also handle saddle points of loss functions [16]. Many methods utilize *momentum* [110] to accelerate convergence time. Different adaptations of momentum leads to several versions of GD, such as Root Mean Square Propagation (RMSProp) and Adaptive Moment Estimation (Adam) [70].

The concept of momentum was introduced in the momentum optimization algorithm [107]. The momentum vector takes the previous gradients into account when computing the gradient. This is done by computing the exponentially decaying moving average of past gradients. The local gradient (multiplied by η) is subtracted from the momentum vector. Momentum speeds up convergence in GD by increasing their ability to escape plateaus and ravines of loss surfaces [132]. In momentum optimization,

the parameter vector $\boldsymbol{\theta}$ is updated by [145]

$$\boldsymbol{m}_t = \beta \boldsymbol{m}_{t-1} + \eta \cdot \nabla C(\boldsymbol{\theta}_t), \tag{2.7a}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{m}_t, \tag{2.7b}$$

where $\boldsymbol{m}_t$ represents the momentum vector for iteration t, and $\beta \in [0, 1]$ represents the momentum scaling parameter which scales the grade of momentum. $\boldsymbol{\theta}$ represents the objective network parameter vector used the optimization.

RMSProp

RMSProp[2] is a GD-based optimization algorithm that requires little tuning of the learning rate hyperparameter due to having an adaptive step size. The gradients for each iteration can vary in magnitude, which leads to difficulty in determining a global learning rate. The local gradient is scaled by the most recent gradients by being divided by an exponentially decaying average of squared gradients, such that [145]

$$\boldsymbol{s}_t = \gamma \boldsymbol{s}_{t-1} + (1 - \gamma) \nabla C(\boldsymbol{\theta}_t) \otimes \nabla C(\boldsymbol{\theta}_t), \tag{2.8a}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\boldsymbol{s}_t}} \otimes \nabla C(\boldsymbol{\theta}_t), \tag{2.8b}$$

where $\otimes$ represent the Kronecker product [146]. Equations 2.8 ensures that each step size adapts by adjusting to the steepness of the slope of the loss surface. The additional hyperparameter, the decay rate, γ, controls the grade of adjusting/scaling, and is typically set to 0.9 [117]. Thus, even for small local gradients, scaling the gradient becomes significant in escaping plateaus and saddle points in the direction needed.

Adam

Adam [70] is an optimization method combining the features of momentum optimization and RMSProp. Adam can be viewed as an extension of RMSProp by including momentum. Utilizing both momentum and an

[2]The algorithm was introduced by Hinton et al. in 2012 in a lecture in a course on NNs. It is also famous for not being published despite being a popular GD method. The lecture can be found at slide 29 on the following url: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

adaptive step size is mathematically represented by [145]

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} - (1-\beta_1)\nabla C(\boldsymbol{\theta_t}), \tag{2.9a}$$

$$\boldsymbol{s}_t = \beta_2 \boldsymbol{s}_{t-1} - (1-\beta_2)\nabla C(\boldsymbol{\theta_t}) \otimes \nabla C(\boldsymbol{\theta_t}), \tag{2.9b}$$

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-\beta_1^t}, \tag{2.9c}$$

$$\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_2^t}, \tag{2.9d}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_t}} \otimes \hat{\boldsymbol{m}}_t. \tag{2.9e}$$

Equation 2.9a represents the exponentially decaying average and Equation 2.9b represents the adapting step size from RMSProp. Since $\boldsymbol{m}_t$ and $\boldsymbol{s}_t$ are initialized by zeros, the authors behind Adam argue that the parameters are biased towards zero. Because of that, Equation 2.9c and 2.9d represent the bias-corrected computations that adjust for this, and the bias-corrected terms are hence used to update the parameters $\boldsymbol{\theta}$ in Equation 2.9e.

2.4.4 Initialization of Neural Networks

Building a NN requires a choice of parameters to the hidden layers. The weights are typically initialized by some distribution. He initialization [48] (also called Kaiming initialization) and the initialization proposed by Xavier Glorot and Yoshua Bengio [41] are the most commonly used initialization methods. They are similar, but the former is an initialization scheme drawn from a zero-centered Gaussian distribution, scaled by $\sqrt{2/n_l}$, such that

$$W^l \sim \mathcal{N}(0, \sqrt{2/n_{l-1}}), \tag{2.10}$$

where W^l is the weight matrix in the hidden layer l, n_{l-1} is the number of neurons in the previous hidden layer $l-1$, and $\mathcal{N}$ represents the Gaussian distribution. This division is done elementwise to the matrix.

Xavier Glorot and Yoshua Bengio's initialization draws weight values from the uniform distribution in $[-1, 1]$, scaled by $\frac{1}{\sqrt{n_{l-1}}}$, such that

$$W^l \sim U\left[-\frac{1}{\sqrt{n_{l-1}}}, \frac{1}{\sqrt{n_{l-1}}}\right], \tag{2.11}$$

where U represents the uniform distribution. Both initialization schemes initialize all the bias vectors with zeros.

2.4.5 Overfitting in Neural Networks

Overfitting is a common problem of NN training [96]. This occurs after the NN has adjusted its model parameters during training and becomes

too well-tailored onto the training data [39, p. 28]. The resulting NN is deemed overfitted to the training data, such that it is can capture the noise of the training data. This makes the NN unable to generalize onto unseen data. Hence, an overfitted model will obtain a high score by predicting on the training instances, but a low score on unseen instances.

Variance and Bias

The generalization error of machine learning models can be characterized in terms of its variance and bias [39, p. 136]. Variance is a metric that represents the dispersion in a given dataset by computing the distance from a given instance relative to the dataset mean. A dataset with large amounts of noise will have a high variance, which is why variance is often mentioned in the context of overfitting. In mathematical terms, the sample variance is represented by

$$\text{Var}(x) = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2, \tag{2.12}$$

where $\bar{x}$ represents the dataset or sample mean, x_i represents a single instance, and N represents the sample size. A model with a high variance is said to be an overfitted model [39, p. 136]. In the opposite case, where the model has not yet captured the underlying characteristics of the training data, the model is said to underfit the training data due to a large *bias* [39, p. 30]. Bias is a metric which represents the systematic error of predictions given by a model [39, p. 136]. This is given by

$$\text{Bias}(\hat{y}_i) = E[\hat{y}_i] - y_i, \tag{2.13}$$

where y_i denotes a ground truth instance, and $E[\hat{y}_i]$ represents the mean prediction of y_i given by multiple model predictions. A common scenario for a high-bias model is that it has stopped training before it was able to learn the characteristics of the training data.

Dividing the Dataset

A common practice in DL is to divide the total dataset into three parts: a training, validation and test dataset. The training set is used to train and update the model parameters. The validation set is held back during gradient computation but used during the training phase *only* for the forward pass in order to tune the configuration of hyperparameters in the model [39, p. 32].

The test set is held back until the final one-time evaluation of the trained

model. Hence, the loss function during evaluation can be used as an unbiased scoring metric based on this last evaluation to assess its generalization quality [39, p. 31]. The use of loss functions as scoring metrics is described in Section 2.7.

Bias-Variance Tradeoff

Bias and variance are conflicting metrics. Hence, one cannot simultaneously minimize both. In model training, the aim is to obtain a balance between the two. A common procedure is to compute the model loss with respect to the training data and another loss with respect to the validation data with each training iteration. The model becomes more biased toward the validation set for each iteration but tracking the training- and validation loss set for each iteration can assist in detecting overfitting in NNs [38].

For each iteration, the training loss will typically become smaller and smaller, and thus leading to an overfitted model with high variance. The validation loss will reduce up to a certain point of iteration but will increase because the model becomes more unable to generalize. However, stopping model training too early leads to a model of high bias. With the tradeoff between bias and variance during model training, the goal is to continue training until validation loss is minimized [39, p. 142].

2.5 Convolutional Neural Networks

For larger images, regular feed-forward NNs become inefficient. Introducing convolutional layers (referred to as convolutions) to the feed-forward NN architecture gives rise to CNNs. They are a class of NNs specifically designed to analyze image data. Because convolutions have been proven to capture the characteristics of image data, almost every application in the field of computer vision is based off the use of CNNs. These applications include object detection, image segmentation, image classification, image reconstruction, video tracking, etc.

One large-scale CNN, *AlexNet* [72], showed large improvements compared to previous networks, and won the 2012 ImageNet Large Scale Visual Recognition Challenge [19]. Shortly after acquiring the state-of-the-art performance, CNNs gained a large surge in popularity. The success of AlexNet spurred papers that proposed more innovative CNN architectures that have long ago surpassed its performance, such as VGGNet [126], GoogLeNet [133] and ResNet [47].

This section starts off by introducing the most fundamental building

blocks of a CNN, before diving into the most modern architectural CNN designs of today's time described in the next section.

2.5.1 Convolutional Layer

The convolution operation in a convolutional layer invokes two important principles that exploit the characteristics of images [145]:

1. Locality, meaning that only a small local area of the image is used to compute the corresponding hidden representations.

2. Translation invariance, meaning that the CNN should be independent to translation of the input image, meaning that the network should react the same to a patch in the image, no matter where in the image it appears.

Images given as input to CNNs are multidimensional arrays that are referred to as *tensors*. The images are typically four-dimensional tensors where two dimensions represents the spatial dimensions, one dimension represents the color channel since images have RGB values representing colors in the image, and one dimension representing the batch size.

Another contrast to the NNs introduced earlier is that the layers of CNNs are not fully connected. Given this, each neuron in, e.g., the first hidden layer is only connected to a small part of the input image. The connections are small, localized regions referred to as a *local receptive field* for a neuron. Having a restricted receptive field means that the first principle is invoked. This mechanism allows for the network to learn many small low-level features, then assemble these into high level-features in the deeper parts of the networks [39].

Each neuron in a hidden layer learns a small patch of the image, where the neuron and the image patch are connected by a weight matrix and an overall bias. The local receptive field slides across the input image (or over each image patch illustrated by the empty black square in Figure 2.5) and looks for the same feature at different locations in the image since each neuron in a hidden layer shares the same parameters. The neuron's shared weights and bias are represented by a tensor, called a kernel. Its size is equal to the local receptive field, referred to as the kernel size. For example, in Figure 2.5, the kernel size is 3×3. Thus, the network does not need separate detectors (kernels) looking for the same feature [82], which significantly reduces the number of parameters in a CNN. This level of parameter sharing invokes the second principle.

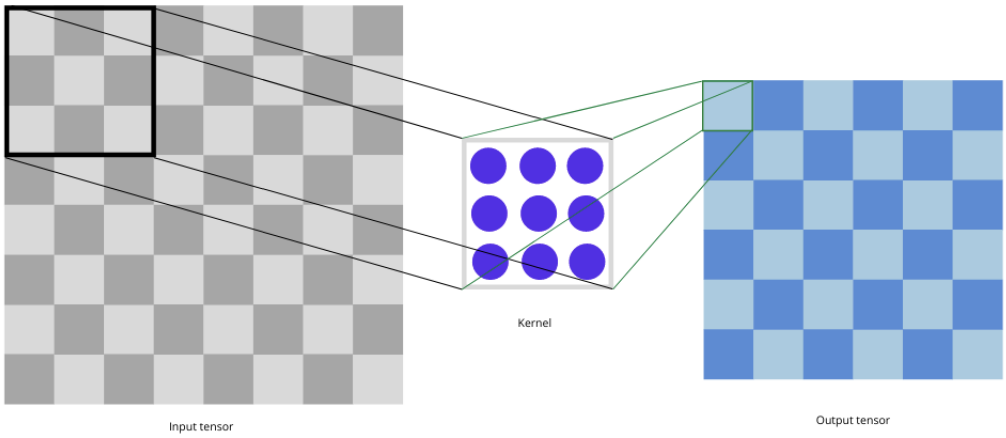

Figure 2.5: The figure illustrates a convolutional layer of one input channel and filter size of 1. The leftmost matrix (in gray) represents the input tensor, the middle matrix (purple dots) represents the kernel, and the rightmost matrix (in blue) represents the output tensor.

By adapting the notation in Section 2.4.1, suppose that layers $l-1$ and l contain one kernel each. Then the input to layer l, represented by the tensor $x_{i,j}^{l-1}$ activated by an activation function σ^l of layer l, is given by

$$a_{i,j}^{l-1} = \sigma^l(x_{i,j}^{l-1}), \tag{2.14}$$

where i, j are iterators over the height and width of the input $a_{i,j}^{l-1}$. Given this, the output of the convolutional layer l with kernel size $k_x \times k_y$ is given by [39, p. 439]

$$z_{i,j}^{l} = b^l + \sum_{m=0}^{k_x-1} \sum_{n=0}^{k_y-1} W_{m,n}^{l} * a_{i+m,j+n}^{l-1}, \tag{2.15}$$

where $*$ is the 1D convolutional operator. This convolutional mapping in Equation 2.15 is referred to as the *feature map* [39, p. 436].

If the number of kernels in layer $l-1$ (which also corresponds to the number of output channels C_{out} of layer $l-1$) is larger than one, one must include a third summing operator to Equation 2.15. Suppose that there are multiple kernels C_{out} of layer l. Given this, the weight tensor connecting layers $l-1$ and l is represented by $W^l \in \mathbb{R}^{C_{out} \times C_{in} \times k_x \times k_y}$ and $b^l \in \mathbb{R}^{C_{out}}$, and the operator $*$ becomes $\star$, which represent the 2D convolutional operator. A collection of kernels in the same layer (also known as the channel size) is known as a *filter* [39, p. 436].

In addition to the kernel size hyperparameter of each convolutional layer, one must also determine the stride. This is the step size of the sliding window over the image, which can connect differently sized feature maps

together [39, p. 432]. In Figure 2.5, the stride is equal to 1 because it moves only one pixel at a time before the convolutional operation at the next location. One can imagine that the window starts from the left corner and ends at the right corner in the input feature map, thus moving in a left-right direction for each row. The stride, together with kernel size, determines the output feature map from the convolution layer.

To summarize, a convolutional layer applies a trainable filter which consists of multiple kernels, which makes it able to detect multiple features at once independently of where they appear in the image [39, p. 437].

2.5.2 Pooling

A pooling layer also known as a downscaling layer, reduces the spatial dimension in a CNN. This downscaling mechanism reduces the computational cost for all subsequent convolutional layers by reducing the total number of computations. Pooling layers can drastically minimize the number of parameters in a CNN by condensing the information in the image [82], thus reducing the need for deeper networks as the receptive field of each hidden layer increases by doing so. Pooling layers also provide a slight rotational invariance, a translational invariance at a larger scale, and scale invariance [39, p. 458].

There are several variants of pooling. For example, average-pooling computes the average of each window, whereas max-pooling computes the maximum of each window [145]. Average-pooling are more computationally expensive than max-pooling. Given some pre-determined output size, adaptive average pooling uses the average-pooling operation, in addition to calculating the correct kernel size to produce the desired output size. One extreme version of average-pooling is global average-pooling, which takes the average of an entire feature map [81]. No matter what variant, pooling operations reduce the output size after each convolutional layer.

Two hyperparameters connected to the computations in pooling layers are the kernel size and stride of the sliding window. Figure 2.6 demonstrates the effect of a pooling layer. Note that the stride can also be referred to as the scale factor as it, together with the kernel size, determines the scaling of the output tensor relative to the input.

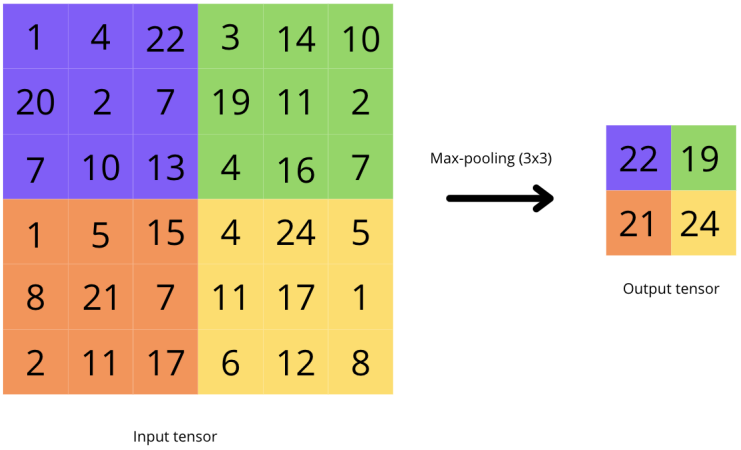

Figure 2.6: A two dimensional max-pooling layer with a 3×3 kernel and a stride of 3. The spatial dimensions are reduced by a factor 3.

2.5.3 Upsampling

Upsampling layers counteract the operation done by pooling layers. They do so by increasing the spatial resolution of a feature map. There are several methods for upsampling. Bilinear upsampling (or bilinear interpolation) uses the nearby pixel values to calculate a weighted average of the new nearest pixel value by using linear interpolations [123]. Nearest-neighbor upsampling copies the values of the neighboring pixels [99]. Dilated convolutions (also called *atrous convolutions*) use a large kernel size, where some values are set to zero to minimize the number of parameters [11]. Transposed convolutions (or deconvolutions) inject additional zero-valued pixels at random locations in the input image before the convolutional operation to increase the output resolution [123].

Two hyperparameters connected to all variants of upsampling layers are the *scale factor* and stride, which are equivalent to the ones for a convolutional layer. A scale factor of two can double the size of the original image, a scale factor of three triples it, and so on.

Dilated convolutions include an additional hyperparameter to the scaling factor, which is the *dilation factor*. This controls the level of dilation of the kernel, i.e., the factor of weights set to zero.

2.5.4 Dropout

Dropout is a regularization technique used in DL models [53] to reduce overfitting of NNs [127]. One applies dropout to a layer by randomly

ignoring or zeroing out the values of neurons during network training. Figure 2.7 demonstrates the mechanisms of a dropout layer in a NN.

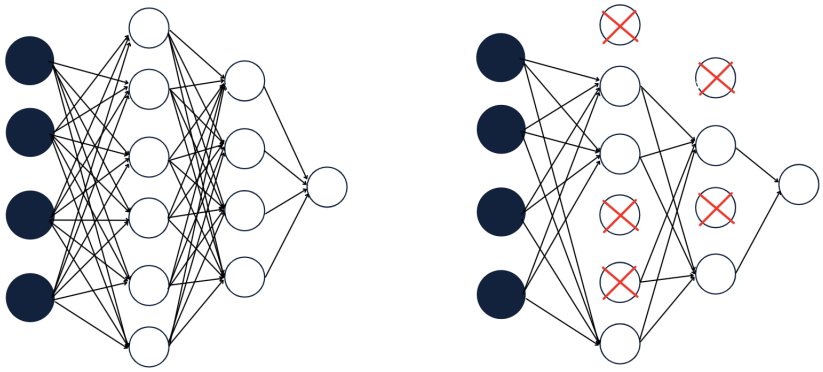

Figure 2.7: This figure illustrates the effect of a dropout layer in a fully connected neural network. From the left: (1) a network before dropout, (2) a network after applying dropout. The circles with red crosses represent the neurons that are dropped.

One hyperparameter linked to the use of dropout layers is the *dropout rate*, which is the probability of one neuron being dropped.

Because of dropout layers, a network of one forward pass can be viewed as an entirely different network compared to one in the next forward pass [93]. Hence, the effect of dropout can be represented by an ensemble of different NNs [127]. Model ensembling has proven to increase model performance of NNs, as seen in many different applications [24, 136].

2.5.5 Batch Normalization

Batch normalization, or batch-norm, is a technique added to deep CNNs to provide better gradient flow and avoid exploding gradients. The inventors of batch-norm state that it improves convergence by handling the internal covariate shift problem [58]. One recent study argues that batch-norm layers accelerate optimization by smoothing the loss surface [119]. Despite these disagreements on why batch-norm improves training, it has empirically shown to accelerate optimization [58]. Batch-norm ensures centering and rescaling of feature maps it is applied to, which hinders divergence due to the controlled magnitude of gradients during training. Therefore, it is considered a standard practice to include batch-norm layers in CNNs [47, 65, 66].

As described in the original batch-norm paper, batch-norm should be applied as a layer after the activation function [58]. At a batch-norm layer, the input feature map from a previous layer is normalized by subtracting the

mean and dividing by the standard deviation based of the current mini-batch statistics. Given this, batch-norm does not work on models where batch size is set to one as each feature map would become tensors of zeros.

Consider the feature map $a_c^l \in \mathcal{B}$ of mini-batch $\mathcal{B}$ from the l^{th} layer and c^{th} channel, as input to a batch-norm layer BN, such that [145]

$$\text{BN}(a_c^l) = \gamma \cdot \frac{a_c^l - \hat{\mu}_{\mathcal{B}}}{\hat{\sigma}_{\mathcal{B}}} + \beta, \tag{2.16}$$

where $\hat{\mu}_{\mathcal{B}}$ and $\hat{\sigma}_{\mathcal{B}}$ is the mean and standard deviation of mini-batch $\mathcal{B}$. The parameters γ, β are learnable scale and shift parameters, respectively, to each individual batch-norm layer. These parameters have shown to give improved results of CNNs [33], and are therefore standard to the batch-norm layer of TensorFlow and PyTorch.

2.6 Modern Convolutional Neural Networks

CNNs have changed to a great extent since their first introduction in the 1980s [77]. Significant improvements of the performance of CNNs are achieved through architectural advancements. These innovative architectures exploit skip connections, information across channels, and combine feature maps of multiple scales.

This section covers several CNNs that introduce the state-of-the-art architectural improvements that have influenced many of today's research projects [65, 134].

2.6.1 ResNet

The Residual Neural Network (ResNet) paper introduced the concept of residual learning to CNNs [47]. Residual learning is achieved with skip connections added to the CNN architecture to avoid the problem of vanishing gradients of deeper networks [41]. Skip connections have shown to efficiently utilize the upstream feature maps, resulting in significant performance boosts compared to deep CNNs that lack them. This was proved by winning the 2015 ImageNet Large Scale Visual Recognition Challenge. The simple improvement of including residual learning allows for accelerated training, deeper networks and lower convergence.

Instead of learning the underlying mapping 2.1 for each added layer, one layer can learn the residual mapping $f(\boldsymbol{x}) - \boldsymbol{x}$ from an upstream layer. When a skip connection connects two blocks of layers in a CNN, as illustrated by the dotted lines in Figure 2.8, the downstream block only learns the residual mapping by using $f(x)$ as input to the activation function.

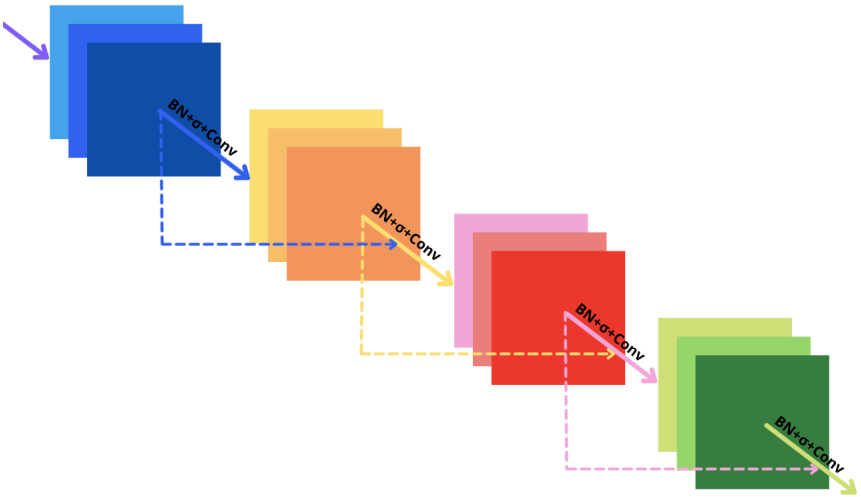

Figure 2.8: Skip connections how they are commonly utilized in a CNN. The solid arrow between each block represents a batch-norm layer (BN), a ReLU (σ) layer and a convolutional layer (Conv). The dotted arrows represent the skip connections added to the ReLU layer as input.

As CNNs becomes deeper, the number of non-linear activations increases. Given this, the high-dimensional loss function surface becomes of more non-convex nature [16]. Skip connections reduce this effect by zeroing out weights of intermediate layers, which in turn reduces the non-convex nature of the loss surface.

Each batch-norm layer in the ResNet architecture is followed by a convolutional layer, which is again followed by a ReLU activation layer. For deep CNNs, this activation function is deemed as a standard activation function because it handles the vanishing gradient problem. This is because it does not produce small derivatives. The gradient of ReLU is either 1 for inputs larger than zero, or 0 for negative inputs. Therefore, multiplying multiple ReLU derivatives together results in a product of either 0 or 1.

2.6.2 U-Net

U-Net is a CNN architecture originally designed for biomedical image segmentation [115]. U-Net adapts its name from the illustrative shape of its U-shaped architecture, as depicted in Figure 2.9.

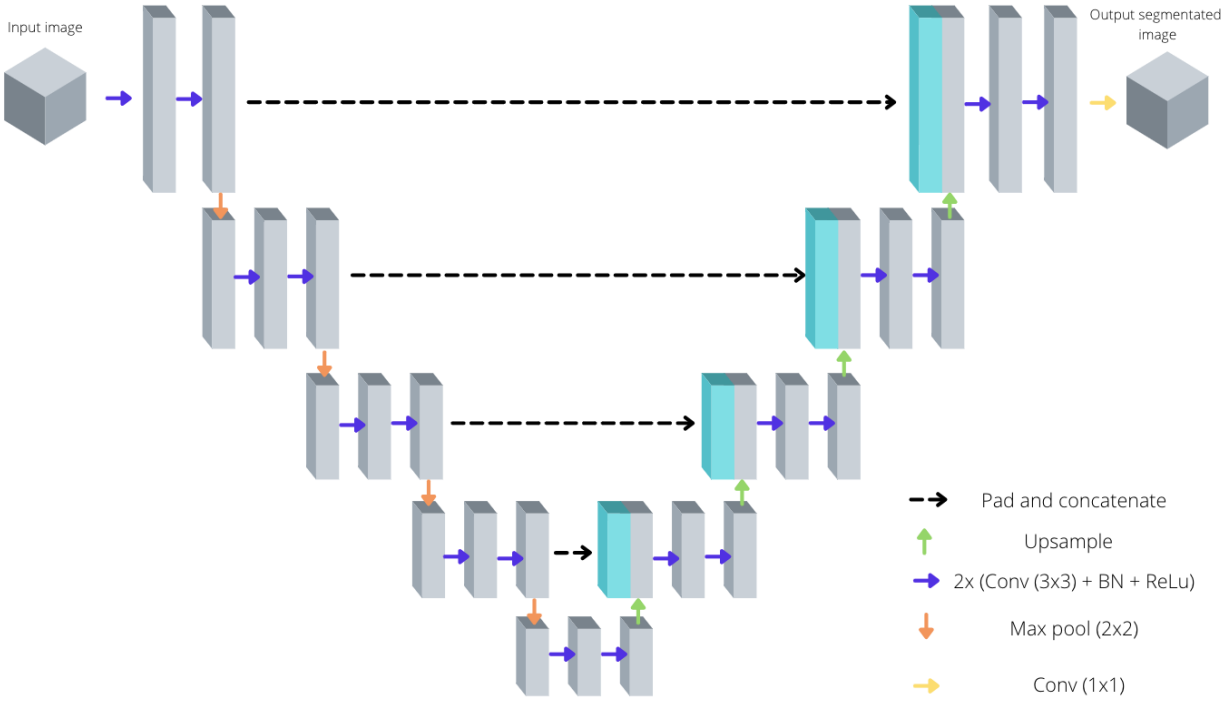

Figure 2.9: This figure represents a simple example visualization of the U-Net architecture performing segmentation of an input image. Each gray box represents a feature map. The blue boxes represent copied boxes from the first half of the network added from the skip connections.

The architecture adapts the skip connections from ResNet. The skip connections allow the strength of low-level features to be combined with high-level features to maintain resolution of the image as it traverses the network, but at the same time maintain the high-level semantics of the image. U-Net is commonly used as a baseline model in many DL applications. It is also known for its simple design and for being able to achieve the state-of-the-art performance with little (a few hundred images) training data due to using data augmentation methods, more specifically, elastic deformations to the images in order to distort them.

In Figure 2.9, the first half of U-Net architecture learns a set of features. Each feature can be treated independently. When going deeper into the network, one ends up with a low-resolution feature map with a large receptive field containing low-level feature representations. They can contain strong semantic information but usually lack spatial information due to the low resolution. Due to the strong semantic information contained within these features, they are considered semantically strong. In contrast to these, the features at the shallow parts of the network have a higher resolution but contain low semantic information and are hence considered semantically weak.

As seen in Figure 2.9, the skip connections act as bridges which connect the semantically strong features in the first half with the semantically weak features in the second half. Combining these features results in a high-resolution output containing high-level semantics. Such an output makes the network suitable at predicting the "what" and the "where" in an image. U-Net a typical network design used in semantic segmentation problems, where there is a pixel-to-pixel correlation between the input- and output image.

The first half of the network is considered the *encoder* part of the network. The encoder translates the image into a set of semantically strong low-level features. Learning semantically strong features may correspond to specific lines or textures in the image. The second half, called the *decoder* part of the network, extracts the low-level features to high-resolution feature maps without losing the semantically strong features. The decoder consists of concatenating the features from the skip connections at every third block and upsampling the features by, e.g., transposed convolution (as included in the original U-Net paper) or bilinear upsampling.

A collection of layers at each height level in Figure 2.9 represents a block. The first four blocks in the original U-Net architecture have convolutional layers that consist of a filter size of $[64, 128, 258, 512]$ kernels, respectively. As more pooling layers are applied in the downsampling part, the purpose of the increasing filter size is to increase the capability of learning semantically strong features, each with a large receptive field. However, in the upsampling part, the blocks have a filter size in the exact opposite order as the downsampling blocks.

Each convolutional layer in the U-Net architecture is followed by a ReLU activation layer to avoid the problem of vanishing gradients in deep CNNs.

This encoder-decoder architecture is common in semantic segmentation problems. Other CNN architectures can be used as an encoder in a segmentation problem as well, such as AlexNet, ResNet or a VGG network.

2.6.3 Squeeze-and-Excitation Network

The squeeze-and-excitation (SE) Network introduces the SE block [56], which is included in several modern CNN architectures [65, 66]. The SE block combines the information across channels of the feature map X to recalibrate the channels of feature maps by dependencies across channels. Figure 2.10 represents the computations of a SE block.

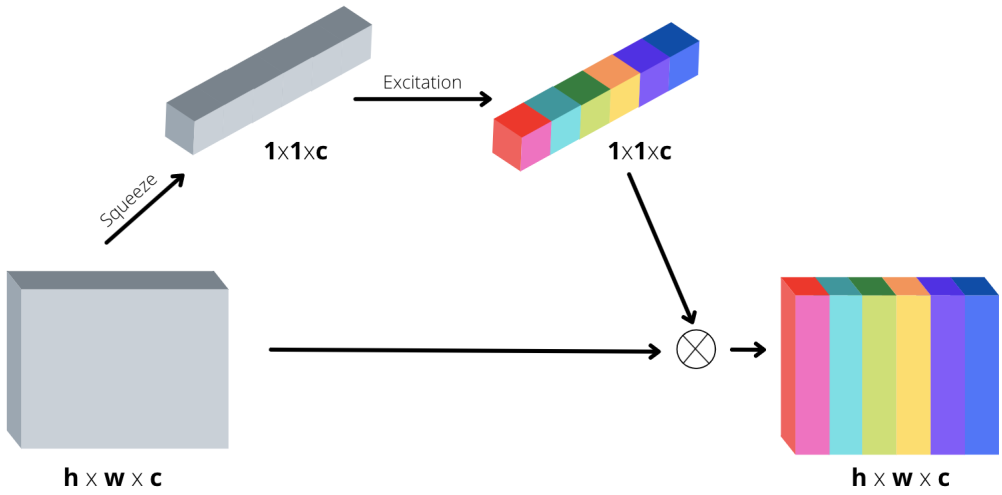

Figure 2.10: The SE module of the SE network. The left gray box represents the original feature map, the second gray box represents the output from the squeeze module, the leftmost multi-color squeezed box represents the per-channel weights from the excitation module, and the rightmost multi-color box represents the original feature map multiplied with the per-channel weights.

The *squeeze* part refers to the use of an adaptive average pooling module to downscale the feature maps at each channel to 1×1, whereas the *excitation* part refers to applying a fully connected linear layer to reduce the number of channels by a scaling factor r, a ReLU activation layer, followed by a fully connected layer that maps the number of channels back to its original number, and lastly a Sigmoid activation layer. The authors of [56] argue that the Sigmoid layer allows multiple channels to be emphasized at once. Finally, the channel weights of the excitation are multiplied elementwise (represented by $\otimes$ in Figure 2.10) to the original input feature X.

The squeeze module collects the global information of each channel, whereas the excitation module obtains the per-channel weights.

2.6.4 DeepLabV2

DeepLabV1 [12] is a CNN specifically designed to perform semantic segmentation and addresses the problem of resolution loss of feature maps in Fully Convolutional Neural Networks (FCNs) by introducing Atrous convolutions (or dilated convolutions), whereas DeepLabV2 [11] introduces atrous spatial pyramid pooling (ASPP) modules to DeepLabV1. The ASPP block is added to the final feature map of the network to recognize features of different scales. Figure 2.11 represents the computations in a ASPP module.

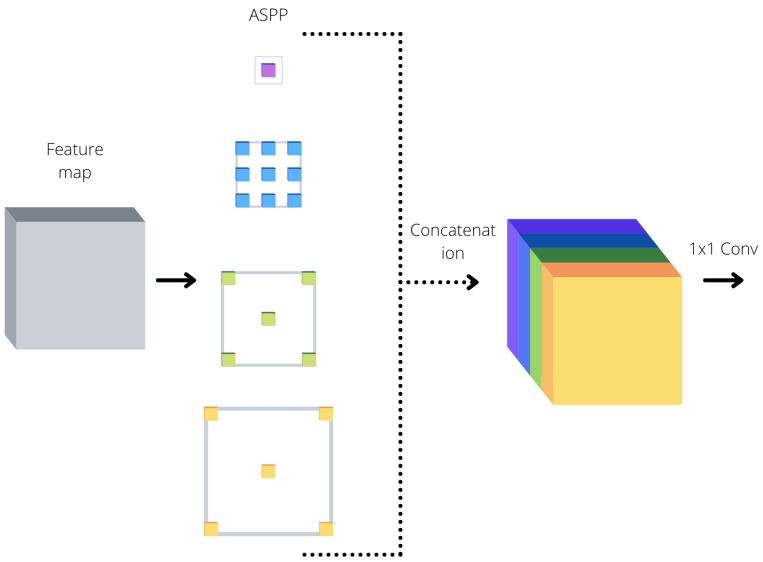

Figure 2.11: This figure illustrates an example of an ASPP module with four blocks containing an atrous convolutional layer with different kernel sizes and dilation rates. Each feature map is concatenated followed by a 1×1 convolution.

The ASPP module consists of several atrous convolution layers, each with different dilation rates, used on the same feature map. Each atrous convolutional layer is followed by a ReLU and batch-norm layer. Finally, the outputs of the ASPP module is concatenated along the channel dimension, followed by a 1×1 convolution to reduce the channel size.

2.6.5 Attention

The intention of using an attention module in a CNN is to focus more on the important features and suppress the unnecessary information [141]. The attention mechanism is in mathematical terms a weighted sum of feature maps. The attention module typically consists of a batch-norm layer, an activation function and a 2D (1×1) convolutional layer. It takes a $C \times H \times W$ feature map and outputs a $1 \times H \times W$ attention map. The attention map is multiplied with the original feature map elementwise. This product is argued to increase representation power by learning which features to emphasize or to suppress [141].

2.6.6 Segmentation Models

U-Net and ResNet (used as an encoder) are especially popular in semantic segmentation. Semantic segmentation aims at dividing the entire image into different semantic classes. In many applications using CNNs, one

typically uses fully connected linear layers at the end of the CNN in order to output a vector of class probabilities. However, in segmentation problems, one instead keeps a fully convolutional neural network (FCN) architecture to output an entire image. In contrast, the classification is performed per-pixel, making the predictions represent a tensor of probability scores at each pixel location.

In binary segmentation problems, one is interested in discriminating between a class of interest and a background class. Hence, the probability scores must be mapped to the colormap representing the class value by thresholding them (typically set by a value of 0.5) to values of 0 or 1, where 1 traditionally represents the class of interest and 0 represents the background class.

2.7 Metrics

A loss function is a requirement for computing gradients and updating the model during the backpropagation. This is a metric that quantifies the quality of a model output relative to the ground truth.

In binary semantic segmentation, one is interested in classifying all the image pixels into either the class of interest or the background class. Given this, the following metrics are formulated in terms of binary image segmentation.

The following subsections provide detailed descriptions of common metrics used as loss functions and scoring metrics for validating the performance of DL models in computer vision. More specifically, three of the main metrics used in this thesis are presented, including their limitations.

2.7.1 Binary Cross-Entropy

BCE is a scoring metric that measures the difference between two probability distributions for a set of events [62]. It is used as a loss function for binary classification problems or in multi-label classification problems. In mathematical terms, the BCE is defined as

$$\text{BCE}(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})), \tag{2.17}$$

where y is a single ground truth variable, and $\hat{y}$ represents a predicted value of a predictive model. The elements in the predicted image $\hat{y}$ are activated by the Sigmoid activation function before applying BCE, as each pixel represents the probability of belonging to the background pixel.

In many cases, it is common for one or multiple classes to be over-represented in terms of the number of instances. Since BCE evaluates individual pixels, large objects (areas of pixels belonging to a specific class) contribute more compared to the smaller ones. To overcome this problem of class imbalance, one can use the weighted version of BCE where the weight proportional to the under-represented class is larger relative to the over-represented class. This is mathematically represented by

$$\text{W-BCE}(y, \hat{y}) = -(\rho \cdot y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})), \tag{2.18}$$

where ρ represents the weight value for the positive class.

2.7.2 Dice Similarity Coefficient

The DSC, also known as the F1 score or the Sørensen-Dice, is a commonly used metric in computer vision tasks. The DSC is known as easy to interpret and implement [113], which measures the spatial overlap between two samples. It is popularly used in medical image segmentation [150] and computer vision. Although first introduced in 1945 [23] as a scoring metric, it was not until 2016 it was used as a loss function for training segmentation models [129]. The DSC is defined as

$$\text{DSC}(y, \hat{y}) = \frac{2 \cdot TP}{(2 \cdot TP) + FP + FN}, \tag{2.19}$$

where true positive (TP) represents number of correctly predictive pixels. FP is the number of wrongly predicted positive pixels, and FN is the number of wrongly predicted negative pixels.

The objective of semantic segmentation is to maximize Equation 2.19, which represents the degree of overlap. This score ranges from 0, representing no overlap, to 1, meaning perfect overlap between the ground truth segmentation mask y and the predicted segmentation mask $\hat{y}$. Figure 2.12 represents an illustrative figure of what the DSC measures.

Most segmentation problems contain class imbalances, where the number of background pixels is typically dominating. In such cases, one can obtain a large score by only predicting the background class. The DSC addresses this class imbalance problem by not taking the background class into account.

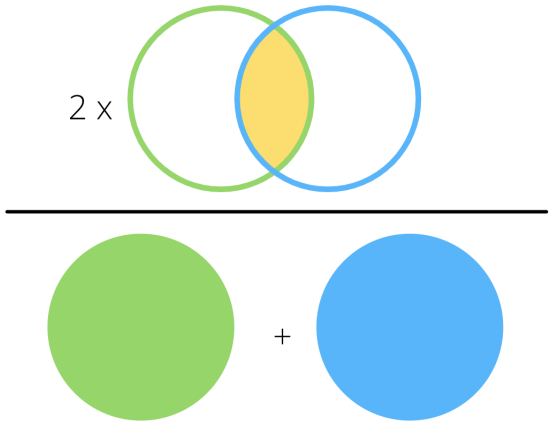

Figure 2.12: This figure represents what the DSC computes with two masks (green and blue circles) as input. The yellow area in the numerator represents the area where the masks overlap, also know as the the TPs.

DSC measures an overlap, hence its value becomes independent of how large the area of the TP pixels is. Because of this, it holds the same value independent of how large or small the ground truth area is, and the DSC value becomes the same. This is also true for all overlap-based computer vision metrics.

2.7.3 Intersection Over Union

The intersection over union (IoU), also known as the Jaccard index or the Similarity coefficient, is a metric commonly used in computer vision tasks [60]. The IoU is very similar to the DSC, but is in contrast mostly used as a scoring metric. The IoU measures the degree of overlap between two distributions. Mathematically, it is defined as

$$\text{IoU}(y, \hat{y}) = \frac{TP}{TP + FP + FN}. \tag{2.20}$$

In Equation 2.20, the IoU ranges from 0 to 1, like the DSC in Equation 2.19. The 1 represents a perfect overlap between the two distributions.

The IoU measures the ratio between the shared area of sample $\hat{y}$ and y, and the areas which are distinct between the two samples. Figure 2.13 gives a visual illustration of what the IoU measures in terms of segmentation masks.

IoU adapted as a loss function is referred to as the *Jaccard distance*. Because one typically wants to minimize the loss function, the Jaccard distance is thus formulated in terms of the dissimilarity between the prediction and ground truth. Since IoU measures the similarity, the dissimilarity is measured by subtracting the IoU from 1. The same argument also yields for DSC when using it as a loss function.

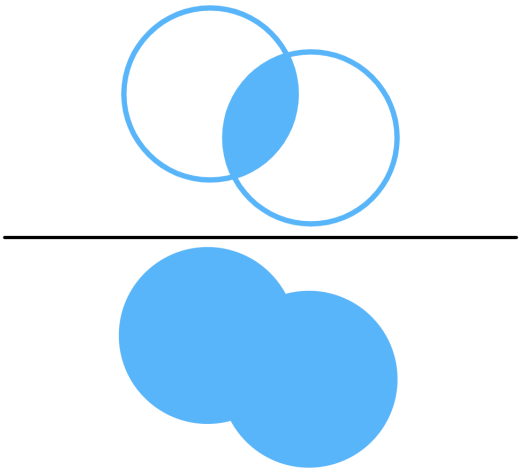

Figure 2.13: The blue circles represent two different input masks. The figure in the numerator represents the area where the masks overlap, and the denominator represents the area of union between the two masks.

2.7.4 Limitations of Metrics in Segmentation Problems

Determining a metric for model evaluation and training is an essential step in segmentation problems. Some metrics are known to be misleading in these cases. For example, naively adapting the accuracy metric for binary segmentation, can sometimes be maximizes whilst completely ignoring the class of interest. This section describes three cases demonstrating the limitations of the DSC and IoU.

The DSC and the IoU are not considered to be suitable metrics for segmentation problems where the ground truth segmentation masks can be small relative to the one pixel. Suppose that for a large (i.e., containing hundreds of pixels) ground truth segmentation mask, the corresponding two predictions that only differ 1 pixel from the truth will have an approximately similar DSC or IoU score. In the case of a small ground truth mask containing only 4 positive pixels as shown in Figure 2.14, suppose that two similar segmentation predictions are made. With only 1 false negative pixel that differentiates the two predictions results in a difference in DSC and IoU of 78% and 67%, respectively. A metric where only a few pixels have such a large influence is considered an undesirable property [113].

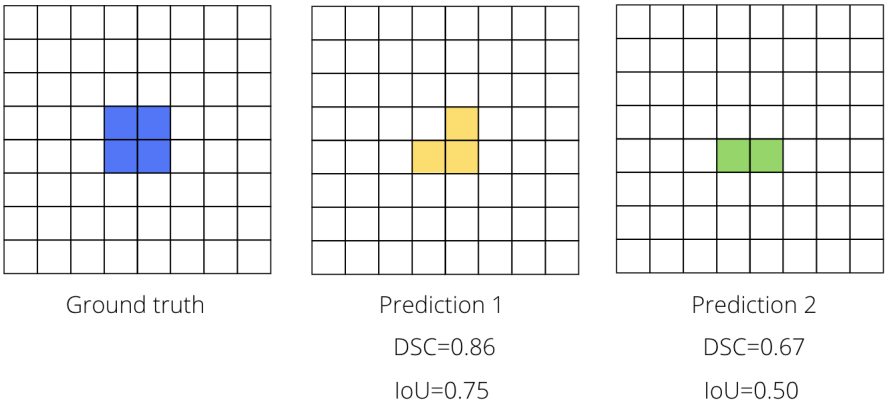

Figure 2.14: The figure illustrates the limitations of DSC and IoU in segmentation problems.

The DSC and IoU are also known to be susceptible to noise. Noise is represented by a single false positive pixel far outside the predicted segmentation mask. In fact, wrongly predicting a single positive pixel can have the same effect as nearly missing an entire object. Hence, these metrics are dependent on the entire batch and have a large variance. If the entire dataset size is divisible by the batch size, one can avoid per-batch scores of high variances.

DSC and IoU are also not able to measure the difference in shapes of segmentation masks. This means that for multiple different predictions of a ground truth segmentation mask containing different shapes, the same resulting DSC or IoU is shared between the shapes [113].

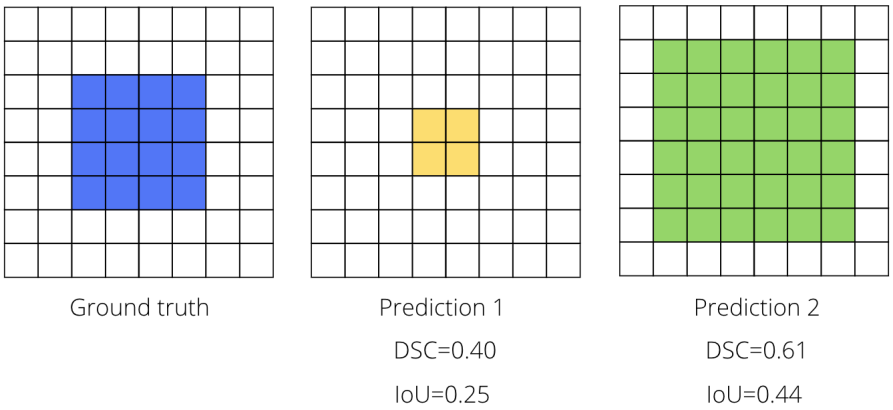

Figure 2.15: The figure represents how DSC and IoU are not able to equally measure over- and under-segmentation in segmentation problems.

The DSC and IoU are not able to equally measure over- and under-segmentation of the same shape, as illustrated in Figure 2.15. Over-segmentation is where the predicted segmentation mask has the max-

imum number of true positives but contains too many false positives. Under-segmentation is when the predicted segmentation mask only has too few true positives relative to the number of false negatives [113].

In summary, the IoU and DSC are know to be the most used overlap segmentation metrics. Despite this, they share a common set of limitations.

2.8 Ethical Considerations in Artificial Intelligence

The increasing awareness and urgency for explainability in AI are because it has become more common for AI technologies to affect our day-to-day-lives on an increasingly larger scale. The need for efficiency and effective reasoning is one consequence of the dominance of AI. Although researchers understand the underlying mathematical principles of such models, it is difficult to examine their inner function responsible for their impressive results. The "black box" is the current image of AI research today because the models consist of complicated NNs which researchers are unable to interpret.

Given the impressive achievements of detection and diagnosis by AI models in research, indicating that they will have a positive impact in medicine. Despite this, real-world deployment of DL models in clinical workflows are rare, and their "black box"-nature is responsible for many of the existing key challenges for delivering clinical impact with AI.

Datasets can contain systematic racial bias which ultimately affects the classifiers trained on these datasets [2, 17, 97]. This has been seen in a CNN trained to detect melanoma in images of skin lesions. The resulting model underperformed on images of skin lesions on people of color because the training data consisted of predominantly fair skinned patients [45]. Cases such as these show how trained AI models can obtain a potential risk of unintended bias towards minorities due to bias contained within in the datasets. Insight into the AI model can help to ensure that the predictions are unbiased to prevent the existing patterns of exclusion and inequality in society. An interpretable model which provides such insights can ensure fairness by preventing possible discrimination against underrepresented groups [26, 92]. Hence, if such "black box"-approaches are to be used in the medical domain, they need to be used with knowledge and responsibility.

AI models' lack of interpretability, explainability, and transparency make them untrustworthy in medical applications where the risks of decision-

making tasks are high. AI models have in many cases seen to outperform or even be on par with experienced physicians [45, 71, 91, 101, 109, 147, 148]. The potential of what they can offer combined with the high risks of decision-making tasks in the medical domain is one of the strongest arguments in favor of explainability. Explainability can be used as a tool to increase accountability of AI models. In this context, accountability is defined as the ability to determine whether a decision was made in line with procedural and substantive standards. Accountability is therefore making someone responsible if those standards are not met [27].

Explainability is used to reveal the reasoning behind individual decisions of AI models, which in addition can prevent and rectify errors of predictions. Because explainability can help to increase the accountability of model predictions, research in XAI will not only help to facilitate the implementation of AI in the medical domain but is also able to establish and increase trust. Therefore, XAI generally offers enormous opportunities for the medical domain [55]. For example, it can help to identify systematic errors and detect results based on wrong assumptions [74, 75]. Hence, the explanations and interpretations can help to facilitate causality. Some of the explainability methods in the next section reveal such errors, and they are some of the most popular methods of explainability used today.

2.9 Explainable Artificial Intelligence

XAI is the research field that aims to obtain an understanding of how an AI model arrives at a decision. Most DL models lack interpretability due to their non-linear mappings that process the image from raw pixels to feature representations, and finally to the final classification. This "black box" nature of DL models is therefore a considerable drawback when considering their integration in safety-critical applications, because it hinders the human experts from being included in their decision. XAI is needed to establish fairness, causality and trust between the user and the model [26].

Explainability is highly relevant for DL applications such as in medical diagnostics, because the cost of making wrong predictions can determine the difference in human life and health. Hence, an explainable medical diagnosis system is therefore needed to gain trust of both physicians and patients.

In the medical domain, obtaining a correct prediction with inappropriate reasoning can be dangerous. The AI model should try to extract and review the same features as the medical experts to make the correct associations between the disease and its features. With this considered, the model can be able to support medical experts in making a comprehensive

assessment of the treatment at hand. Therefore, building systems that can visualize model decisions, model bias and uncertainty of the prediction model is of utmost importance to ensure safe decision making and facilitate trust between DL models and physicians.

The methods described in the following subsections address the "black box" problem of DL models and provides a short overview of the different well-known methods of XAI.

2.9.1 Feature Visualization

XAI methods can be as simple as visualizing what a CNN really learns. This method is known as feature visualization [29, 94, 144]. One of example of this is depicted in Figure 2.16.

Feature visualizations allow the end-user to observe how the CNN model learns unique representations based on a set of images, giving insight into the hidden layers. This technique visualizes the activation of a specific layer as an input image traverses the model. Erhan et al. [29] argue that these visualizations can be used to compare learned features across different models tested on the same dataset to understand the test error.

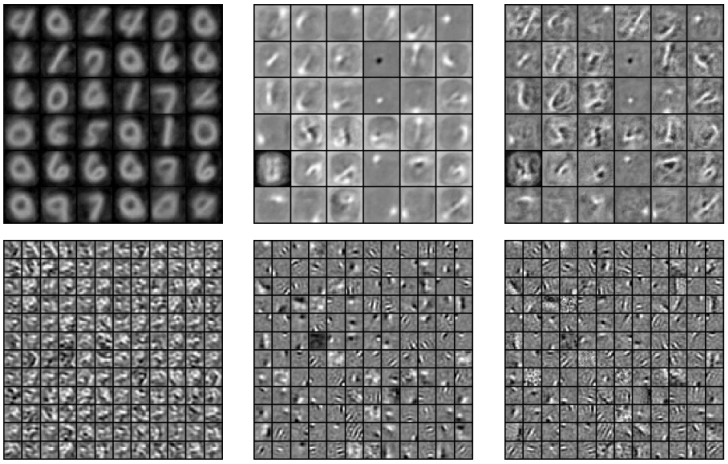

Figure 2.16: Results from [29] showing feature visualization of multiple hidden units in a hidden layer of a NN trained on images containing handwritten letters.

Visualizing features is an optimization problem, and by doing so one must find the input that maximizes the activation-specific trained feature map of a CNN. Given this, one chooses an image to optimize, uses it as input to a trained CNN, creates a loss based on the negative of the model output and the feature map of interest, and finally adds L2 regularization and total variation loss. The input image is thus iteratively optimized using GD

to maximize the effect of a feature map.

In practice, one finds the maximum activation by searching through a set of training images, then selects the set that maximizes the activation. The methods introduced in the sections below have been shown to provide insights during model training, and one method shows how XAI approaches can reveal systematic biases of a specific class in a classification task.

2.9.2 Grad-CAM

Feature attribution methods are an additional concept of XAI. One such gradient-based approach called *Grad-CAM* (Gradient-weighted class activation mapping) has become popular over the last few years [121]. This is part of XAI because feature attributions explain how much each feature contributed to a prediction of a specific class. Grad-CAM works by choosing a convolutional layer in a CNN and then examining the gradient information flowing through that layer.

This technique uses the gradients of a classification score with respect to the feature map activations of a convolutional layer. The gradients are then global average-pooled to obtain the per-neuron importance weights. The authors of Grad-CAM argue that the weights capture the important values of a specific feature map for a specific class. The original paper applies Grad-CAM to the last fully connected layer because it has the best compromise between high-level semantics and spatial information. Despite this, the technique can in principle be applied to any layer in a CNN. The result can be visualized as a heatmap and provided to a human observer for verification and validation in order to gain insights into what the model has learned.

Class activation maps allow the user to verify if a specific part of the input image may have "confused" the network, causing it to make a wrong prediction. Recent adaptations of Grad-CAM have shown state-of-the-art performance in DL-based classification and detection on medical data.

Hicks et al. [52] show how Grad-CAM can provide explainability in a CNN classifying sex from electrocardiograms (ECGs). Their results show how the trained network analyzes the ECG like a trained cardiologist. Detecting sex from ECGs alone is known to be a near impossible task for human experts, but their results show that they were successful with an 89% accuracy. Some of their results are shown in Figure 2.17

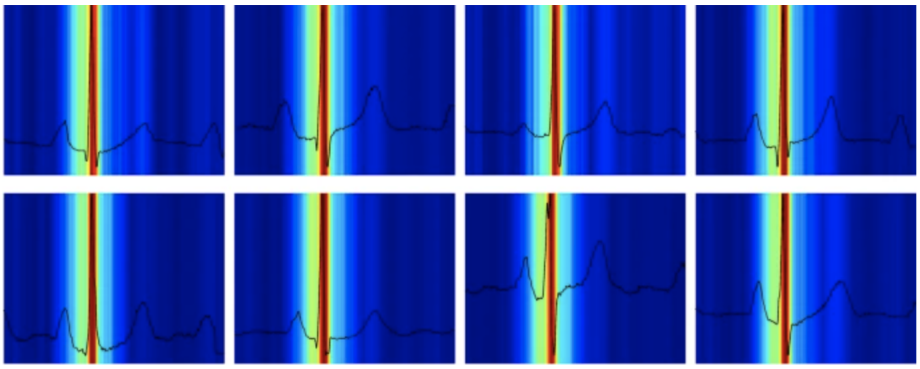

Figure 2.17: This figure shows one example of the use of Grad-CAMs in ECGs [52].

Panwar et al. [102] propose a Grad-CAM approach in lung X-ray images. They adapted a transfer-learning approach on binary classification between Covid-19 and non-Covid-19 positive patients, obtaining an accuracy of approximately 96%. Non-Covid-19 positive patients had either Pneumonia or other pulmonary diseases. The authors utilized Grad-CAM to make heatmap representations on the X-ray's. Their results revealed that the patients diagnosed with Pneumonia had greater chances of being classified as a false positive by the DL algorithm.

One non-CNN approach, such as the one provided by He et al. [49], proposes a feed-forward NN architecture to classify post-operative complications after lung cancer surgery. Their training data were based on tabular patient data consisting of 36 features. They included the use of Grad-CAM to perform feature selection such that they were able to rank the features according to their importance.

2.9.3 Layer-Wise Relevance Propagation

One popular feature attribution method is by using Layer-Wise Relevance Propagation (LRP) [74]. By propagating the model output backwards through the model, one can find the relevance score (attribution) for each layer. Each relevance score for each input pixel can be mapped to a colormap, and hence be used to visualize the feature attribution in terms of a heatmap.

One groundbreaking reveal provided by the LRP method showed that the winners of the PASCAL Visual Object Classes (VOC) Challenge [31] were able to predict the class "horse" because these horse images contained a copyright watermark [75]. Hence, the model was trained based on the wrong association, and removal of the watermark in the horse images resulted in misclassification. This result demonstrates how a model can

make predictions based on the incorrect associations. Models that learn these wrong associations are commonly referred to as "Clever Hans" predictors, which are named after the horse that was able to count [118]. It was however later revealed that the horse was in fact not able to perform these calculations, but gave this impression based off the questioners' reactions.

2.10 Uncertainty Estimation

In applications where DL models give high-stake predictions, one requires the model to not only give accurate predictions, but also give some information about the quality of the prediction. For prediction models, the quality of prediction corresponds to their predictive uncertainty estimates. Uncertainty can be used to assess the undesirable behavior of a trained model. For segmentation problems, the pixel-wise uncertainty estimates can help at verifying the pixels that were omitted during thresholding, and allow for human experts to review the predictions and the model itself to prevent potential silent failures by a DL model.

The two main approaches demonstrate how to estimate predictive uncertainties of deep NNs by ensembling. Before going into these methods, the idea behind ensembling will be introduced.

2.10.1 Ensembling

The main idea behind ensembling is to aggregate the strength of independent models to obtain a collective higher score than that of a single baseline model. The strength of each model is also believed to complement one another, where some weaknesses will cancel out. This intuition is also given in the Jury theorem. This proves that, under certain assumptions, the increasing the number of voters increases the probability of arriving with the correct verdict compared to that of a single voter [30]. This behavior is also demonstrated in the *Wisdom of the Crowd* effect [131], which relies on the ensembles' ability to filter out the noise of individual voters to get closer to the ground truth.

Ensembling in AI today can be summarized into two main steps. The first is to train different models, and the second is to combine their predictions. What differs the models can be the model architecture, the use of different training data, different training regimes, different initialization schemes, etc. One can combine the predictions by averaging them, or by custom voting rules. The two famous ensembling techniques are *boosting* [34] and *random forest* [7].

Ensemble methods have often been shown to outperform any single classifier [24], and are seen to win large DL competitions [136].

2.10.2 Monte Carlo Dropout

A paper by Yarin Gal et al. [35] from 2016 added another reason to use dropout other than the benefit of reducing overfitting of NNs [127]. This paper showed that training a deep NN with dropout layers is mathematically equivalent to Approximate Bayesian inference in deep Gaussian processes [39, p. 360]. Given this, they introduce a technique called MC dropout to estimate the predictive uncertainty of a deep NN using dropout during inference time. This method can also increase model performance. The following algorithm describes the procedure of MC dropout:

1. Train a base model (deep NN) containing dropout layers.
2. Predict on a test set using the trained model with dropout layers turned on.
3. Repeat *step 2*, N times.
4. Compute the mean and variance of the N predictions.

For a classification problem, the mean predictions are equivalent to the mean probability provided by the N models. The variance of the N probabilities (or predictions) represents the estimated predictive uncertainty of the baseline model.

The use of MC dropout during inference time can be interpreted as an ensemble of different models, with the different dropout layers that differentiate them. Figure 2.18 represents the pipeline of collecting a mean prediction with an ensemble of dropout models, where the dotted frame contains the ensemble of MC dropout models. Using MC dropout empower the *wisdom of the crowd*-phenomenon without the necessity of training multiple deep NNs to construct the ensemble.

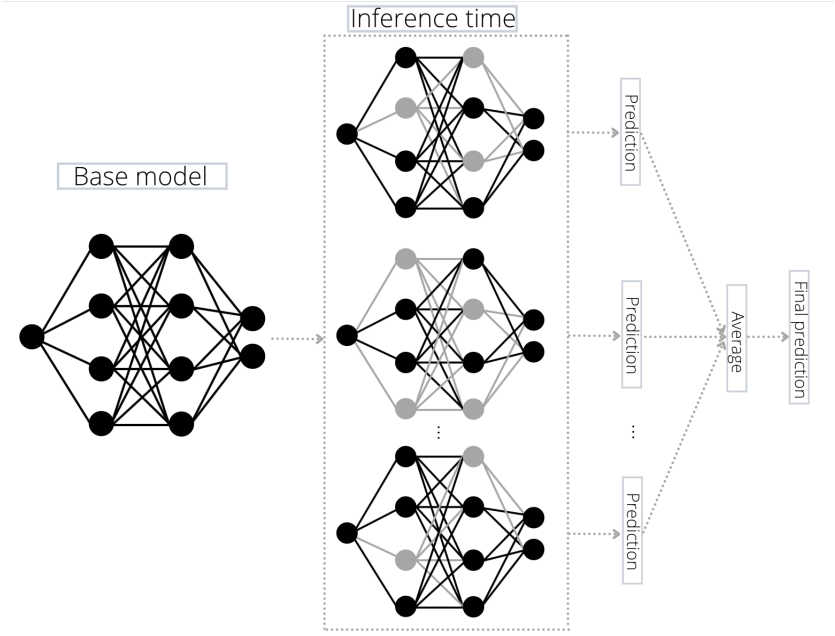

Figure 2.18: This diagram represents an illustration of the MC dropout pipeline. The networks inside the dotted rectangle represent the ensemble of the trained baseline model where some neurons are dropped out during inference time.

2.10.3 Deep Ensembles

Lakshminarayanan et al. [73] propose an ensembling approach to extract the estimated predictive uncertainty called a *deep ensemble*. This method is an alternative method to estimating the predicted uncertainties of an ensemble of deep NNs. Fort et al. [32] show how ensembles perform well on out-of-distribution samples such as for example adversarial examples [43]. This is because their random parameter initialization enforces different loss trajectories during training of each model in the ensemble, where the results are diverse predictions. This is further elaborated and discussed in Chapter 5.

An example illustration of building a deep ensemble is depicted in Figure 2.19. Note that this pipeline does not include preprocessing steps, data augmentation or adversarial training that can be additionally added to the pipeline. The models in the deep ensemble have equal architecture and are trained separately on the same training data.

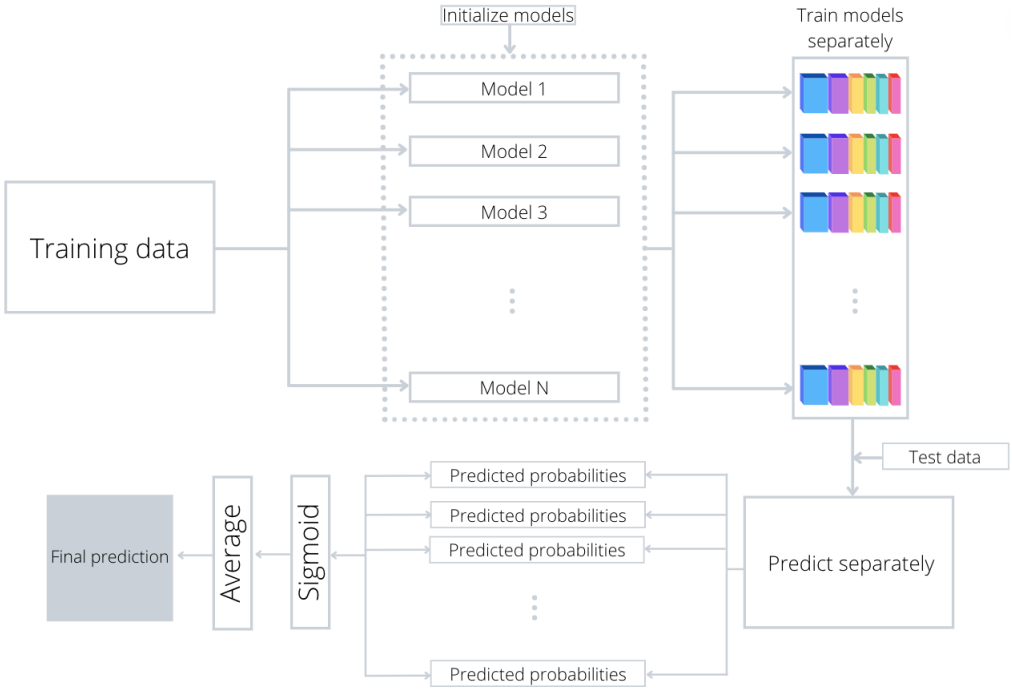

Figure 2.19: This diagram represents the deep ensemble pipeline, which consists of an ensemble size of N models.

One method to ensure calibration of the models in the ensemble is to use a proper scoring rule as a loss function during training [73]. Cross-entropy loss is defined as a strictly proper scoring rule, but the DSC is not [42]. Additionally, one must define the model architecture of the base model and determine the ensemble size, which is the number of models (deep NNs) in the ensemble.

Each individual model is initialized and trained independently, with the same training dataset and the same set of hyperparameters during training. Furthermore, each trained model is used for independent prediction of the test dataset. The predictions are converted to probabilities with a Sigmoid activation function (for a binary problem), and the average probabilities of each of the models in the deep ensemble are computed to construct the final output. For semantic segmentation, the output after averaging is thresholded to discriminate between the two classes. The variance of the predicted probabilities of each model represents the estimated predictive uncertainties.

Lakshminarayanan et al. [73] conclude that the recommended ensemble size is 5, because it dramatically reduces the test loss compared to a single baseline model. For cases where the ensemble size is larger than 5, the score will reduce but fluctuate somewhat. They also argue that adversarial training can help reduce the loss on some datasets, but only for some

metrics, and is therefore not strictly necessary in all cases.

2.11 Summary

This chapter contains medical background on CRC, colonoscopy, CAD in medicine, including an overview of popular colonoscopy datasets used today. In addition, we dive deep into DL tools and describe important concepts of XAI, such as ethics and popular explainability methods used today.

The first three sections in this chapter have a focus on the medical background. Section 2.1 gives a medical overview of CRC and its screening methods. The next section, Section 2.2, contains an overview of CAD in recent history and today. Section 2.3 contains information of three commonly used colonoscopy datasets containing polyps and pixel-wise annotations of these.

The four next sections, sections 2.4, 2.5, 2.6 and 2.7, introduce theory on NNs, CNNs, modern CNNs, and common metrics used in computer vision. These four components assemble the toolbox for all the experiments conducted in this thesis.

The last part in this chapter largely focuses on XAI. Section 2.8 gives the reader an overview of the ethical considerations of the lack of explainability in DL and the importance of XAI. Section 2.9 introduces common explainability methods used today in the context of various use-cases. The last section on XAI, Section 2.10, introduces two important research papers as their research methods introduced in these papers were adapted in this thesis for further development.

The purpose behind this chapter is for the reader to obtain necessary prior knowledge and therefore be able to understand the importance of DL-based CAD in medicine. In addition, the reader obtains a general overview of how explainable methods can determine the future of DL-based CAD in clinical workflows. With this extensive background, the reader can fully understand the experimental methods which are described in the next chapter.

Chapter 3

Methods

Detecting the cause of an error when building a large project or framework can become extremely difficult if everything works up until a certain point. Meeting obstacles along the road requires that one must try to detect the error by performing a large amount of many small inspections along the pipeline. These inspections include reporting the sub-products along the pipeline, which can for example correspond to performance metrics, tensor shapes, paths, iteration number, etc., which are reported back to the programmer. Sometimes, one can also take mathematical operations for granted by, for example, mixing the order of operations that is mathematically in-equivalent to the intended operation. These mistakes can be hard to detect but may be more obvious to a researcher that has a perspicuous understanding of what the results should look like.

This chapter describes the experimental setup for producing the results presented in this thesis. Describing these methods aims to connect the theory presented in Chapter 2 with the results which are presented later in Chapter 4. Appendix A contains information on how all source code related to this thesis can be found.

3.1 Resources

This section contains the details and descriptions of all the supporting tools used to conduct the experiments introduced in this thesis. These tools include the software, hardware and dataset used. We also provide explanations as to why one framework was preferred instead of the other and try to describe the reasoning behind the choices we made.

3.1.1 The Dataset

The dataset used in this thesis is Kvasir-SEG [106]. We used this for training, validating and testing each DL model. This dataset underwent

the same experiments, including preprocessing steps and dataset splits for each training regimen corresponding to a CNN. Due to time restrictions, we excluded the use of other annotated polyp datasets, such as CVC-ClinicDB [4] and ETIS-Larib Polyp DB [125] described in Section 2.3.

3.1.2 Software

This section describes the software specifications and frameworks that were used to produce the results in this thesis. All the hardware was provided by the Simula Research Laboratory.

The programming language Python 3.7.3[1] has been used to conduct all experiments. Python is a general-purpose programming language that enables fast implementation of various tasks. It is open-source, which allows for continuous independent development of many popular frameworks. One can easily organize, preprocess and visualize data which makes Python a natural choice for many researches. Combined with the open-source machine learning libraries such as PyTorch [104], Keras[2] and TensorFlow [1], Python becomes a powerful tool and thus a natural choice for the experiments conducted in this thesis.

We use PyTorch [104] as a supporting framework for implementing CNN architectures and metrics, preprocessing steps of images, constructing iterable data loaders, building training- and evaluation pipelines, and data visualization. PyTorch is an open-source DL tensor framework which supports computational graphs to easily move tensors between the central processing unit (CPU) and the GPU. PyTorch also supports automatic differentiation as explained in Section 2.4.3.

Other researchers may prefer Keras combined with TensorFlow for DL experiments, but in contrast to PyTorch, they do not utilize dynamic computational graphs. Instead, they utilize static graphs, which require that all graph nodes, i.e., variables such as loss function and other parameters must be defined at the start of each execution. The graph is rebuilt and reused for each session. Not only can this be inconvenient for other applications, such as in NLP, but also when debugging. For example, debugging a NN implementation without defining the loss function, which is often the use-case for many researchers, can be difficult in the Keras and Tensorflow environment.

The computational graphs are built dynamically in PyTorch, which allows for flexible debugging and modification of the internals of the graph any-

[1]Full Python documentation can be found on https://docs.python.org/3/
[2]Keras documentation can be found on https://keras.io/

time during development. One drawback is that the graph is rebuilt after each iteration, meaning that each execution takes more time as compared to static graphs. Since implementation of large NNs was a major topic in this thesis, PyTorch allows for more efficient debugging due to its use of dynamic graphs and was thus a natural choice for conducting the experiments presented in this thesis.

3.1.3 Hardware

All experiments have been performed on the *eX3 computing cluster* established by the Simula Research Laboratory. More specifically, all models were trained on an NVIDIA Tesla V100 GPU[3] with 16 GB random access memory (RAM). The eX3 infrastructure has four Intel Xeon Gold 6130 processors. Each of these high-performance processors is a 64-bit 16-core x86 multi-socket server microprocessor.

3.1.4 Memory Constraints

Each model was trained on a GPU that had a RAM of 32 GB. With this considered, a set of measures had to be taken to train the deep and highly memory-consuming CNNs. First, the input images were resized to 256×256 to reserve memory for training (compute the gradients) the DL model. Second, a batch size of 32 was used to relieve memory. Third and lastly, we trained the model with mixed precision on the GPU, which is a memory-efficient computation scheme described in Section 3.3.4.

3.2 Preprocessing

This section contains the details on all preprocessing steps of the images before they were used in model training, validating and testing. Note that we did not include any postprocessing steps, as they are commonly included in DL competitions to further increase performance.

3.2.1 Normalization

Image transformations were used as a preprocessing technique to adjust the original pixel intensities of an image. The original input images contain RBG color channels where the pixels of each channel can have different intensity values ranging from 0 to 255. Normalization is a method for re-scaling intensity values to a predefined range. Common normalization techniques in image processing include re-scaling or clipping.

[3]More information about the NVIDIA Tesla V100 can be found on https://www.nvidia.com/en-us/data-center/v100/

Data standardization is one approach to normalizing an image. We used this technique to each input image to adjust the intensity values of each color channel to lie in the range $\in [0, 1]$. Normalizing the color channels is done by subtracting the mean and dividing the by standard deviation for each pixel using the statistics based on a specific color channel. Standardization is useful when combining features of different magnitude across different dimensions. If we do not scale the input images, the magnitude of the learned feature maps can vary for each feature map. Therefore, during optimization, multiplying the learning rate in the feature maps at different dimensions can significantly differ from another. The result can be the effect of overcompensating a feature in one dimension, whereas we undercompensate a feature in a different dimension. This effect is not ideal for the case of getting stuck in a local minimum because the optimization algorithm has difficulties at centering on a set of weights in the loss surface. With this considered, one can introduce per-weight learning rates to avoid this effect but consequently, introducing more hyperparameters to the training scheme is unfavorable as it makes the task of hyperparameter tuning more complicated as compared to having less hyperparameters.

To avoid this, normalization is deemed as a standard practice in DL [108, 116]. One known issue related to color normalization is that the resulting image can become too unnatural and not resemble the original image. However, for the use of the Kvasir-SEG dataset we argue in favor of color normalization to adjust for the changing lighting conditions as commonly seen in colonoscopy images where a bright light is usually used. An example of these bright flashlight-like spots can be seen in Figure 2.1 and Figure 2.2.

The standard deviation and mean of all color channels were only calculated for the training dataset, which is 80% of the total Kvasir-SEG dataset, i.e., consisting of 800 images. Dataset statistics should not be calculated for datasets that are used for evaluation. The training should be completely independent of the test dataset to evaluate its generalizability. Since the model is trained on the images in a specific scale or representation, the validation and test dataset must also match those representations. Hence, the validation and test data are transformed to PyTorch tensors like the training data. Each element in these tensors are normalized based on the training data statistics for each corresponding color channel. The image pixel values were transformed to tensors using `pytorch.ToTensorV2` from the Albumentations library [8] in Python.

3.2.2 Resizing

As mentioned in Section 3.1.4, each image and its corresponding mask were resized to 256×256 as a part of the preprocessing. We did this to relieve memory during training. To conserve image information, we purposely avoided image cropping. This resizing process was also done to the validation- and test dataset.

3.2.3 Dataset Splitting

We split each of the three datasets into three separate parts as described in Section 2.4.5. We take 80% (or 800 images) of the total dataset into the training set, 10% (or 100 images) into the validation set, and 10% into the test set. This ratio is common in various different applications, which is referred to as the *Pareto principle* [28], stating that 80% of consequences come from 20% of the causes. This ratio is especially common in machine learning because it has empirically contributed in many cases to reduce overfitting and train models that generalize well onto unseen data. Gholamy et al. [40] argues mathematically to why this is the case.

The datasets are split by building a random permutation array. Its indices range from 0 to the total size of Kvasir-SEG minus one. The filenames of the entire dataset are sorted according to the random permutation array by indexing. By doing so, the permutation array represents a random order of indices for the filenames to be sorted by. Once the filenames are randomized by the permutation array, the images and their corresponding masks are put into separate folders. The training-, validation-, and test dataset are assigned its own folder. The purpose behind folder separation is to prevent data leakage, e.g., when some instances that belong to the test dataset get mixed into the training set. Due to folder separation, the split is held constant in each experiment.

The training- and validation data are shuffled before each time they are loaded into a model. During training, the loss surface is fixed for a certain training dataset, where it can contain multiple local minimums and saddle points. In such optimization problems, the optimizer is less likely to escape these points. To mitigate this problem, we use a GD variant using mini-batches combined with shuffling of the batches at each iteration. Hence, we shuffle both the training and validation data for each iteration during training to reduce the possibility of repeatedly being stuck in local minimum.

3.2.4 Data Augmentation

As explained in Section 2.3, the use of large datasets to obtain generalizable NNs is a prerequisite for their success. Data augmentation is a standard technique in computer vision to mimic the effect of large datasets [105]. This is utilized by transforming an image using a random combination of different transformation techniques. The different transformations can for example include translating the image, cropping, flipping, resizing, or using color jitter that randomly transform the brightness, contrast and saturation. As we are limited to $1,000$ images that are within the Kvasir-SEG dataset, we include data augmentation to obtain a model capable of generalizing when performing predictions on other data outside of the training dataset.

Data augmentation is only done for training data. The data augmentation transformations are performed on-the-fly for each time the training data is loaded into the model at the beginning of each forward pass. Considering this, the model can learn a larger variety of features than those provided in the original training set. These additional features, provided with transformations, are introduced with each training iteration.

Each augmentation is assigned with a probability, which represents the likelihood of a transformation being applied to an image. Thus, multiple transformations can be simultaneously applied to the same image. We use the random rotation transformation method with a limit of 35 degrees counterclockwise with a corresponding probability of 10%, a horizontal flip transformation that horizontally flips the image with a probability of 50%, and a vertical flip with a 10% probability. We limited the number of possible transformations to three as we do not intend to distort the images too much such that the augmentations completely destroys the important features that characterize the polyps. The augmentations were implemented using the Albumentations library [8].

3.3 Model Training

To develop a baseline model to construct an ensemble to estimate the predictive uncertainty with the MC dropout- and deep ensemble methods. We develop the model based on the input data described in Section 2.3, the preprocessing steps explained in Section 3.2 including the data augmentation techniques described in Section 3.2.4.

This section contains the descriptions of the training regimen of all baseline models used in the experiments. To explain why we arrived at this training regimen, we introduce the aims and goals of model training. Af-

terwards, we dive further into the details of building the training pipeline. This pipeline consists of model selection, mixed-precision training, hyper-parameter tuning and assessment of the overfitting problem which are described in each of the following sections.

3.3.1 Aims and Goals

The main aims and goals of the training regimen for all the baseline models are the following:

Aim 1 Train a baseline model sufficient for polyp segmentation in colonoscopy images.

Aim 2 Utilize and test the state-of-the-art CNN architectures designed for semantic segmentation to reach **Aim 1**.

All model training and evaluation were conducted on hardware that is generally regarded as consumer-grade.

3.3.2 Model Selection

The first step in constructing a training pipeline is to decide upon a model architecture. This choice will have a direct impact on the segmentation quality. We ultimately decided on only using two different model architectures. This section contains the descriptions of the process behind model selection.

Model depth, which is the number of layers in a NN architecture, was prioritized in the model selection process. Depth increases the amount of filters, meaning that we increase the chances of learning feature-maps that corresponds to the polyp features. This is essential for polyp recognition as it directly affects the number of learned feature maps after training. With this considered, we decided beforehand to prioritize model depth in the model selection process. A memory constraint of 32 GB of RAM during training was also an important consideration, but in this case, we chose to reserve as much memory to model depth because training a deep NN requires a large amount of memory for gradient computation.

The pixels in the input and its corresponding annotation have a one-to-one correspondence. This is because one pixel classified as the polyp in prediction mask must desirably represent the pixel belonging to a polyp in the input image. With this considered, an encoder-decoder structure was not only strictly necessary to upsample the prediction mask back to its input size, but also contain downsampling layers to obtain a large receptive field. Therefore, we consider an encoder-decoder structure as an important requirement to the process of model selection in order to utilize

a model suitable to perform semantic segmentation.

There is no generic way of selecting a model and its hyperparameters. One common procedure is to start with a rough guess based on similar experiences. This can also be based on third-party experiences such as a lecture course, blog post or a research paper of a similar problem. We chose ResUNet++ as one of the two models for this exact reason. This is because the original research paper that proposed this architecture used the same dataset as the one used in this thesis, i.e., Kvasir-SEG. Moreover, ResUNet++ was specifically designed for automatic polyp segmentation and has an encoder-decoder structure. Since ResUNet++ was able to achieve state-of-the-art performance on the same datasets used in this thesis, we used the same set of hyperparameters as in the research paper [66] as a natural starting point for the manual hyperparameter tuning process. We also consider ResUNet++ to meet the requirement of model depth in addition to having an encoder-decoder architecture suitable for semantic segmentation.

ResUNet++ consists of residual blocks (or skip-connections), squeeze and excitation blocks, ASPP, and attention blocks. Figure 3.1 represents a block-diagram of the ResUNet++ architecture, where the ASPP module inside the dotted box represents the bridge that connects the encoder (the top four blocks in Figure 3.1) and the decoder (the bottom four blocks in the same figure). The encoder consists of four encoder blocks and three SE blocks. The SE module is described in Section 2.6.3, and ASPP is described in Section 2.6.4.

The input and output of each encoder block are added to a sum by a skip connection, which is passed to a SE block. The feature map from the last encoder block is passed to a ASPP block which acts as a bridge to combine filters at differently sized receptive fields. The three last encoder blocks contain strided convolutional layers which reduce the feature map size by a factor of 2 [66].

The decoder blocks contain skip connections, which adds the feature map from each encoder block to a concatenation layer (pink rectangles in Figure 3.1) in the decoder block. This is also done for the skip connections to each attention layer in the decoder block. The attention block consists of a miniature version of an encoder-decoder structure. The output feature map from the mini encoder-decoder block is multiplied with the attention map. The attention mechanism is explained in Section 2.6.5.

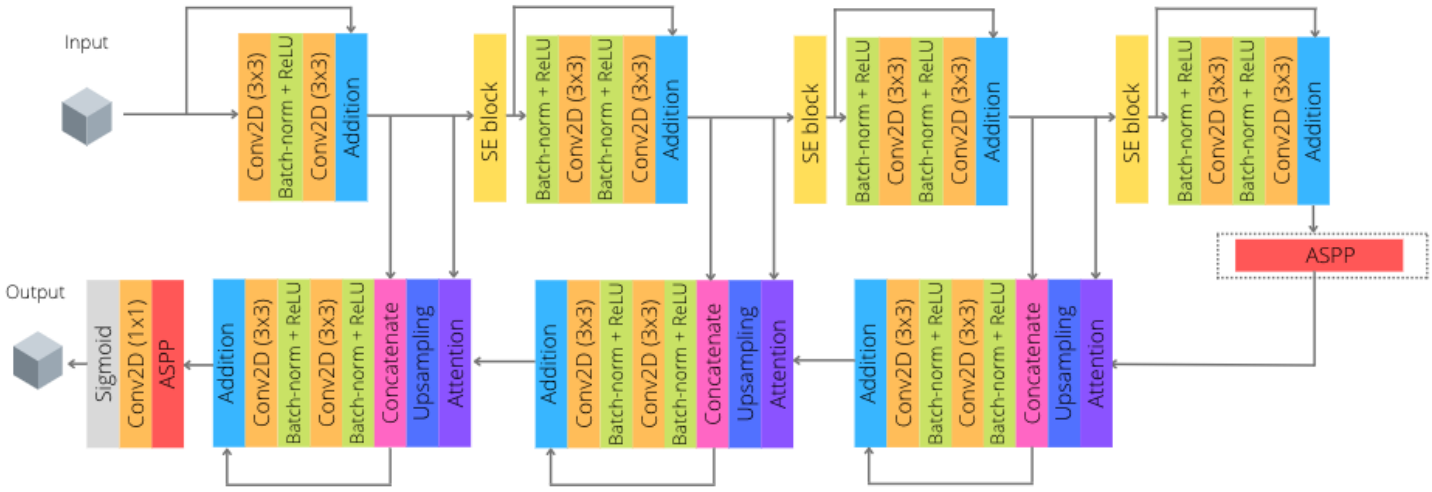

Figure 3.1: The ResUNet++ architecture. Each colored rectangle represents a specific module. Orange represents a two-dimensional convolutional layer, green represents a combination of a batch-norm layer and a ReLU layer, light blue represents an addition operation, pink represents a concatenation of feature maps, purple represents an attention layer, red represents an ASPP module, dark blue represents an upsampling layer, yellow represents a SE block and gray represents a Sigmoid activation. The top modules represents the encoder, the middle ASPP module represents the bridge, and the modules at the bottom represents the decoder.

Each attention map is passed to an upsampling layer. We replace the nearest-neighbor upsampling technique, which is the proposed upsampling mechanism in the original ResUNet++ paper [66], with the bilinear upsampling technique. We argue in favor of using bilinear upsampling because we want to produce a smoother upsampling in exchange for a potentially larger computation time [6, 46].

Skip connections are also located between each decoder block as seen in Figure 3.1. The upsampled feature map is concatenated with the feature map from one encoder block before being passed to a double convolutional block. The feature map from each concatenation layer in a decoder block is added to the output of each decoder block by a skip connection. The final block in the decoder consists of an ASPP module, a 1×1 convolution and a Sigmoid activation layer to construct bounded probabilities in the $[0, 1]$ range.

We also wanted to include a simpler model in contrast to ResUNet++ for comparison in the experiments we conducted. Considering this, we chose U-Net as the second model architecture for experiments of uncertainty estimation. We consider U-Net to meet the requirements of model

depth and having an encoder-decoder structure. We implemented the exact model architecture as proposed in the original U-Net paper [115], but replaced all transposed convolutions with bilinear interpolation layers, which is described in Section 2.5.3. We did so to avoid checkerboard artifacts, which are commonly seen in models containing transposed convolutions. Checkerboard artifact comes from a two-dimensional uneven overlap when using a kernel size that is not divisible by the stride [98]. The uneven overlap intensifies for each time the kernel is multiplied with the pixel, resulting in a checkerboard-like pattern in terms of pixel intensities.

When experimenting with model architectures, we wanted to train a model that was able to obtain a high DSC based on the test data. We consider a high DSC of being a value above 70%, because this is what we interpret to be a model sufficient for generalization, and thus also a trained baseline model sufficient for polyp segmentation as described by the aims and goals in Section 3.3.1. Considering this aim, we experienced that U-Net and ResUNet++ obtained a high score above 70% during model training with the same hyperparameters as in [66]. We were unable to reach a high DSC score during training in some of the experiments. Due to memory and time constraints, we were unable to use extensive hyperparameter tuning techniques in order to achieve a high DSC score during model evaluation. This is not only mentioned later in Section 3.3.5, but also further elaborated when discussing our results in Chapter 5.

The two model architectures also meet **Aim 2** as we defined by the aims and goals prior to model training in Section 3.3.1. They have both shown state-of-the-art performance in various applications [66, 115].

For each CNN architecture implementation, we avoid the direct implementation of the Sigmoid layer directly into the output layer of each model architecture. Instead, we combine the Sigmoid layer to the loss computations. The reason for this will be explained in Section 3.3.5.

3.3.3 Model Implementation

The model architectures were implemented using the `torch.nn` namespace, which provides layers and modules such as dropout-, two-dimensional convolution-, activation- and batch-norm layers. Model architectures and specific modules such as the ASPP block,the SE block and the attention block were implemented as a subclass from the `torch.nn.Module`.[4] Hence, a CNN architecture is implemented as a mod-

[4]See the documentation on this PyTorch base class on https://pytorch.org/docs/stable/generated/torch.nn.Module.html.

ule consisting of other modules. This structure enables the user to define blocks as a module that can be frequently incorporated into the architecture without repeatedly defining its structure each time it is reused. Considering this implementation structure, the process of managing large-scale NN implementations becomes significantly easier.

The model or module architecture is defined inside the `__init__` method and allows us to initialize the attributes of the class. In addition, the forward pass computations must be defined within each module as a method which contains specific computation. The forward pass computations will be executed when the model is called upon as an instance.

3.3.4 Mixed Precision Training

As mentioned in section 3.3.2, we wanted to prioritize model depth when training a model. Due to the memory constraints we described in Section 3.1.4, we needed to save some memory on the GPU for training such a model. Mixed-precision training enables us to do so. PyTorch supports automatic mixed precision training. It is called "mixed precision" because it switches between half- and single-precision, i.e., 16- and 32 bits. Two benefits of using mixed precision are a significantly shorter training time and lower memory requirements. Micikevicius et al. [90] show that utilizing mixed precision training on deep CNNs (AlexNet, InceptionV2, ResNet50 etc.) on large datasets maintained the same accuracy as with single-precision using the same set of hyperparameters. Mixed precision training is only fully utilized on GPUs with tensor cores. The hardware used in this thesis, described in section 3.1.3, have tensor cores. Tensor cores have been optimized for performing the computation of a matrix-matrix multiplication of two half-precision square matrices and an addition of a third half- or single-precision matrix. Since these types of computations are common in all NNs, the use of tensor cores is intended to speed up the training time of NNs.

All training computations in PyTorch are by default performed using single-precision (or FP32). Mixed precision training is the technique of using half-precision (or FP16) for specific computations during training, without causing the model to diverge. Divergence occurs because some gradients fall below the half-precision range since they become zero due to rounding errors.

To maintain the ability for model convergence and model accuracy during mixed precision training, one keeps a "master copy" of the FP32 weights, and a corresponding copy of the FP16 weights. The weights and activations of the forward pass, and the weight and activation gradients of back-

propagation are computed using the FP16 copy. In the last step where the weights are updated, the FP16 weight gradients from backpropagation are applied to the original FP32 copy to update the weights, thus resulting in a FP32 weight matrix. This step prevents underflow during model training.

All optimization operations are done using the FP32 master copies to maintain accuracy. Despite these operations, we are still able to speed up the training time and reduce the memory capacity. Since some gradients become too small to be represented in FP16 as mentioned earlier, the product of multiplying a small gradient with a small learning rate can become zero during optimization, thereby unfavorably affecting the model accuracy.

Loss scaling is another step in mixed precision training to prevent underflow. Scaling up the small gradients will preserve values that will not be represented in FP16 range. Scaling the gradients by some factor ensures model convergence and maintains accuracy as compared to full FP32 training. Scaling the loss before backpropagation is an efficient method of scaling all the gradients by the chain-rule in backpropagation. The loss must be unscaled before gradient update. The pipeline for the process of mixed-precision training is illustrated in Figure 3.2.

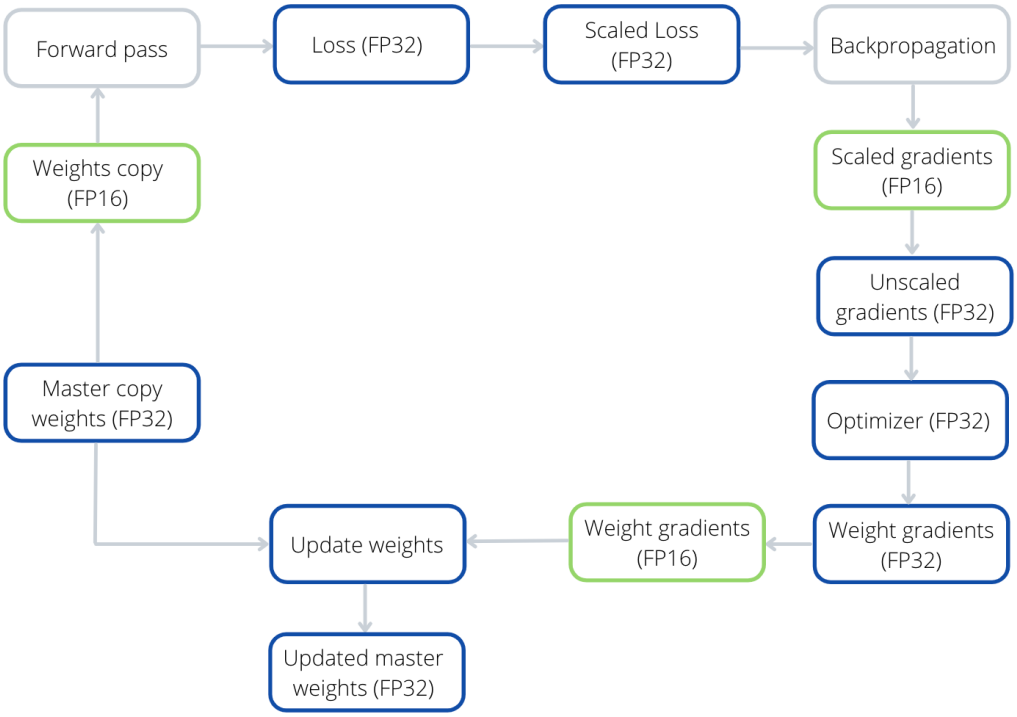

Figure 3.2: This flowchart represents the mixed precision training pipeline. The blue boxes represent the FP32 variables, and the green boxes represent the FP16 variables.

Now that the general idea of how mixed precision training should be fully utilized has been explained, the rest of this section contains the technical details on how this was implemented.

Autocasting means automatically converting specific variables to a desired precision to improve GPU performance. Wrapping the forward pass calculations inside the `torch.cuda.amp.autocast` context manager enables autocasting for these computations. Autocasting is thread-local which must be considered when working with parallelization on multiple GPUs. An example of using an autocast-wrapper is illustrated on lines 3–6 in Listing 3.1. The next step is to scale the loss using the `torch.cuda.amp.GradScaler` module to create scaled gradients for backpropagation. Even though the scaled gradients flow through backpropagation, the backpropagation itself is not done inside the autocast context manager. The default initial scale factor in PyTorch is 65536.0 (2^{16}). The gradients are then unscaled and used in the optimizer before it updates its parameters, such that the scale factor does not hamper the learning rate.

The `torch.cuda.amp.GradScaler` module automatically detects and stops overflows or underflows by checking if the gradients contain infs (infinity values) or NaNs (Not a Number), and if not, the scaling factor grows with a factor of 2. For the case of infs or NaNs being detected, the optimizer skips the current step, and the scale factor is reduced by a factor of 2.0 and the optimizer skips the currents step to avoid corrupting the parameters. An example of using such a scaler is illustrated in Listing 3.1.

Listing 3.1 represents an example pseudo-code for integrating mixed-precision training.

```
1     scaler = torch.cuda.amp.GradScaler()
2     for input, target in data:
3         # forward pass
4         with torch.cuda.amp.autocast():
5                 output = model(input)
6                 loss = criterion(output, target)
7         end
8         # backpropagation
9         optimizer.zero_grad()
10        scaler.scale(loss).backward()
11        scaler.step(optimizer)
12        scaler.update()
13    end
```

Listing 3.1: Pseudo-code for using an autocasting wrapper and scaling gradients.

In Listing 3.1, line 9 represents zeroing out previous gradients, line 10 represents the loss scaling, line 11 represents updating the gradients, and line 12 represents updating the scale factor.

3.3.5 Hyperparameter Tuning

Due to the time and memory constraints explained in Sections 1.3 and 3.1.4, we did not utilize extensive hyperparameter tuning techniques during model training. Some of these extensive methods involve using K-fold cross validation [128] or grid-search algorithms [80]. These methods are known to be computationally expensive, time-consuming and require enormous amounts of memory because we need to store the model parameters for each fold or grid element corresponding to a specific hyperparameter configuration.

Instead of using extensive tuning schemes, we chose to tune hyperparameters manually by using almost the same set of hyperparameters as in the ResUNet++ paper [66]. This is because this model was able to obtain the state-of-the-art results using the same dataset as explained in Section 3.3.2. In this paper, the resulting test scores from training ResUNet++ was a DSC of 0.8133 on Kvasir-SEG. Hence, we found that using these parameters was a natural starting point for the manual hyperparameter tuning process. After the manual tuning process of the hyperparameters for each baseline model, the hyperparameters were held constant for each baseline model across the ensemble.

We chose a batch size of 32 for training the following models:

- U-Net.
- ResUNet++.
- Dropout-modified ResUNet++.
- Dropout-modified U-Net.

Note that each of these models were each trained using BCE and DSC as loss functions. For one deep ensemble, we used a total of (ensemble size $\times$ epochs) training iterations. Finally, we train one dropout-modified U-Net and one dropout-modified ResUNet++ each for the two different loss functions.

Parameter Initialization

For weight and bias initialization of each layer in each of the CNNs, we use the default parameter initialization provided in PyTorch which

is the Gaussian He initialization scheme described in Section 2.4.4 for all convolutional layers. We utilize He initialization such that the variance of activations across all layers are the same during each forward pass. This stability will help in avoiding vanishing and exploding gradients during backpropagation. This initialization scheme was also used in the original deep ensemble paper [73], and we wanted to be consistent with this approach to be able to reproduce these results.

Optimizer

We chose the Adam optimizer described in Section 2.4.3 for training each model in List 3.3.5. Adam was chosen to avoid getting stuck in sub-optimal endpoints such as saddle points or local minimums, as it includes the use of momentum and an adaptive learning rate. Adam is also preferred before other optimizers as we have had several previous satisfactory experiences with using this optimizer. We used Adam with its default parameters provided by PyTorch except the learning rate.

The default PyTorch values of Adam is a $\beta_1^t = 0.9$ and $\beta_2^t = 0.999$ in the Adam algorithms (i.e., equations 2.9). To facilitate numerical stability, a small floating-point value of 10^{-8} is added to the denominator of Equation 2.9e.

We did not use the default initial learning rate for Adam. Instead, the initial learning rate for all training schemes is set to a value of 10^{-4}. This was the learning rate value which resulted in the quickest and most stable convergence when tracking the validation loss value for each epoch during training.

Learning Rate Scheduler

We used no learning rate scheduler in training for the following models:

- The ensemble of U-Nets trained with BCE and DSC.
- The ensemble of ResUNet++ trained with DSC.
- Dropout-modified ResUNet++ with DSC.
- Dropout-modified U-Net with DSC.

The reason for not using learning rate schedulers for training on these models is because they were able to reach a high DSC that we defined at the start of Section 3.3.5. We did not consider using learning rate schedulers to further increase the DSC, as we already reached our end-goal of training. In addition to this, we have already used adaptive learning rate optimizers (Adam), as described earlier instead of adjusting the learning

rate manually. We also have good experience with only using adaptive learning rate optimizers from past similar experiments.

Training ResUNet++ with BCE loss was ultimately the most difficult to optimize out of all the models in List 3.3.5, both with and without dropout layers. Therefore, we included a learning rate scheduler into training the deep ensemble of ResUNet++ models trained with BCE. We used `optim.lr_scheduler.CosineAnnealingWarmRestarts` to periodically adjust the learning rate during model training. It utilizes the cosine function as a learning rate scheduler in addition to warm restarts. By warm restarts one means that the learning rate is reverted to its original value. Loshchilov et al. [84] was the first to introduce the concept of warm restarts, but with the combination with SGD. One can in practice combine warm restarts with any optimizer.

We decided upon this specific learning rate scheduler because we wanted to combine the strengths of having both a large and small learning rate. Using a large learning rate means that one increases the chances of effectively escaping sub-optimal minima but using a small learning rate will potentially allow the algorithm to converge to a near-true optimal point.

There are three main hyperparameters related to cosine annealing with warm restarts. The first is the minimum learning rate which we set to 10^{-8} as we wanted to avoid the case of potentially increasing convergence time. The second hyperparameter is T_0, which is the number of iterations until each warm restart, and we set this value to 20. The third hyperparameter is the multiplication factor that we want to multiply with T_0 to increase or decrease the number of warm restarts for a specific number of training iterations. We use the default multiplication factor, which is equal to 1.

Tracking Overfitting

We track the loss over the validation set and the loss of the training set by each epoch. We visualized this using Matplotlib [57]. During each epoch, after computing the training loss and updating the model parameters (i.e., weights and biases), we compute the training- and validation loss by summing the loss over each mini-batch and dividing it by the batch size. Thus, we obtain the mean loss over its corresponding dataset. We compute the training- and validation loss for each epoch and save these to a list. Afterwards, we visualize these in a plot showing the loss vs. epochs, which from hereon are referred to as *loss-plots*.

By analyzing the loss-plot, we observed if the validation loss started to increase for the predetermined number of epochs. We observed that there

were some general fluctuations in the loss, but overall, it did not start to increase for 150 epochs. Hence, we did not continue training for any of the models past the point of 150 epochs to sustain generalizability.

Loss Functions and Evaluation Metrics

BCE and DSC were separately used as loss functions during training. This is because we wanted to compare a model trained with a proper scoring rule to a model which is not. In addition, we wanted to compare a loss function that handles class imbalance (i.e., DSC) to a one that does not (i.e., BCE). Class imbalance is an especially important factor to consider for polyp segmentation, as many polyps appear small from the colonoscopy images. Hence, we are curious to see how BCE affects the model performance compared to DSC.

As evaluation metrics for evaluating the model at each epoch during training and during testing, we used DSC for evaluation metrics for each trained model. We compute the mean DSC based on the entire validation set to evaluate the model performance during training, whereas during test time, the mean DSC is computed based on the test dataset.

We avoided direct implementation of a Sigmoid layer to the last layer of U-Net and ResUNet++. Instead, we used `nn.BCEWithLogitsLoss` which combines a Sigmoid layer and BCE loss in a single class. This was to secure numerical stability and was recommended by the PyTorch documentation.[5]. With this considered, included elementwise Sigmoid computations in each image tensor, followed by a threshold which transformed each tensor element into values representing class labels before computing the DSC.

3.4 Uncertainty Estimation

This section provides the reader with all details on implementing the two main methods for estimating uncertainty, MC dropout and deep ensembles. The main steps include adding dropout layers to facilitate MC dropout, the tuning of hyperparameters, training, saving and loading models, prediction stacking, and visualization.

Both uncertainty-extraction pipelines heavily focus on the part of loading the pre-trained weights into CNNs. Therefore, Section 3.4.2 is dedicated to describing common pitfalls one should be careful of when loading model weights onto the GPU device.

[5]The documentation of the BCE loss module used can be found on https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

3.4.1 Monte Carlo Dropout Implementation

As described in Section 3.3.2, we decided upon ResUNet++ and U-Net architectures as the baseline models for the uncertainty estimation methods. To facilitate MC dropout, we modified these model architectures with dropout layers as these do not originally contain any dropout layers. We adapt the same approach as DeVries et al. [22] by implementing one dropout layer to each of the five intermediate blocks in the U-Net architecture. The dropout-modified U-Net is illustrated in Figure 3.3 where the green rectangles represent the dropout layers. A total of five dropout layers are added to U-Net.

Adding Dropout Layers

Furthermore, we modify ResUNet++ with dropout layers to facilitate MC dropout. We do so by adding one dropout layer after the second and third SE block in the encoder structure, and a dropout layer to each of the two intermediate decoder blocks after the concatenation. See Figure 3.4 for reference where a black rectangle represents the dropout layer. Hence, we add a total of four dropout layers to ResUNet++.

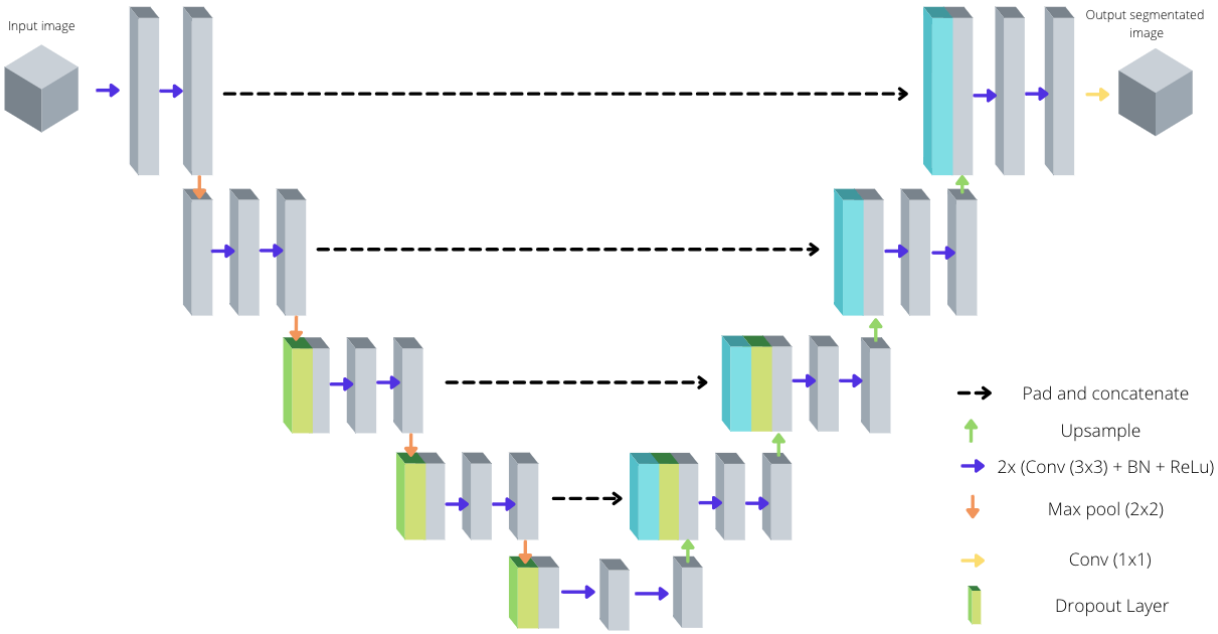

Figure 3.3: Figure represents the modified U-Net architecture containing dropout layers.

Tuning Dropout Rate

To determine the dropout rate, we tracked the validation DSC for the dropout-modified U-Net and ResUNet++. We did so for a total of 150

epochs using different dropout rates equal to 0.0 (which is equivalent to using no dropout layers), 0.1, 0.3, and 0.5. The goal behind the use of many candidate dropout rates was to determine the optimal dropout rate for the two dropout-modified models. We conducted a total of four experiments as we tuned the dropout rate using the DSC and BCE loss metrics for training the two models. From all experiments on U-Net, the dropout rate of 0.1 in the dropout layers resulted in the highest validation DSC out of the other dropout rates. Hence, we chose this dropout rate for the dropout-modified U-Net. When training the dropout-modified ResUNet++ with DSC loss, we used a dropout rate of 0.1 because the effect of dropout did not seem to make a significant difference to the evaluation during training. When training ResUNet++ with BCE loss, we found that a dropout rate of 0.3 gave the highest validation score during training, after 90 epochs. With this considered, we decided to use this dropout rate and stop training after 90 epochs where we reached the trained model corresponding to the highest validation score. Considering the results of these experiments, we ended up using the dropout rate of 0.1 for all ensembles except for the dropout-modified ResUNet++ trained with BCE loss where we used a dropout rate of 0.3.

Many NN architectures with dropout layers may contain different dropout rates corresponding to different dropout layers throughout the network. Due to the time-demanding labor of tuning the dropout rate for each dropout layer, we restricted ourselves to only include a global dropout rate for each model.

Saving and Loading Models

After determining the placement of dropout layers in the models and their corresponding dropout rate, we proceeded by training the two dropout-modified models. We used the preprocessing steps and data augmentations for the input training data as explained in Section 3.2 and Section 3.2.4.

For training, we used two different loss functions, DSC and BCE, which resulted in a total of four training sessions for each dropout-modified model. We used a total of 150 epochs when training each dropout-modified model, with the exception of training the dropout-modified ResUNet++ with BCE loss, in which we used 90 epochs.

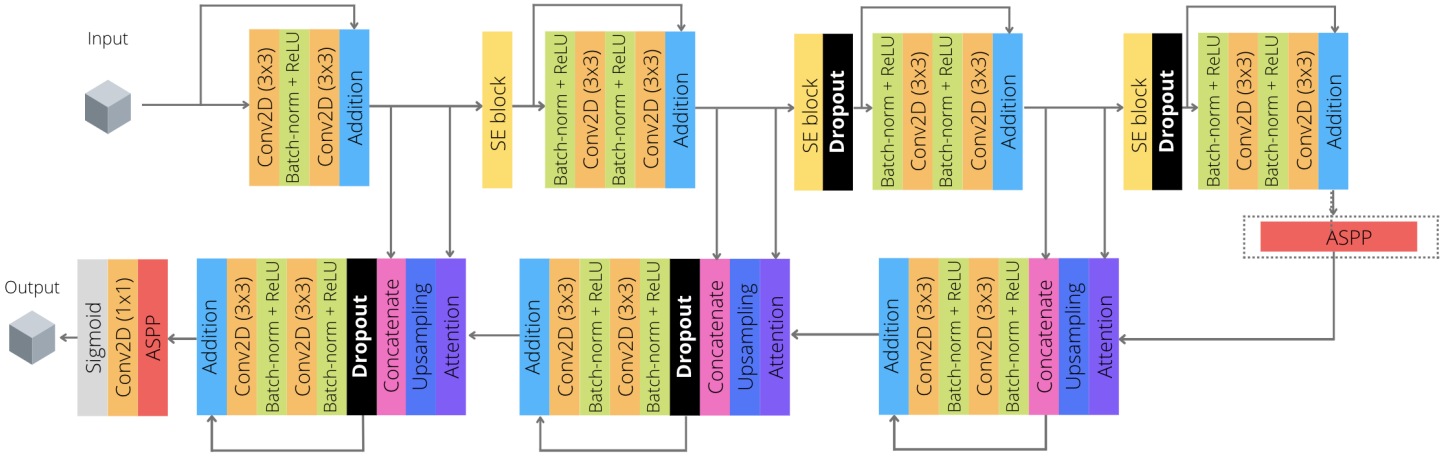

Figure 3.4: Figure represents the dropout-modified ResUNet++ architecture.

After the last epoch, the model parameters were saved as a Python dictionary object using `torch.save(checkpoint, PATH)` where checkpoint contains dictionary objects that represent the model parameters, the current epoch, or the optimizer state. The dictionary object ensures that each saved tensor value is mapped to its corresponding layer. This dictionary object is saved in a *.pt*-file and then loaded into a model architecture instance. Note that one must define the original model architecture which corresponds to the tensors in the dictionary.

One can additionally save the state of the optimizer. By doing so, one can proceed with the training process later. The saved model can, at a later point in time, be loaded and used for prediction on other datasets. An example of how to load a saved model in PyTorch is shown in Listing 3.2.

```
# load model
device = torch.device("cuda")
model = ModelArchitecture().to(device)
checkpoint = torch.load(PATH, map_location =
    device)
model.load_state_dict(checkpoint["
    model_state_dict"])
```

Listing 3.2: Code example that demonstrates how to load a saved model on a specific device.

Once a model is loaded, one must call `model.eval()` to freeze batch-norm and dropout layers before evaluating the model upon prediction. To facilitate MC dropout, we want to keep dropout layers on or in training-mode during evaluation. We must define a module that ensures this

mechanism. Such a module is defined in Listing 3.3, by enabling the dropout layers on line 5.

```
1 def enable_dropout(model):
2     # enable dropout layers
3     for m in model.modules():
4         if m.__class__.__name__.startswith("
    Dropout"):
5             m.train()
```

Listing 3.3: Listing represents a module that enables dropout layers during inference time.

Prediction Stacking

With a model that can be loaded and a module that enables dropout, we have the necessary tools to facilitate MC dropout. When using MC dropout, we define the number of forward passes during inference time as the ensemble size. The model predicts on the exact same test dataset for each forward pass. We load the model weights into their corresponding model architecture, call `enable_dropout(model)` and compute the prediction inside a loop, which is ensemble size-long. The predictions corresponding to one input image is collected at each forward pass and stacked together along a dimension that is ensemble size-long. It is important that the predictions that are stacked together correspond to the same input image. The process of prediction stacking is visualized in Listing 3.4 and also described in Paper B.2.

```
1 def stack_predictions(x):
2     K = []
3     # loop over models in ensemble
4     for model in ensemble:
5         prediction = model(x)
6         K.append(prediction)
7     # transform to probabilities
8     K = Sigmoid(K)
```

Listing 3.4: The listing represents pseudo-code for prediction stacking by each model in an ensemble.

We transform each element in the stacked tensor using the Sigmoid function to obtain the predicted probabilities, which is illustrated on line 8 in Listing 3.4. The elementwise mean is taken along the ensemble size-dimension. Thus, we obtain a mean prediction tensor corresponding with a size equal to the original input image size. Each element in the predic-

tion tensor is thresholded to represent a class label. The threshold value is equal to 0.5. We could in practice calibrate this value for this specific problem, but instead use the most common threshold value in binary and multi-level classifications due to time restrictions. Calibrating this threshold value leaves a topic for further research.

The variances from the sigmoided stacked tensor are computed. We normalize each variance tensor to be in the range of $[0, 1]$ and mapped to a colormap `matplotlib.pyplot.imshow(im, cmap="turbo")`. We prefer the Turbo colormap which is a rainbow colormap which clearly accentuates the areas of high variance as compared to other colormaps in Matplotlib. We observe that this type of colormap allows for quick visual assessment as it provides high-contrast and smoothly varying transitions to images. This colormap is also known to be color blindness friendly. The values in this colormap can be viewed in Figure 3.5, where the red colors represent the pixels with high values of variance and blue pixels represent low values of variance. We will refer to these images as uncertainty heatmap representations as they represent the spatial areas containing values of either agreement or disagreement between the models in the ensemble.

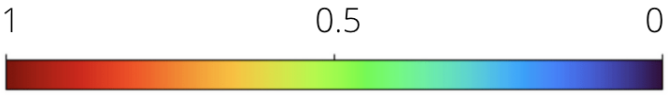

Figure 3.5: This colorbar represents the values in the Turbo colormap from the Matplotlib library.

3.4.2 Deep Ensemble Implementation

The main idea behind deep ensembles is to extract the uncertainty from a NN, where each network in the ensemble is an independently trained model. This process involves two main steps: (i) training an ensemble consisting of a baseline model and (ii) combining the predictions. Each model in the ensemble focuses on the same training and validation set throughout training. After each model in the ensemble is trained, the set of predictions based off the same test data is averaged across each model in the ensemble. The variance across each model prediction will represent the estimate predictive uncertainty.

We start by separately training each model. Each baseline model in the ensemble is trained with the same configuration of hyperparameters. We trained U-Net with DSC loss with the same hyperparameters (except the loss) as U-Net trained with BCE loss. For ResUNet++ trained with DSC, we used the same set of hyperparameters as ResUNet++ trained with BCE,

except that we must use a learning rate scheduler when training with BCE loss. This learning rate scheduler and the set of hyperparameters used for each model is described in Section 3.3.5.

A total of 16 separate models in each ensemble were trained. By starting the deep ensemble experiments by training a total of 5 U-Nets with DSC loss, we found a trend of increasing validation DSC with the increasing ensemble size. With this taken into account, we decided to increase the ensemble size up to 16 models to fully capture this trend, with a hope of maximally increasing the ensemble performance as much as possible. With the trend that we captured in the first generated deep ensemble of 16 U-Nets, we decided to keep the number 16 fixed for the next ensembles. Memory- and time constraints as mentioned in sections 1.3 and 3.1.4 also influenced this choice. The results from the first experiments using a deep ensemble of 5 baseline models can be found in Appendix B [61].

Each model was saved as a *.pt*-file, equivalent to the saving process explained in Section 3.4.1. Each ensemble was saved in a folder based on the same baseline model and loss, such that the algorithm can loop through an entire folder when one wants to load in a ensemble. Upon loading, we freeze the batch-norm layers by calling upon `model.eval()`, and each model in the ensemble predicts on the test data.

Prediction Stacking

Each prediction collected from a model is stacked along a dimension which represents the ensemble size. The Sigmoid activation function is used to transform the stacked predictions to probability values. The mean is computed along the ensemble size-dimension to obtain the average prediction by each model in the ensemble. We threshold each element in the mean prediction tensor by 0.5 to obtain the class labels. We were not able to calibrate this value due to time restrictions, but this opens a topic for future investigation. The elementwise variance is computed along the ensemble size-dimension from the previous stacked probability tensor. This process is illustrated in Listing 3.4, which is equivalent to the prediction stacking described in the MC dropout pipeline.

The variance tensor is normalized to lie in the range $[0, 1]$ to secure readability. We map the variance tensors to the colormap `matplotlib.pyplot.imshow(im, cmap="turbo")` provided by the Matplotlib [57] library.

Pitfalls in Ensemble Learning

The procedure of loading in a trained ensemble and combining its predictions is poorly documented in literature. With this considered, much time was invested into debugging and managing the loading and prediction process of deep ensembles. One must be careful when loading in different models because we noticed strange behaviors during this process. More specifically, we unintentionally used the same *.pt*-file upon loading model parameters into a defined model architecture. We noticed this problem because the DSC remained constant with the ensemble size and the variance for each prediction was equal to 0 across the ensemble. The only solution to this problem was to separately load the model parameters into the model architecture during the forward pass method. Loading the different model parameters outside the forward pass method caused the algorithm to load in the same model parameters even though the algorithm looped through different *.pt*-files.

A similar problem occurred once we visualized the DSC for different ensemble sizes. We noticed that iterating through an ensemble size of 1 to 16 using a list of paths where the pre-trained model weights were stored, caused the algorithm to load in the same model parameters for each model in the ensemble. But when manually changing, i.e., defining the ensemble size by having a fixed list of paths, resulted in a different DSC for one ensemble size compared to the DSC for another ensemble size. To overcome this problem, the ensemble size was sent into the deep ensemble class as a parameter using a bash script which loops over the different ensemble sizes and redefines the directory for each iteration. After completing these experiments, we found that the `model.load_state_dict()` (given in Listing 3.2) or the `torch.load(PATH)` method serializes this Python dictionary object[6]. We know that the serialized data is bound to the path, but we made sure to de-serialize this object before loading. Despite this, we suspect that our loading problem stems from this serializing mechanism, resulting a path or directory that cannot be redefined for each iteration. Our results support this claim because when we tried to re-define paths, the same model weights gets loaded in for a defined ensemble size. The cause of this pitfall leaves a topic for further investigation.

3.5 Summary

These methods were motivated by the lack of explainability in DL. We argue that uncertainty estimates can provide explainability to non-DL experts by delivering additional information about the inner mechanics be-

[6] https://pytorch.org/tutorials/beginner/saving_loading_models.html

hind their decision-making process. Providing explainability in the form of uncertainty estimates can be used to critically assess if the model gives overconfident results. The need for explainability and transparency to the model decision can be used by medical experts to establish accountability and trust between the model and user. The potential gain from explainability methods is the possible integration of DL models in medical clinics to prevent observational oversights which may improve prognosis upon diagnosis.

In this chapter, we presented the main methods behind the process of training deep CNNs to estimate their predictive uncertainties. These methods include the preprocessing steps of the input data, hyperparameter tuning, model selection, model training, mixed-precision training and implementation of the MC dropout- and deep ensemble pipeline. The preprocessing steps may vary depending on the problem at hand. For example, data augmentations are not strictly necessary in the rare case of having millions of training examples. The process of tuning hyperparameters is a non-trivial task and the becomes increasingly difficult with the number of hyperparameters. Tracking loss is one efficient method to avoid overfitting during model training. Mixed-precision training can become strictly necessary for training deep CNNs when being restricted by memory. We finish this chapter by describing important pitfalls of ensemble prediction on the GPU. The models were used to construct a set of different ensembles which we were able to extract the predictive uncertainties in terms of spatial heatmap representations. The corresponding results from these methods will be presented in the next chapter.

Chapter 4

Results

In this chapter, we present the results from tuning, testing and uncertainty estimation of DL models. The majority of the results is based on the test dataset, as these reflect the level of generalization from the trained models. Many of these results include the uncertainty representations which we extracted from an ensemble using MC dropout and a deep ensemble using the hold-out test set based on the Kvasir-SEG dataset.

4.1 The Effect of Dropout Rate

In this section, we present the results from training and tuning the dropout models. We include the uncertainty heatmap representations based off the MC dropout pipeline.

The following figures represent the results of the effect of using different dropout rates to each dropout-modified model architecture. These are depicted in Figure 4.1 and Figure 4.2. This effect is measured in terms of the DSC based on the mean prediction on the Kvasir-SEG validation set consisting of 100 images after each training iteration. We observe the DSC for each epoch for different colors corresponding to a specific dropout rate value. Blue corresponds to a model with no dropout layers, orange corresponds to a dropout rate equal to 0.1, green corresponds to a dropout rate of 0.3, and red corresponds to a dropout rate of 0.5.

Figure 4.1 represents the dropout rate effect of a dropout-modified U-Net model trained with the BCE loss, and the same applies for Figure 4.2 but trained using the DSC loss.

Figures 4.3 and 4.4 represent the results from training ResUNet++ with different dropout rates. Figure 4.3 represents the dropout-modified ResUNet++ model using different dropout rates and trained with the BCE loss, and the same applies for Figure 4.4 but trained with the DSC loss.

Figure 4.1: The result of dropout-modified U-Net trained with BCE loss and validated on Kvasir-SEG validation set on each epoch.

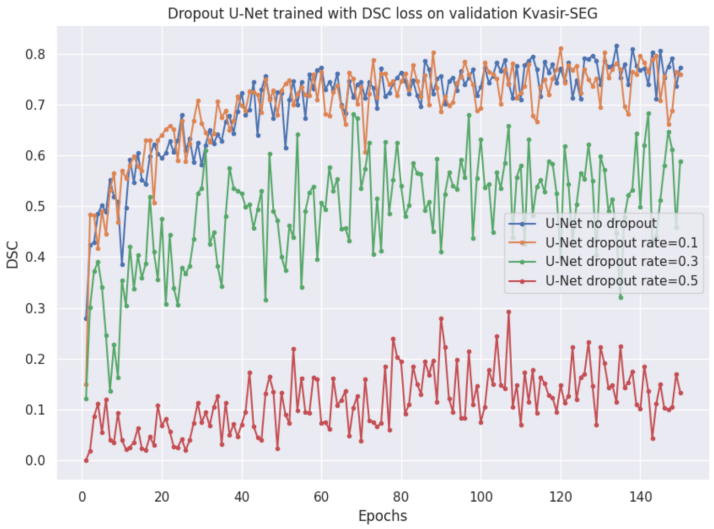

Figure 4.2: The result of dropout-modified U-Net trained with DSC loss and validated on Kvasir-SEG validation set on each epoch.

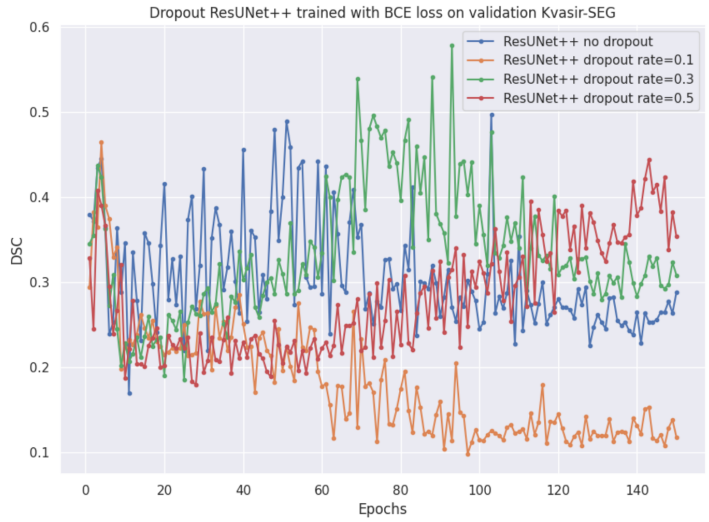

Figure 4.3: The result of dropout-modified ResUNet++ trained with BCE loss and validated on Kvasir-SEG validation set on each epoch.

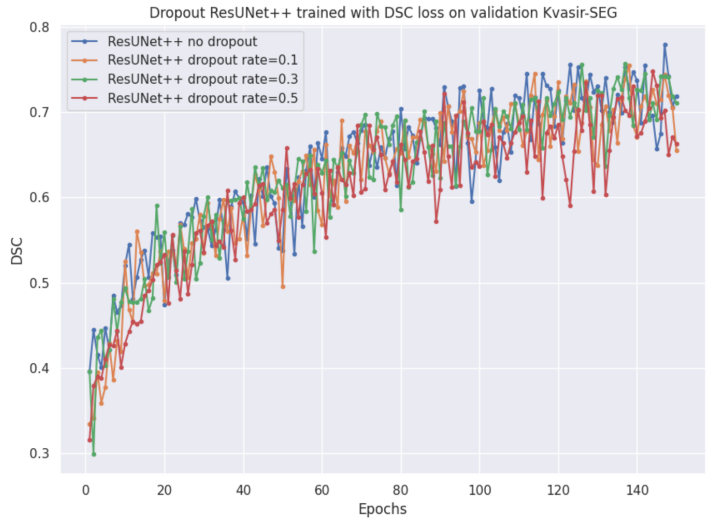

Figure 4.4: The result of dropout-modified ResUNet++ trained with DSC loss and validated on Kvasir-SEG validation set on each epoch.

4.2 Predictions and Uncertainty Heatmaps

This section contains the resulting predictions and their corresponding uncertainty representations based on the Kvasir-SEG test data. These results are based on two different uncertainty estimation approaches, MC

dropout and deep ensembles. We mainly include images based on ensembles with an ensemble size of 16. To enable easy comparison later in Chapter 5, we display the results from these approaches side-by-side. The results corresponding to MC dropout are displayed on the left, whereas the results from the deep ensemble are displayed on the right in figures 4.5, 4.6, 4.7 and 4.8. The mean DSC from test time corresponding to these images are displayed in Table 4.2 and Table 4.1.

Figure 4.5 corresponds to the original input images, ground truth masks, predicted masks and uncertainty representations. These results are based on three randomly chosen original input images from the Kvasir-SEG dataset and were generated using MC dropout and a deep ensemble. In both methods, U-Net is used as the baseline model and trained using DSC loss with an ensemble size of 16. These results are from Paper B.2.

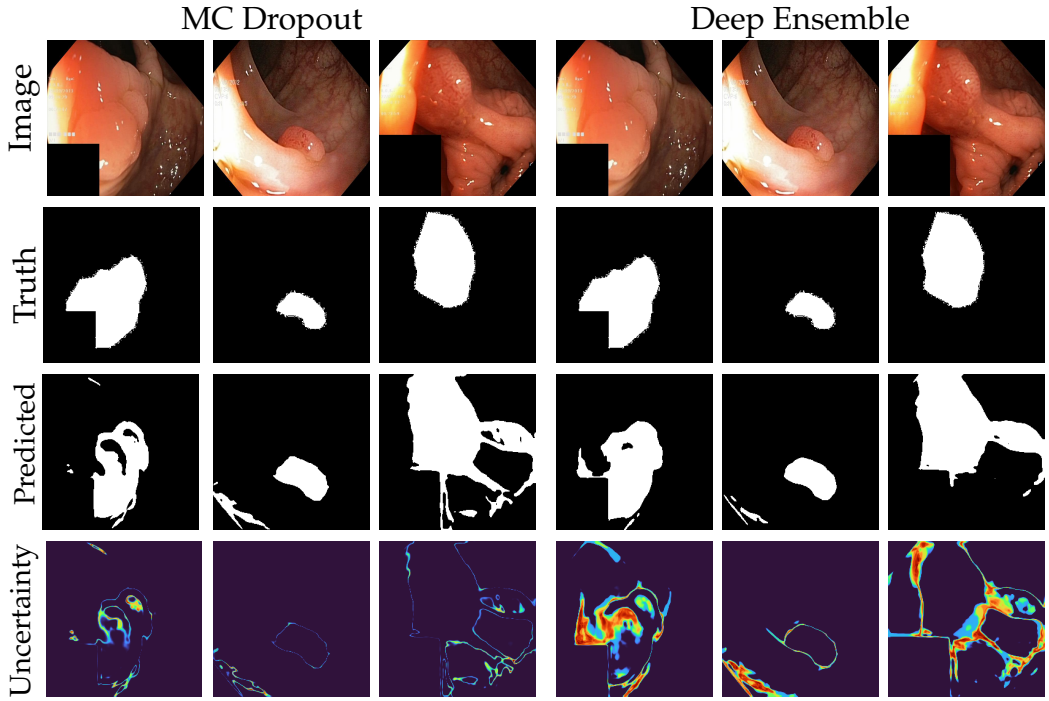

Figure 4.5: Three original input images from Kvasir-SEG, the actual annotations, predicted annotations, and their corresponding predictive uncertainties. These images were generated using MC dropout and deep ensembles with U-Net as the baseline model trained with the DSC as the loss function. We display the results from the two methods side-by-side.

Figure 4.6 contains original input images, ground truth masks, predicted masks and uncertainty representations based on three input images from the Kvasir-SEG test dataset. The predicted masks and uncertainty representations were generated using MC dropout and a deep ensemble, each with an ensemble size of 16. In both ensembles, U-Net is used as the

baseline model and trained with the BCE loss.

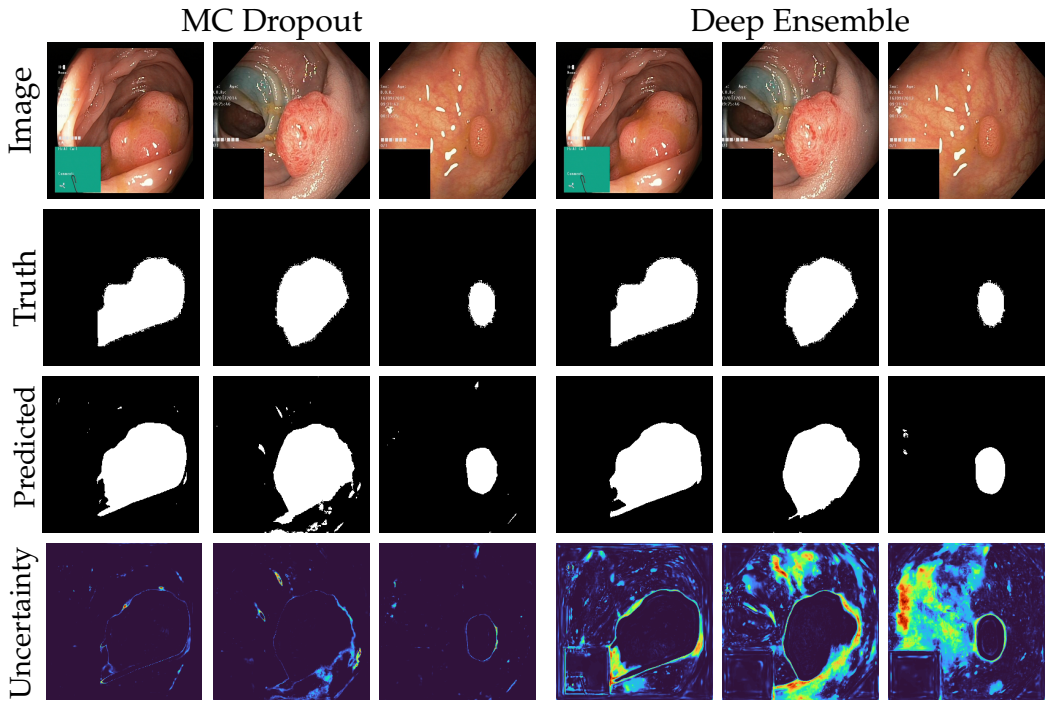

Figure 4.6: Three original input images from Kvasir-SEG, ground truth annotations, predicted annotations, and their corresponding predictive uncertainties. These images were generated using MC dropout and deep ensembles with U-Net as the baseline model trained with the BCE loss function.

Figure 4.7 displays the original input images, ground truth masks, predicted masks and uncertainty representations corresponding to two ResUNet++ based ensembles trained with DSC loss. These results are based on three images from the Kvasir-SEG test dataset. The two ensembles were generated using the MC dropout- and deep ensemble method. In both approaches, the predicted masks and uncertainty representations were generated using an ensemble size of 16 with ResUNet++ as the baseline model trained with DSC loss.

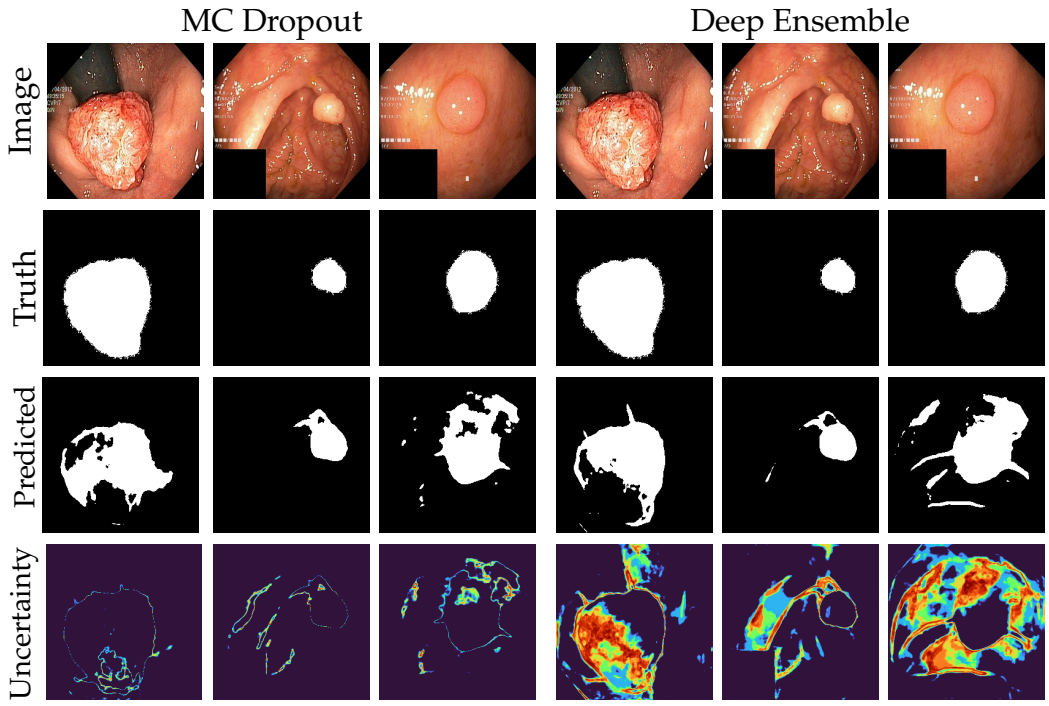

Figure 4.7: Three original input images, ground truths, predictions and uncertainty heatmaps based on the Kvasir-SEG test dataset. The predictions correspond to an ensemble of 16 MC dropout ensemble of ResUNet++ as baseline. The baseline model is trained with DSC loss.

Figure 4.8 contains the original input images, ground truth masks, predicted masks and uncertainty representations corresponding to the MC dropout ensemble and the deep ensemble. These ensembles were based on ResUNet++ as the baseline model and trained with the BCE loss function. These images are based on the three input images (top row) from the Kvasir-SEG test dataset. The predictions and uncertainty representations correspond to outputs using the MC dropout and deep ensemble approach. Both methods are based on an ensemble size of 16.

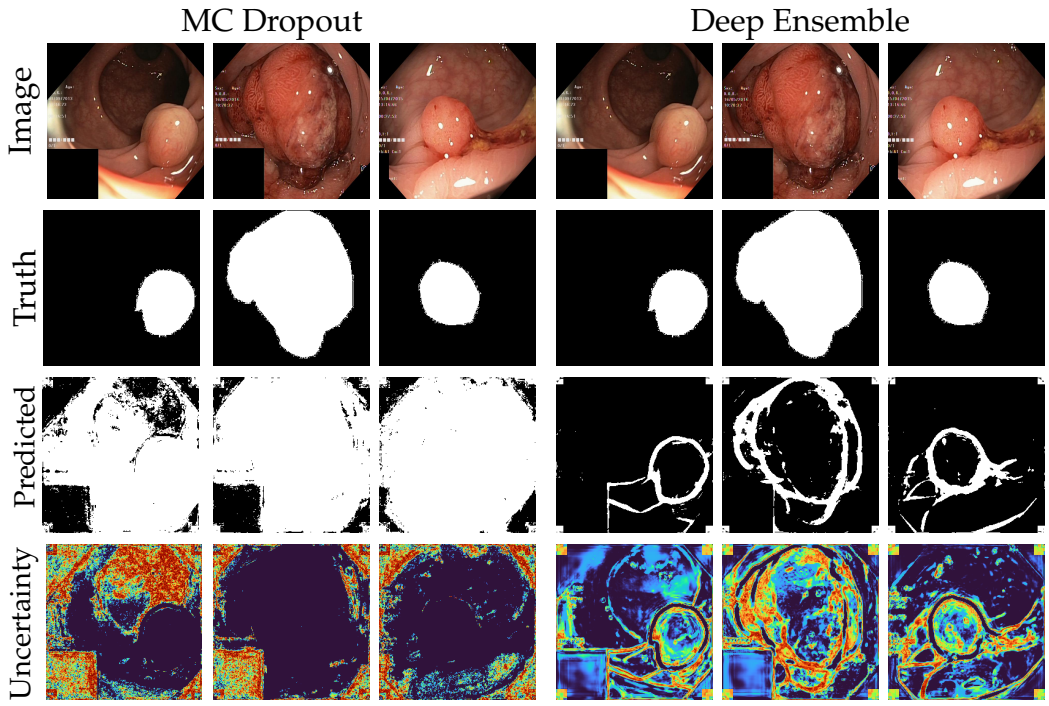

Figure 4.8: Three original input images, ground truths, predictions and uncertainty heatmaps based on the Kvasir-SEG test data. These predictions correspond to an ensemble of 16 MC dropout models with ResUNet++ used as baseline. The baseline model is trained with BCE loss.

The resulting predicted segmentation mask and uncertainty representations corresponding to the use of different loss functions from figures 4.5, 4.6, 4.7 and 4.8 are displayed side-by-side depending on the model architecture in Section 4.3.

Table 4.1 represents the mean DSC score based on the Kvasir-SEG test dataset. These results are based on different MC dropout ensembles from using two different dropout-modified model architectures, ResUNet++ and U-Net. Each model was trained with two loss functions, DSC and BCE, during training. These results correspond to an ensemble of 16 baseline models.

Table 4.1: The summarized mean test DSC from an ensemble generated by MC dropout using 16 forward passes with different CNN architectures as baseline models.

DSC	Baseline Model	Loss Metric
0.7190	ResUNet++	DSC
0.2850		BCE
0.7538	U-Net	DSC
0.7505		BCE

Table 4.2 represents the summarized results from a deep ensemble using two different CNN architectures as baseline models, ResUNet++ and U-Net, trained with different loss functions, DSC and BCE, during training. These results correspond to an ensemble size of 16.

Table 4.2: The summarized mean test DSC from a deep ensemble with an ensemble size of 16 using different CNN architectures as baseline models.

DSC	Baseline Model	Loss Metric
0.7370	ResUNet++	DSC
0.3346		BCE
0.8172	U-Net	DSC
0.8034		BCE

Figure 4.9 represents the 16 individual predicted masks from an deep ensemble of ResUNet++ models with an ensemble size of 16 trained with DSC loss. The right image represents the uncertainty representation corresponding to the 16 individual predictions from the left side in the figure. This figure is from a submitted research paper given in Paper B.2.

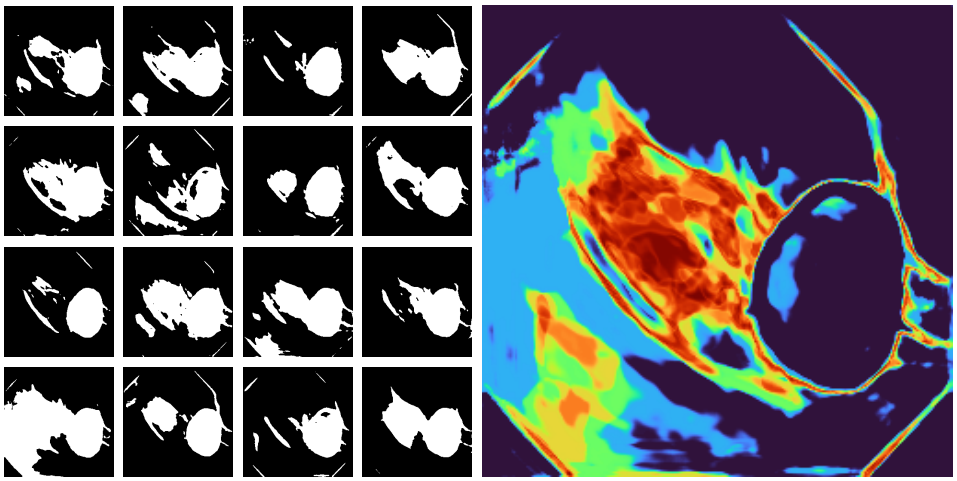

Figure 4.9: The right image represents the uncertainty representation based on 16 individual predictions shown on the left. These images are based on the ResUNet++ deep ensemble.

4.3 Comparing Ensembles Trained with Different Loss

In this section, we collect the images from the previous section and display them side-by-side for easy comparison in the next chapter.

Figure 4.10 displays the collected results by using U-Net as the baseline model in the two different uncertainty estimate methods. We focus on the comparison of general trends in the results between the use of two different loss functions, DSC and BCE. Left labels indicate their respective loss function. These images are also displayed in figures 4.5 and 4.6. We emphasize that the DSC-based results are based on another set of input images that the BCE-based results. These are still displayed here to compare general trends.

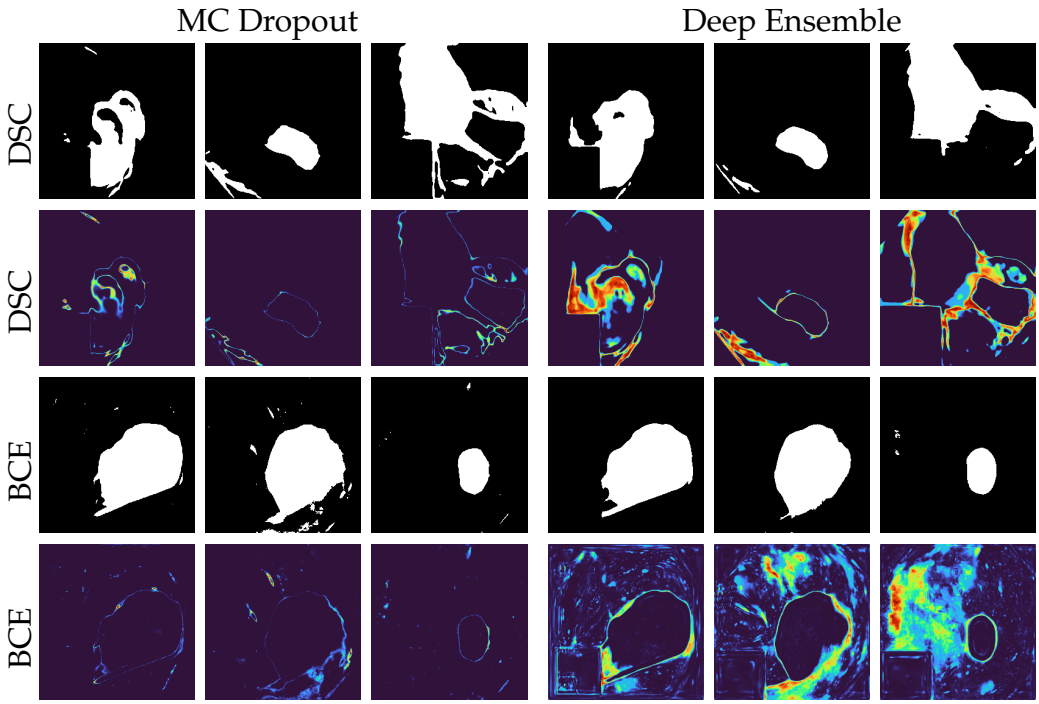

Figure 4.10: We display the results from different U-Net-based ensembles using different loss functions, DSC and BCE.

We collect the results from Figure 4.7 and Figure 4.8 which are displayed as a single figure in Figure 4.11. We emphasize that the DSC-based results are based on another set of input images that the BCE-based results. These images are displayed together to compare general trends. These results are based on the two different uncertainty estimation approaches, MC dropout and deep ensembles, with ResUNet++ as the baseline model using an ensemble size of 16.

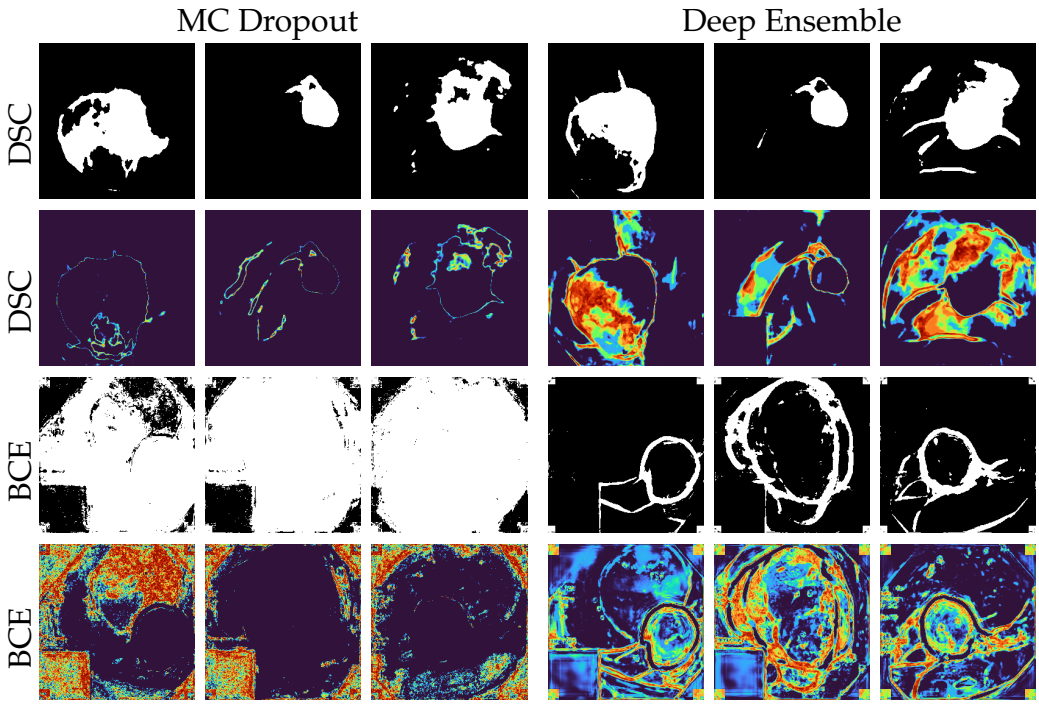

Figure 4.11: We display the results from different ensembles based on ResUNet++ as baseline using different loss functions, DSC and BCE.

4.4 Performance of MC Dropout and Deep Ensembles

In this section, we summarize the performance in terms of the mean test DSC of MC dropout and deep ensemble with different values of ensemble sizes ranging from 1–16. We split into two figures, where Figure 4.12 represents ensembles with U-Net used as the baseline model, and Figure 4.13 represents the ensembles with ResUNet++ as the baseline model.

We borrowed inspiration from Mehrtash et al. [89] to visualize the results given in figures 4.12 and 4.13. More specifically, one specific figure from their paper displays the loss vs. the ensemble size for the same baseline model but trained with different loss functions.

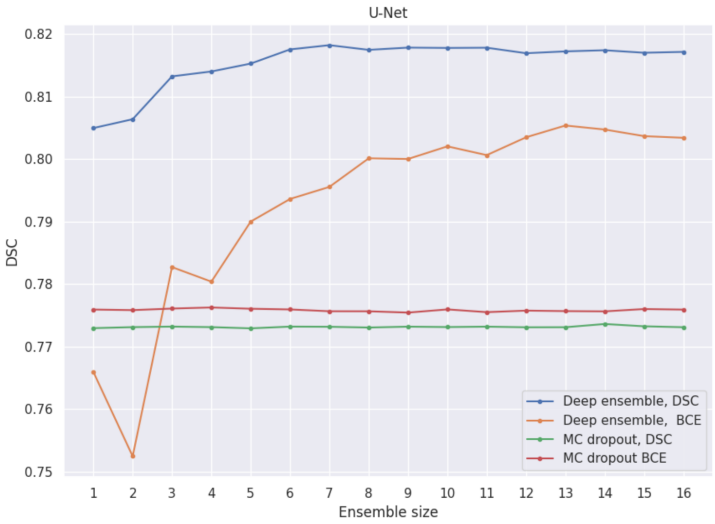

Figure 4.12: The mean DSC during test time for MC dropout ensembles and deep ensembles using U-Net as the baseline model. The different ensembles are trained with different loss functions.

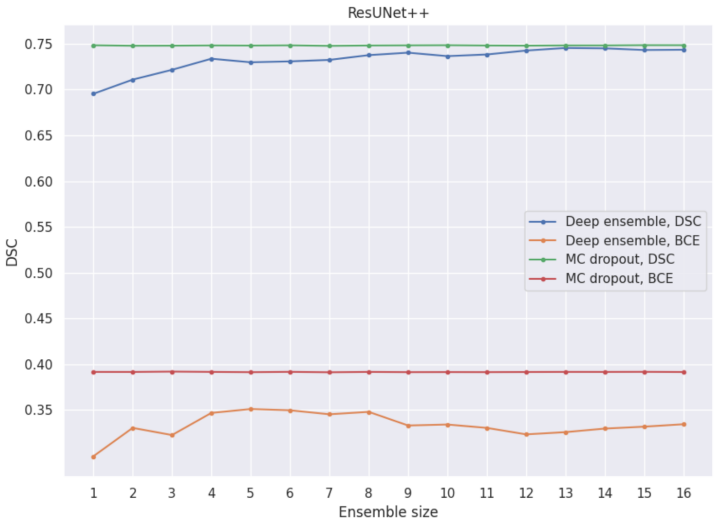

Figure 4.13: The mean DSC during test time for MC dropout ensembles and deep ensembles based on ResUNet++ using different loss functions.

One important note is the DSC values given by the different MC dropout ensembles in Table 4.1, do not correspond to correspond to the values given in Figure 4.12 and Figure 4.13 because of the randomness given by dropout layers at each test run.

4.5 Summary

In this chapter, we presented the results from the experiments which we described and explained in Chapter 3. Section 4.1 contains the results from tuning the dropout rate hyperparameter for the dropout-modified CNNs, U-Net and ResUNet++, trained with different loss functions.

In Section 4.2 we presented the results from the two uncertainty estimation approaches, MC dropout and deep ensembles, for the two CNN architectures trained with two different loss functions, DSC and BCE. We displayed the results from the two different uncertainty estimation methods side-by-side. We do so to make the methods easily comparable as we discuss the differences of these results in Chapter 5. Figures 4.5, 4.6, 4.7 and 4.8 represent these figures. Each of these figures is based on 3 original images from the Kvasir-SEG test dataset. Tables 4.1 and 4.2 represent the corresponding test scores from the entire test dataset. Lastly, Figure 4.9 represents the 16 different predicted masks corresponding to each of the models in a deep ensemble. Their corresponding uncertainty representation is given on the right side of the figure. These predictions including

the uncertainty estimate are based on a deep ensemble consisting of 16 ResUNet++ models trained with DSC loss.

In Section 4.3, we collect images from Section 4.2 and collect the results corresponding to the same CNN architecture but based on different loss metrics during training. The results corresponding to a specific configuration of baseline model and loss metric are based on different input images. Despite this, they are displayed together to capture general trends between the methods.

Section 4.4 contains the summarized test results from the trained ensembles based on the two different CNNs, U-Net and ResUNet++. The test DSC is displayed for different ensemble sizes ranging from 1 to 16. Figure 4.12 represents the DSC scores for different ensemble sizes from a U-Net-based MC dropout ensemble and a deep ensemble trained with DSC and BCE. Figure 4.13 and Figure 4.12 represent performance for different ensemble sizes in terms of the mean DSC, using U-Net and ResUNet++, respectively.

These results displayed in this chapter are visualized to emphasize the difference in uncertainty representation, prediction, and performance, using two different uncertainty estimation methods. We also include the use of two different CNN architectures each trained with two different loss functions for further comparison. These results provided in this chapter lay the foundation of the discussions presented in the next chapter.

Chapter 5

Discussion

In this chapter, we discuss the results given in Chapter 4 given by the experiments described in Chapter 3. We focus on the comparison between uncertainty estimation approaches with the use of different CNN architectures. These models were trained with different loss functions for further comparison. Furthermore, we repeat the research objectives and research question defined in Chapter 1. We explain why these results reached our objectives and answer the research question. These answers aim to provide the readers with the main contributions behind this thesis.

5.1 Dropout Rate

In Figure 4.1 and 4.2, we see that an increasing dropout rate results in a lower DSC score for the validation dataset during training. For U-Net trained with the BCE and DSC loss, a dropout rate of 0.1 and using no dropout gave approximately the same validation DSC score for each epoch. We ended up using a dropout rate of 0.1 because we needed dropout layers to facilitate MC dropout. Until approximately 40 epochs, the DSC for a dropout rate of 0.0 and 0.1 increases steeply. The increase in DSC stabilizes after this point. The models corresponding to larger dropout rates seem to have a high variance in the DSC between each epoch, which indicates a instability of large regularization provided by large dropout rates.

Figure 4.3 represents the dropout rate tuning for ResUNet++ trained with BCE. Figure 4.4 represents the tuning of the corresponding model but trained with DSC. For Figure 4.3, we observe that the largest dropout rate of 0.5, resulted in the lowest validation score during training. Despite this, we see that the overall highest DSC score was obtained using a dropout rate of 0.3 around the 90th epoch. We therefore save the model corresponding to a dropout rate of 0.3 after 90 epochs as this model converges at the highest validation score around this area as compared to the

different models with other dropout rates. For the dropout-modified ResUNet++ trained with DSC, we observe that the model behaves roughly the same during training and converges independently of dropout rate. Thus, we chose to use the model corresponding to a dropout rate of 0.1 for the MC dropout experiments because we wanted to limit the effect of regularization by using larger dropout rates.

5.2 Deep Ensemble and MC Dropout with U-Net

We discuss our results presented in Chapter 4 with a focus on Section 4.2 and Section 4.3, which contains the predicted segmentation masks and corresponding uncertainty representations based on MC dropout and the deep ensemble uncertainty estimation methods. This section only focuses on the U-Net based ensembles for the two uncertainty estimation methods.

5.2.1 U-Net Trained with DSC

We regard the predicted masks and uncertainty representations given by the U-Net based MC dropout ensemble and deep ensemble trained with DSC in Figure 4.5. Our immediate observation is that the deep ensembles give predicted segmentation masks that are more similar to the ground truth compared to the predictions provided by MC dropout. This observation is also supported by the fact that the mean DSC scores based on the Kvasir-SEG test dataset which are given in Table 4.1 and Table 4.2. From these, we see that the MC dropout gives a mean DSC score of 0.7538 and the deep ensemble gives a mean DSC equal to 0.8172.

When observing the uncertainty representations given by MC dropout in Figure 4.5, we notice that the uncertainty estimates are elusive and therefore give little information about the uncertainty corresponding to the model prediction. Considering this, we regard these uncertainty estimates as difficult to interpret due to the low and few uncertainty values across the uncertainty representations. Despite these elusive uncertainty representations, we see that the areas of high uncertainty values correspond to pixels gathered around the border of the predicted segmentation mask. These results indicate that the MC dropout ensemble agrees upon the pixels corresponding to the predicted polyp. Despite this, there are few to no uncertainty estimates in the majority of the areas that correspond to FP or FN pixels. Instead, the uncertainty estimates are concentrated around the boundary of the FP and FN areas. However, inside these areas, we see no direct connection between the incorrectly classified pixels and the corresponding uncertainty values. We would like to see large uncertainty estimates inside the incorrectly predicted regions. Considering this, as this

uncertainty estimation method lacks this property, this estimation method is not sufficient for this specific case.

We now observe large areas of high and low uncertainty values across the uncertainty representations given by the deep ensemble in Figure 4.5. These values are a contrast to the low and few uncertainty estimates given by the MC dropout approach. Like the uncertainty representations given by the MC dropout approach, there are areas of medium to high uncertainty values concentrated around the pixels which correspond to the border of the predicted polyp. However, the deep ensemble includes uncertainty values located at the same area which corresponds to the edge of the anchor object in the bottom left corner as well. These results indicate that the ensemble recognizes the polyp border and the corner anchor object. As a result, the deep ensemble can predict large amounts of TPs and true negatives (TNs) despite having large quantities of corresponding uncertainty estimates. In some of the areas with large uncertainty estimates, we see that a large amount of these pixels are being misclassified. In other words, these areas correspond to FP and FN predictions in the predicted segmentation mask, indicating that the large uncertainty estimates and their corresponding pixels are likely to be misclassified in the predicted mask. Due to these results, we suspect a connection between the large uncertainty estimates and the corresponding pixels that are misclassified in the predicted segmentation masks.

5.2.2 U-Net Trained with BCE

Figure 4.6 shows the resulting predictions and uncertainty estimates given by the MC dropout ensemble and the deep ensemble using U-Net trained with BCE as the baseline model. The predicted masks given by the MC dropout and deep ensemble method seems to be of similar quality, but in contrast to the deep ensemble predictions, the noise (i.e., white pixels outside of the polyp area) inside the MC dropout predictions indicate that these predictions have a higher mean DSC score. The results given in Table 4.1 and Table 4.2 support this observation, where the MC dropout ensemble have a mean DSC of 0.7505, whereas the deep ensemble have a mean DSC of 0.8034.

When observing the uncertainty representations in Figure 4.6, we first notice that the uncertainty estimates from the MC dropout ensemble are not prominent when compared to the uncertainty estimates given by the deep ensemble. Again, the areas which correspond to non-zero uncertainty values given by MC dropout are focused on the border of the predicted polyp, like what we observed in Figure 4.5. Additionally, we observe uncertainty values at the areas which correspond to noise in the predicted masks, indi-

cating a disagreement in the confidence between the models in the ensemble. For the uncertainty representations given by the deep ensemble, we see different values of uncertainty estimates across the entire uncertainty estimation, except for the area which corresponds to the inside the predicted polyp in the predicted mask. High values of uncertainties are not only gathered around the polyp border, but also around the black or green anchor object in the bottom left corner of the input image. We also notice that the largest areas of zero uncertainty are located inside the predicted polyp mask and inside the anchor object, indicating that the deep ensemble recognizes these objects and is able to correctly classify these. This indication is supported by the fact that these areas are correctly classified in the predicted segmentation mask.

5.2.3 Comparing DSC and BCE for U-Net

We observe the predicted masks and the uncertainty representations given by the MC dropout method with U-Net used as baseline. We focus on the comparison between the MC dropout ensemble trained with DSC and for the MC dropout ensemble trained with BCE. These corresponding results are gathered in Figure 4.10. We see that there is little difference between the uncertainty representations from using two different loss functions due to their elusive nature. For both cases, we see that the few pixels corresponding to uncertainty values given by the DSC-trained MC dropout ensemble have higher values (yellow pixels) than the uncertainty estimates given by BCE-trained MC dropout ensemble (cyan or light blue pixels). However, for the predicted masks, we see a difference in quality between the two MC dropout ensembles. We see this quality difference because the predicted masks have small areas of FPs branching out from the TP areas for the ensemble trained with DSC. In contrast, the ensemble trained with BCE give predicted masks that seem closer to the ground truth but contain more noise. Despite these differences, the DSC score during test time are highly similar for the two MC dropout ensembles, with DSC scores equal to 0.7538 and 0.7505 given in Table 4.1.

We examine the results from the deep ensemble using U-Net as baseline trained with DSC and the results from the deep ensemble trained with BCE in Figure 4.10. Based on these results, we observe the difference in quality between the two approaches given in the predicted masks. The predicted masks that correspond to the BCE-trained ensemble appear to be highly similar to the ground truth masks, and the predicted masks given by the DSC-trained ensemble are not as close to the ground truth. This is because they have small regions of FP pixels branching far outside the TP regions. Despite these misclassified pixels, the mean DSC scores in Table 4.2 are not representative in these six examples. Additionally, we

observed numerous prediction masks containing noise for each of the individual models in the BCE-trained deep ensemble. Since this noise is missing in the mean predictions, we know that the individual models in the ensemble were able to cancel out this noise. Consequently, we see large values of uncertainties spread across the uncertainty representations. To investigate this further, we would like to display each of the individual predictions provided by a deep ensemble to see how the noise is canceled out. We were not able to display these results this way due to time restrictions. Additionally, we believe that MC dropout are not sufficient at canceling out this noise due to the little diversity between each model in the ensemble.

We examine the uncertainty estimates given from the U-Net based deep ensemble in Figure 4.5 with DSC as loss, and for the corresponding ensemble but trained with BCE as loss in Figure 4.6. The uncertainty estimates with DSC are concentrated around in the areas which correspond to the borders of the predicted polyp mask. For the BCE-trained ensemble, the areas of high uncertainties appear far outside the boundary of the predicted polyp mask as well. We suspect that these high uncertainty estimations arise from the white writing in the left side of the image and the glares from the light of the endoscopy probe. Despite these high uncertainty estimates in these areas, the ensemble cancels out these probabilities which may correspond to misclassified pixels after thresholding.

In Figure 4.5 and Figure 4.6, we see that the predicted uncertainty masks for theBCE-trained ensembles are closer to the ground true masks compared to the predictions given by the DSC-trained ensembles, which is regardless of the uncertainty estimation method. The overall mean DSC score contradicts this observations as the DSC-trained ensembles gave the highest DSC score compared to the corresponding BCE-trained ensembles. We must consider the fact that the sets of three and three predictions that we displayed in these figures are not representative of the entire batch.

5.3 Deep Ensemble and MC Dropout with ResUNet++

We discuss our results given in Section 4.2 and Section 4.3, which contains the predicted segmentation masks and their corresponding uncertainty representations based on the MC dropout and the deep ensemble uncertainty estimation methods. This section only focuses on the ResUNet++ based ensembles in each of the uncertainty estimation methods.

5.3.1 ResUNet++ Trained with DSC

From examining the results from the DSC-trained ensembles in Figure 4.7, we notice that the predicted masks corresponding to the deep ensemble have more FPs as compared to the number of FNs. For the predictions given by the MC dropout method, the leftmost image has more FNs compared to FPs, but for the other two predictions it appears to be the opposite case. It is challenging to estimate which of these sets of predictions correspond to the highest mean DSC score based on these two sets of results alone, but from Table 4.1 and Table 4.2, the deep ensemble predictions give a mean DSC of 0.7370, and the MC dropout ensemble give a mean DSC of 0.7190. These scores indicate that the deep ensemble of 16 models gave the best performance when compared to 16 MC dropout models.

Regarding the uncertainty representations in Figure 4.7, we see the same pattern as observed in Figure 4.5 and Figure 4.6, where the uncertainty estimates given by MC dropout are unclear and difficult to interpret due to their elusive nature. We also notice that, for the rightmost uncertainty image, there are some uncertainty estimates in some of the areas which correspond to the predicted FNs. Note that this is not consistent for the other uncertainty representations, more specifically for the leftmost uncertainty image, where some of the areas of zero uncertainty values correspond to FN predictions. We acknowledge that this is the least desirable outcome, where we have misclassified pixels with corresponding low uncertainties indicating an overall high confidence among the models in the ensemble.

For the uncertainty representations given by the deep ensemble in Figure 4.7, we see several large areas corresponding to high uncertainty values. For the leftmost deep ensemble uncertainty representation, the large areas of uncertainty, specifically the red pixels, correspond to FN predictions in the predicted mask. We see that the majority of the pixels corresponding to large uncertainties are misclassified. Other pixels of large uncertainty are, however, correctly classified. For the rightmost uncertainty representation, we especially notice large uncertainty values both far outside and at large areas near the border of the actual polyp. Additionally, the majority of pixels which correspond to the actual polyp have zero uncertainty values. We see a similar connection for the results given by the DSC-trained U-Net based deep ensemble as observed in Figure 4.5. These results strongly indicate that the pixels which correspond to the largest values of uncertainties are more likely to be misclassified based on these results given by the deep ensemble method. In contrast to the MC dropout ensemble, the deep ensemble provide a is a desirable property by assigning low uncertainties both inside the polyp and the background, i.e., obvious TPs and TNs, and high uncertainties to the FPs and FNs. This is useful information to a medical expert, as the model appears more critical

towards the misclassified regions.

5.3.2 ResUNet++ Trained with BCE

In Figure 4.8 we see the results given by the ResUNet++ based ensemble using MC dropout and deep ensemble. For the predicted masks and uncertainty representation given by the MC dropout method, we see large areas of zero uncertainty. These areas correspond pixels in the predicted segmentation mask that appear to be over-segmented and thus have FP predictions outside the border corresponding to TP, i.e., where the actual polyp is located. Large areas of FP corresponding to over-segmentation have a corresponding zero to low uncertainty in the uncertainty representation, which indicates an overconfident model. This is undesirable as the property of overconfidence makes predictions models unable to generalize onto unseen data instances. Users can benefit from properly quantified uncertainties, by increased trust and reliability to the model predictions. This prerequisite proper uncertainty estimation methods, which do not assign high confidence to poor predictions. Unfortunately, we see this drawback in MC dropout based on an overconfident baseline model.

From examining the predictions and uncertainty representations given by deep ensemble in Figure 4.8, we see predicted TPs corresponding to the border of the ground truth polyp masks, but the model is unable to predict the inside of these borders. The result is a border of TPs filled with FNs. However, we observe a spread of large uncertainty values inside of these borders. Additionally, these correctly predicted borders correspond to zero uncertainty values in the uncertainty representations. We do not know if each of individual models in the ensemble were able to recognize the true polyp border, or if they were able to perform this way due to their diversity. This level of diversity can cancel out incorrectly classified predictions. Additionally, we see that the models largely focuses on the recognized edges in the prediction masks. Not only do we see this in the predictions, but further so in the uncertainty representations containing uncertainties at the many edges both around the polyp or in the background. Moreover, despite the poor quality in the predicted segmentation masks, the uncertainty estimates reflect a connection to the prediction masks which we have seen earlier in previous experiments with deep ensembles (see Figure 4.5 and Figure 4.7). More specifically, there is a connection between the large uncertainty estimates in the incorrectly predicted areas, and the zero (or low) uncertainty estimates corresponding to the correctly predicted areas.

5.3.3 Comparing DSC with BCE for ResUNet++

The resulting predictions and uncertainty estimates from the ensembles using ResUNet++ as the baseline model are displayed in figures 4.7 and 4.8. These results are displayed together in Figure 4.11. Our first observation is that the predictions given by the DSC-trained ensembles are of much higher quality than the BCE-trained ensembles. This is also supported by the results in tables 4.1 and 4.2, where the DSC-trained ensemble give a mean DSC score of 0.7190 and 0.7370 for the MC dropout ensemble and the deep ensemble, respectively. In contrast, the BCE-trained ensembles give a mean DSC score of 0.2850 and 0.3346. The uncertainty estimates that correspond to these low scores are given in Figure 4.8. In this figure, we observe are high values of uncertainty across the entire uncertainty representation, except the areas inside or near the ground truth, i.e., where the polyps are located. We argue that an ensemble that contains underfitted models that are unable to learn feature representations from the training dataset, should be able to produce large uncertainty estimates. This reflects the fact that the models make predictions that are far from the ground truth.

For the predicted masks given by the MC dropout ensemble given in Figure 4.11, we see the difference in quality where the DSC-trained model is far superior to the predictions given by the BCE-trained model. The former model gave a mean DSC score of 0.7190, whereas the latter gave a mean DSC of 0.2850. These poor-quality predictions correspond to a model that was able to obtain a low DSC score of 0.4379 during training. These results strongly indicate that the baseline model is underfitted, which means that it was unable to learn the feature representations that characterize the polyp.

Regarding the uncertainty representations from the MC dropout ensembles trained with different loss functions in Figure 4.11, we notice that the areas of uncertainty by the DSC-trained model do not have a clear connection to the pixels that are incorrectly classified. This observation is supported by the fact that several FN and FP pixels have a corresponding zero uncertainty in the uncertainty representation. However, it appears to be a clear connection between the ground truth polyp border, including other borders that, for the untrained eye, seem similar to that of a polyp border (see the second input image from the left in Figure 4.7), and the areas of moderate to low uncertainty values.

For the results given by the BCE-trained MC dropout model in Figure 4.11, we see a connection between the areas of high uncertainty and the predicted TNs. We also observe a similar connection between the areas of low to zero uncertainty and the FPs. For an ideal case of having a prop-

erly trained segmentation model that is neither underfitted or overfitted, one expects its corresponding uncertainty representations to have high uncertainty estimates for the incorrectly predicted areas (i.e., FNs and FPs), and low uncertainty for the correct predictions (i.e., TPs and TNs). Since the uncertainty representations lack this property, we conclude that these estimations give little to no value because the prediction model is overconfident in the misclassified predictions. We suspect that the reason in this case, is that the model is underfitted. These models are considered to be unable to learn the right associations by focusing on the irrelevant features. Hence, this may be the reason why the low uncertainty estimates are focused on the incorrect predictions and why the large uncertainty estimates are focused on the correct predictions. We are curious to see how models intentionally trained to be overfitted and underfitted affect the uncertainty representations, which leaves an open research topic for future investigation.

Observing the results by the DSC-trained ResUNet++ based deep ensembles in Figure 4.11, we see areas of large uncertainty estimates concentrated around the predicted segmentation mask and that some of these areas are more likely to be incorrectly classified. The background area and most of the area inside of the predicted polyp mask have zero uncertainty. In contrast, the BCE-trained ensemble in the images below have uncertainty estimates spread all over the image. There is a clear connection between the predicted TP and the areas of zero uncertainty. The areas which correspond to FPs have corresponding large uncertainty estimates. Since the ensemble correctly predicts the borders of the polyps and because of the connection between incorrect predictions and large uncertainty estimates, we suspect that the deep ensemble was somewhat able to learn some of the features corresponding to the polyps. We suspect that these learned features correspond to the border or edges of the polyp because of the correctly predicted areas in the predicted mask.

Based on the uncertainty estimates and the mean DSC score during test time, the ResUNet++ ensembles trained with BCE indicate underfitting. We were unsuccessful in tuning the hyperparameters for these models to converge for the number of epochs we were restricted to due to time constraints. Using extensive hyperparameter tuning techniques such as grid search is a topic for further investigation. We notice that large areas of high uncertainty are common in both cases, i.e., for the MC dropout ensemble and the deep ensemble trained with BCE. In contrast to the deep ensemble, there is no connection between the incorrect predictions and the large uncertainty estimates provided by MC dropout. Instead, we observe a connection between the incorrect predictions and the low uncertainty estimates, and between the correct predictions and the high uncertainty estimates. We conclude that this unwanted behavior is due to a highly

underfitted model.

5.4 Performance and Comparison of MC Dropout and Deep Ensembles

Regarding the performance in terms of the DSC during test time by the U-Net based ensembles in Figure 4.12, we notice a large increase in terms of the mean DSC for the two deep ensembles. In contrast, this behavior does not apply for the two MC dropout ensembles which appear to have approximately the same DSC score for each ensemble size ranging from 1 to 16. For the MC dropout ensembles and deep ensembles, using DSC as loss during training corresponds to the highest performance for both ensemble types even though this function is not defined as a proper scoring rule. For the DSC-trained deep ensemble, the optimal ensemble size is 7, which corresponds to the overall highest DSC during test time. Although, ensemble sizes of 6–16 give similar performance. For the BCE-trained deep ensemble, the optimal ensemble size is equal to 13.

For the performance of the ResUNet++ based ensembles in Figure 4.13, we notice a similar behavior as for the U-Net based ensembles where the DSC score varies for the ensemble size in deep ensembles, but this behavior do not apply to the MC dropout ensembles. The optimal ensemble size is equal to 13 and 5 DSC- and BCE-trained deep ensembles, respectively. Regardless of the uncertainty estimation method and baseline model, we see that the DSC-based ensembles give an overall higher performance than the BCE-trained ensembles.

Independent of the ensemble type, the ensembles based on the ResUNet++ architecture perform worse than the U-Net based ensembles. The results from the original ResUNet++ paper [66] contradict our results as they were able to obtain a higher DSC score for the ResUNet++ compared to the U-Net model. Both were trained and tested based on the exact same dataset, Kvasir-SEG. As we performed relatively little hyperparameter tuning for the two model architectures, we know that these contradictory results leave a topic for further investigation. However, for this specific configuration of hyperparameters, U-Net baseline model obtained the highest performance in all cases. To further investigate this, we need to reproduce the results given in the original ResUNet++ paper if possible.

Our results strongly suggests that the MC dropout method does not increase the performance with increasing ensemble size. With this considered, we argue that the purpose of utilizing MC dropout for this specific use-case is mainly to extract spatial uncertainty representations. We are

unsure how the different ensemble sizes will directly affect the quality of the uncertainty representations provided by MC dropout. This is because all uncertainty representations in this thesis are based on an ensemble size of 16. Despite, increasing the ensemble size for MC dropout is not computationally expensive in terms of memory and time as we only save and load one single model. On the other hand, using deep ensembles gives the benefit of both increasing the performance during test time and the quality of the uncertainty representations. The main drawback is time consumption if the models are trained one at a time, or the large demand for hardware (GPUs) if the models are to be trained in parallel.

We argue that the increasing performance provided by deep ensembles is caused by the diverse nature among the models in the ensemble. We can observe this diversity between each prediction in Figure 4.9. Since the models are separately initialized and trained, it is expected that the models converge to a similar endpoint after training. The separate training can allow these models to obtain a completely different set of weights. This is analogous to how two individuals can arrive at the same solution using different thinking strategies. With this separate training scheme, we intuitively know that the number of candidate loss surfaces during training will increase with the ensemble size. We argue that this level of diversity is the main contribution behind the increasing performance, which depends on the increasing ensemble size. This is because a diverse set of models are collectively more likely to arrive closer to the optimal convergence point. In contrast, we argue that the models in a MC dropout ensemble are highly correlated because each model is generated based on the one trained baseline model. Thus, each model in the ensemble is based on the exact same loss surface during training and has no other candidate functions to increase the chance of arriving at the optimal convergence point. The models in the MC dropout ensemble are not separately trained as for the models in the deep ensemble, which prerequisites a less diverse set of predictions. We argue that the lack of diversity in the predictions provided by MC dropout is the reason to why this approach is outperformed by deep ensembles.

In addition to arguing that the diversity of an ensemble increases the total performance, we also argue that the level of diversity between the models across an ensemble highly affects the quality of the uncertainty representations. Intuitively, less model diversity in the ensemble means leads to less diverse predictions. As we already have seen for the resulting uncertainty representations by MC dropout, the resulting uncertainties between the models in the ensemble become obscure and elusive. These unclear uncertainty representations give little information and are therefore hard to interpret. However, for uncertainty representations provided by deep ensembles, the information given in these images indicates what parts of the

image the ensemble struggles to recognize and correctly classify. These parts are also more likely to be incorrectly classified. Considering this correlation, we argue that these uncertainty representations are intuitive to visually assess and interpret. For the predicted segmentation masks given by deep ensembles, the diverse set of models across the ensemble combine their strengths and are hence able to cancel out their weaknesses. For example, if one model incorrectly classifies a pixel where the predicted prethresholded probability falls just below the threshold of 0.5, the resulting prediction will be incorrectly classified. If the other models in the ensemble, however, predict a probability above this threshold, the incorrect prediction will be canceled out by the other models. The resulting predictions correspond to a TP pixel. This is what we mean when we argue that a diverse set of models in an ensemble can cancel out their weaknesses. We also use this argument to explain the reason why the performance stabilizes right below the maximum DSC score when increasing the ensemble size seen in Figure 4.12 and Figure 4.13.

MC can also become unfavorable in applications where one does not want to include dropout layers to the model architecture. Regularization by dropout is not always wanted and can thus become challenging when lacking the computational power and memory to facilitate performance increase and uncertainty estimation by deep ensembles.

Considering all the above arguments, we conclude that the deep ensemble approach is more favorable compared to the MC dropout method. This is because it is able to provide interpretable explanations behind the model predictions. In addition, they hold the property of diversity between the models in the ensemble which can increase the total performance when increasing the ensemble size. Deep ensembles have the drawbacks of being more computationally expensive and time consuming compared to the MC dropout approach.

5.5 Problem Statement Revisited

We will revisit our research questions which were defined in Chapter 1 by first discussing our research objectives. We will repeat these and sequentially explain how we reached these research objectives.

Objective 1: *Explore how uncertainty estimates connect the model prediction with the input features.*

This research objective is supported by the fact that we were able to explore relations between the uncertainty representations and the predicted segmentation masks with different uncertainty estimation approaches.

For the deep ensemble approach, we found that the incorrectly classified areas in the predicted segmentation masks have a corresponding high uncertainty estimation. We were not able to establish similar relations for the properly trained MC dropout ensembles. Despite the uncertainty estimation approach used, we were able to find that the areas containing values of uncertainty estimates were concentrated around the border of the predicted segmentation mask.

Objective 2: *Explore how the use of different loss functions affect the model predictions and the corresponding uncertainty.*

We were able to reach the second objective by exploring two different loss functions, DSC and BCE, during training for different ensemble types. By doing so, we were able to explore uncertainty estimation techniques based on different model training characteristics. Overall, the DSC-trained ensembles outperformed the BCE-trained ones in terms of the mean DSC score during test time,, even though BCE is categorized as a proper scoring function. In addition, the predicted segmentation masks contained more noise for BCE-trained ensembles regardless of the type of uncertainty estimation approach. However, the deep ensemble was able to cancel out a majority of this noise which we experienced for each of the individual models in the ensemble. As a result of noise, the uncertainty estimates of varying size was spread across the uncertainty representations.

For the DSC-trained ensembles, we observe more outlying FP pixels branching outside of the TP areas. For the uncertainty representations, there are larger areas of uncertainty estimates spread across the entire uncertainty representation when using with the BCE-trained deep ensemble. These values correspond to mainly moderately low uncertainty values. In contrast, for the DSC-trained deep ensemble we see more concentrated areas of large uncertainty estimates around the predicted segmentation masks. For MC dropout, we observe little difference between the uncertainty representations provided by the DSC-trained ensemble and the BCE-trained ensemble. All these observations summarize our main findings after reaching the second research objective.

Objective 3: *Use two different architectures to examine if the uncertainty estimation performance is affected by the model architecture.*

This third research objective was reached by comparing the performance in terms of the mean DSC during test time for the two uncertainty estimation techniques, using two different CNN architectures, U-Net and ResUNet++. All ensembles based on the U-Net architecture outperformed the ensembles based on ResUNet++, regardless of the uncertainty estimation method used. These results contradict the results given in previous

studies and since our findings are based on the lack of extensive hyper-parameter tuning techniques, we believe that this leaves a new research objective that can be further investigated in future research.

We have now revisited the research objectives defined at the start of this thesis. The aim behind these objectives was to aid us in answering the main research question. Answering **RQ1** leaves the reader with the main contributions from the conducted research. We repeat our main research question:

RQ1: *How can the predictive uncertainty estimates improve the understanding of the model prediction?*

We answer this question by testing each combination of two uncertainty estimation techniques, trained with different loss functions, and based on two different CNN architectures. We argue that the uncertainty representations provided by the deep ensemble approach can provide intuitive explanations that can be interpreted by non-DL experts. This is because they contain information about areas that are more or less likely to be incorrectly classified by the DL model. Pixels in the predicted segmentation mask that have corresponding large uncertainty estimates are more likely to be incorrectly classified. Additionally, the pixels corresponding to low uncertainty estimates are more likely to be correctly classified. We claim that these correlations between the prediction and its corresponding uncertainty representation is information that a typical end-user will expect based on the general understanding of uncertainty. Based on this claim, we conclude that these uncertainty representations are easy to interpret and hence provide useful explanations to non-DL experts and can therefore help to improve the understanding behind the model prediction.

With the results given by the MC dropout approach, we found that the resulting uncertainty estimates are both elusive and hence difficult to interpret. Additionally, we did not find clear correlations between the uncertainty estimates and the pixel areas that are incorrectly predicted in the corresponding predicted segmentation masks. We thus conclude that this approach is insufficient at providing explanations to the model predictions, and therefore lack at improving understating behind DL-based model predictions to users.

We answer this research question through testing and extending two popular uncertainty estimation techniques by comparing them in a segmentation use-case. This answer lays the foundation for the main contributions given in this thesis. Reaching our research objectives solves our research question by testing uncertainty estimation techniques in different scenarios. These scenarios include the use of different loss functions and model

architectures. We claim that these results bring value to the field of DL and XAI, and is therefore consistent with the motivation stated in Chapter 1. This is supported by the fact that we were able to show how segmentation models can detect polyps based on colonoscopy imaging and additionally provided informative explanations behind the model output in the form of uncertainty representations given by *one* of the uncertainty estimation techniques that we tested.

5.6 Summary

In this chapter, we have pointed out and discussed our main findings from the results presented in Chapter 4. First, we analyzed and discussed the reasoning behind the experiments of tuning the dropout rate in Section 5.1. For all models, we ended up using the dropout rate at the epoch which corresponded to the maximum DSC during validation.

Furthermore, in Section 5.2 we describe the main findings from the two uncertainty estimation techniques, MC dropout and deep ensembles, with the use of U-Net as the baseline model. We compare the two uncertainty estimation techniques by analyzing and discussing their corresponding predicted segmentation masks and the uncertainty estimations. Lastly, we compared the use of different loss functions, BCE and DSC, by observing the quality in predictions and uncertainty representations. Despite the uncertainty estimation technique, the use of DSC during training resulted in the highest mean DSC score during test time in all cases.

In Section 5.3, we analyzed the main findings from using the ResUNet++ architecture as the baseline model for the two uncertainty estimation approaches. We focus on discussing the main differences in the predictions and uncertainty representations from these two approaches. We also claim to see large quality differences in using the different loss functions during training.

Moreover, we focus on the difference in performance and the overall comparison between the two uncertainty estimation techniques in Section 5.4. The main conclusions based on our findings are contained within this section.

Lastly, in Section 5.5 we revisit the research objectives and main research question which we defined at the beginning of this thesis. We discuss how the conducted experiments, and their results meet these objectives and answer the research question based on our main findings.

Chapter 6

Conclusion

This chapter summarizes the conclusions based on our main findings. We repeat and answer our problem statement defined in Section 1.2, which lays the foundation for the main contributions given by this thesis. Lastly, we discuss our views on the potential future work which can be extended from the research presented in this thesis.

6.1 Summary

DL-based CAD systems can bring significant value to clinical workflows by decreasing the number of observational oversights in diagnostic medicine. These systems are considered "black boxes" due to their lack of explainability and transparency to their decision-making process. Considering these issues, medical experts lack knowledge and trust in the model output. Not only are these attributes important in high-risk applications where the model output can determine the future of human life and health, but they can include human observers into the decision-making process by revealing potential systematic biases or inappropriate correlations between the disease and outcome. Therefore, explainability can help to prevent potential disasters in the future. By being motivated by the general lack of explainability methods in DL, we adapted and extended two existing popular uncertainty estimation techniques by comparing them in a segmentation use-case. We included the use of two different CNN architectures and two different loss functions for further comparison.

We presented visual uncertainty representations as heatmaps based on segmentation masks that highlight polyps from colonoscopy images. We trained, validated and tested two CNN models based on the Kvasir-SEG dataset. For uncertainty estimation, we adapted MC dropout and deep ensembles and compared them. Additionally, we included two different loss metrics, DSC and BCE, for training each model. Later on, we answer

our research question by testing each combination of the two uncertainty estimation techniques, trained with different loss functions, and based on two different CNN architectures.

Our results show that MC dropout was insufficient at providing interpretable uncertainty representations and lacks at connecting the uncertainty estimates at the misclassified regions. This method was not able to increase the performance during test time, which contradicts with the results given in previous studies. However, when using the second technique, we have shown that this method was able to provide visually interpretable and informative uncertainty representations by connecting large uncertainty estimates to predictions that are more likely to be misclassified. MC dropout was unsuccessful at providing such correlations. In addition, the deep ensembles were able to obtain higher performance during test time when increasing the ensemble size. Our main results consist of predicted segmentation masks and their corresponding uncertainty representations.

6.2 Contributions

We revisit the main contributions stated in Section 1.6 by repeating and answering **RQ1**. Below, we sequentially repeat and answer these research objectives, which helps solve our main research question.

Objective 1: *Explore how uncertainty estimates connect the model prediction with the input features.*

This objective is supported by our exploration of different uncertainty estimation methods in a segmentation use-case. We established connections in the information provided by the uncertainty representations and their corresponding predictions. These connections only apply to one of the two approaches that we tested, the deep ensemble method. The other approach, MC dropout, was insufficient at establishing such correlations, and therefore did not provide useful and informative uncertainty estimates.

Objective 2: *Explore how the use of different loss functions affect the model predictions and the corresponding uncertainty.*

Including the use of different loss functions, DSC and BCE, during training, allowed us to meet this objective. By doing so, we were able to explore uncertainty estimation techniques based on different model training characteristics. Overall, using DSC as loss during training resulted in a model that outperformed the corresponding BCE-trained models. Not only did

DSC provide better predictions in terms of the test score, but in contrast to BCE, it also provided clearer uncertainty estimates corresponding to areas that are more or less likely to be correctly classified.

Objective 3: *Use two different architectures to examine if the uncertainty estimation performance is affected by the model architecture.*

Like the second objective, this third objective aims at testing uncertainty estimation methods based on different model characteristics. In this case, these characteristics correspond to different CNN architectures. This objective is supported by the fact that we explored the use of different CNN architectures. By doing so, we investigated if and how each model affects the performance of the two uncertainty estimation approaches. Overall, U-Net outperformed ResUNet++, which contradicts the results presented in a previous study. This leaves a topic for further investigation due to the contradictory results.

Our work aims at extending explainability by providing predictive uncertainty estimates. As we see above, we proposed the three research objectives above to aid us in answering the following research question:

RQ1: *How can the predictive uncertainty estimates improve the understanding of the model prediction?*

We answer this question by testing each combination of two uncertainty estimation techniques, trained with different loss functions, and based on two different CNN architectures. The spatially distributed uncertainty estimations were visualized as heatmaps. We claim that the uncertainty representations provided by the deep ensemble approach give explanations that can be interpreted by non-DL experts because these uncertainty estimations act as our existing general understanding of uncertainty in science. To clearly answer our research question, we argue that uncertainty representations provided by deep ensembles are indeed able to improve the understanding of the model prediction based on our findings.

The other uncertainty estimation approach, MC dropout, was unable to improve the understanding of the model prediction by its uncertainty representations. This is because we found that the resulting uncertainty estimates were difficult to interpret. The information given in the uncertainty estimates did not correlate with the model predictions.

Reaching our objectives and answering the research question lay the foundation of the main contributions behind this thesis. We argue that these contributions add value to the field of XAI, which is consistent with the motivations behind this thesis stated in Section 1.1. Applying uncertainty

estimation to semantic segmentation shows that we were able to extract explanations capable of adding more information to the model predictions. These explanations can help to improve the understanding of the model prediction, which can be used to make more informed decisions on whether to trust a model's predictions. We hope that this work leaves motivation for further development for explainability- and uncertainty estimation methods to further increase the trust and acceptance of DL-based CAD in the future.

6.3 Future Work

Our main research question can be extended in potential future work. One can include a user study for uncertainty representations to be tested and evaluated by actual medical doctors in real clinical workflows. Together with experienced colonoscopists and other clinicians, we can study their user behaviors and needs when using automated polyp detection that provides uncertainty representations. This allows us to evaluate and establish the true potential of uncertainty estimations in a real clinical scenario. Additionally, we would like to use uncertainty estimation techniques to calibrate important parameters such as the threshold value used during model prediction, or the weight factor used in different loss functions like the weighted BCE. We would also like to study how the uncertainty estimates are affected if we change or remove specific layers or entire modules inside a CNN architecture. Extending existing uncertainty estimation methods using other voting techniques also leaves an interesting research topic. Lastly, we would like to develop our own novel uncertainty estimation technique, based on the main strengths of what we learned from these existing uncertainty estimation methods.

Bibliography

[1] Martín Abadi et al. "TensorFlow: A System for Large-Scale Machine Learning." In: *Proceedings of the 2016 USENIX Conference on Operating Systems Design and Implementation (OSDI)*. Vol. 16. 2016, pp. 265–283. DOI: 10.5555/3026877.3026899.

[2] Solon Barocas and Andrew D. Selbst. "Big Data's Disparate Impact." In: *SSRN eLibrary* (2014). DOI: 10.15779/Z38BG31.

[3] Atilim Gunes Baydin, Barak A. Pearlmutter, and Alexey Andreyevich Radul. "Automatic differentiation in machine learning: a survey." In: *CoRR* (2015). arXiv: 1502.05767.

[4] Jorge Bernal et al. "WM-DOVA maps for accurate polyp highlighting in colonoscopy: Validation vs. saliency maps from physicians." In: *Computerized Medical Imaging and Graphics* 43 (2015), pp. 99–111. DOI: 10.1016/j.compmedimag.2015.02.007.

[5] Hanna Borgli et al. "HyperKvasir, a comprehensive multi-class image and video dataset for gastrointestinal endoscopy." In: *Scientific Data* 7 (2020), p. 283. ISSN: 2052-4463. DOI: 10.1038/s41597-020-00622-y.

[6] Alan C. Bovik. "Chapter 3 - Basic Gray Level Image Processing." In: *The Essential Guide to Image Processing*. Ed. by Al Bovik. Academic Press, 2009, pp. 43–68. DOI: 10.1016/B978-0-12-374457-9.00003-2.

[7] Leo Breiman. "Random Forests." In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[8] Alexander Buslaev et al. "Albumentations: Fast and Flexible Image Augmentations." In: *Information* 11.2 (2020). DOI: 10.3390/info11020125.

[9] Tim Byers et al. "American Cancer Society guidelines for screening and surveillance for early detection of colorectal polyps and cancer: Update 1997." In: *CA: A Cancer Journal for Clinicians* 47.3 (1997), pp. 154–160. DOI: 10.3322/canjclin.47.3.154.

[10] Ronald Castellino. "Computer aided detection (CAD): an overview." In: *Cancer imaging : the official publication of the International Cancer Imaging Society* 5.1 (2005), pp. 17–9. DOI: 10.1102/1470-7330.2005.0018.

[11] 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 (2018), pp. 834–848. DOI: 10.1109/TPAMI.2017.2699184.

[12] Liang-Chieh Chen et al. “Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs.” In: (2016). arXiv: 1412.7062.

[13] Kenneth W. Clark et al. “The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository.” In: *J. Digit. Imaging* 26.6 (2013), pp. 1045–1057. DOI: 10.1007/s10278-013-9622-7.

[14] George Cybenko. “Approximation by superpositions of a sigmoidal function.” In: *mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314. DOI: 10.1007/BF02551274.

[15] C. Darken, J. Chang, and J. Moody. “Learning rate schedules for faster stochastic gradient search.” In: *Proceedings of the 1992 IEEE Workshop Neural Networks for Signal Processing (NNSP)*. 1992, pp. 3–12. DOI: 10.1109/NNSP.1992.253713.

[16] Yann N. Dauphin et al. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.” In: *CoRR* 2 (2014). DOI: 10.5555/2969033.2969154.

[17] Thomas Davidson, Debasmita Bhattacharya, and Ingmar Weber. “Racial Bias in Hate Speech and Abusive Language Detection Datasets.” In: *Proceedings of the 2019 Workshop on Abusive Language Online (WOAH)*. Association for Computational Linguistics, 2019, pp. 25–35. DOI: 10.18653/v1/W19-3504.

[18] “DeepLesion: automated mining of large-scale lesion annotations and universal lesion detection with deep learning.” In: *Journal of medical imaging* 5.3 (2018). DOI: 10.1117/1.JMI.5.3.036501.

[19] Jia Deng et al. “ImageNet: A large-scale hierarchical image database.” In: *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[20] Peter Denning et al. “Computing as a Discipline: Preliminary Report of the ACM Task Force on the Core of Computer Science.” In: *Proceedings of the Nineteenth SIGCSE Technical Symposium on Computer Science Education*. Association for Computing Machinery, 1988, p. 41. DOI: 10.1145/52964.52975.

[21] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In: 1 (2019), pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[22] Terrance DeVries and Graham W. Taylor. "Leveraging Uncertainty Estimates for Predicting Segmentation Quality." In: (2018). DOI: 10.48550/arxiv.1807.00502.

[23] Lee R. Dice. "Measures of the Amount of Ecologic Association Between Species." In: *Ecology* 26.3 (1945), pp. 297–302. DOI: 10.2307/1932409.

[24] Thomas G. Dietterich. "Ensemble Methods in Machine Learning." In: *Proceedings of the 2000 International Workshop on Multiple Classifier Systems (MCS)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15. DOI: 10.1007/3-540-45014-9_1.

[25] Kunio Doi. "Computer-aided diagnosis in medical imaging: Historical review, current status and future potential." In: *Computerized Medical Imaging and Graphics* 31.4 (2007), pp. 198–211. DOI: 10.1016/j.compmedimag.2007.02.002.

[26] Finale Doshi-Velez and Been Kim. "Towards A Rigorous Science of Interpretable Machine Learning." In: *arXiv: Machine Learning* (2017). arXiv: 1702.08608.

[27] Finale Doshi-Velez et al. "Accountability of AI Under the Law: The Role of Explanation." In: (2017). DOI: 10.48550/arxiv.1711.01134.

[28] Rosie Dunford, Quanrong Su, and Ekraj Tamang. "The Pareto Principle." In: *The Plymouth Student Scientist* 7 (2014), pp. 140–148. DOI: 10.4135/9781412950596.n394.

[29] Dumitru Erhan et al. "Visualizing Higher-Layer Features of a Deep Network." In: *Technical Report, Univeristé de Montréal* 1341 (2009).

[30] David M. Estlund. "Opinion leaders, independence, and Condorcet's Jury Theorem." In: *Theory and Decision* 36 (1994), pp. 131–162. DOI: 10.1007/BF01079210.

[31] Mark Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective." In: *International Journal of Computer Vision* 111 (2014), pp. 98–136. DOI: 10.1007/s11263-009-0275-4.

[32] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. "Deep Ensembles: A Loss Landscape Perspective." In: *ArXiv preprint* (2019). ArXiv: 1912.02757.

[33] Jonathan Frankle, David J. Schwab, and Ari S. Morcos. "Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs." In: *CoRR* (2020). arXiv: 2003.00152.

[34] Yoav Freund and Robert E. Schapire. "Experiments with a New Boosting Algorithm." In: *Proceedings of the 1996 International Conference on International Conference on Machine Learning (ICML)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 148–156. DOI: 10.5555/3091696.3091715.

[35] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning." In: *Proceedings of the 2016 International Conference on on Machine Learning (ICML)*. Vol. 48. 2016, pp. 1050–1059. DOI: 10.5555/3045390.3045502.

[36] Adrian Galdran, Gustavo Carneiro, and Miguel Ángel González Ballester. "Double Encoder-Decoder Networks for Gastrointestinal Polyp Segmentation." In: Feb. 2021, pp. 293–307. DOI: 10.1007/978-3-030-68763-2_22.

[37] Jakob Gawlikowski et al. "A Survey of Uncertainty in Deep Neural Networks." In: (2021). DOI: 10.48550/ARXIV.2107.03342.

[38] Stuart Geman, Elie Bienenstock, and René Doursat. "Neural Networks and the Bias/Variance Dilemma." In: *Neural Computation* 4.1 (1992), pp. 1–58. DOI: 10.1162/neco.1992.4.1.1.

[39] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019. ISBN: 978-1491962299.

[40] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva. "Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation." In: 2018.

[41] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the 2010 International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 9. Journal of Machine Learning Research - Proceedings Track, 2010, pp. 249–256.

[42] Tilmann Gneiting and Adrian Raftery. "Strictly Proper Scoring Rules, Prediction, and Estimation." In: *Journal of the American Statistical Association* 102 (Mar. 2007), pp. 359–378. DOI: 10.1198/016214506000001437.

[43] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples." In: *CoRR* (2015). arXiv: 1412.6572.

[44] Endre Grøvik et al. "Deep learning enables automatic detection and segmentation of brain metastases on multisequence MRI." In: *Journal of Magnetic Resonance Imaging* 51.1 (2019), pp. 175–182. DOI: 10.1002/jmri.26766.

[45] H.A. Haenssle et al. "Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists." In: *Annals of Oncology* 29.8 (2018), pp. 1836–1842. DOI: 10.1093/annonc/mdy166.

[46] Dianyuan Han. "Comparison of Commonly Used Image Interpolation Methods." In: *Proceedings of the 2013 International Conference on Computer Science and Electronics Engineering (ICCSEE)*. Atlantis Press, 2013, pp. 1556–1559. DOI: 10.2991/iccsee.2013.391.

[47] Kaiming He et al. "Deep Residual Learning for Image Recognition." In: vol. 1. 1. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[48] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *CoRR* (2015). DOI: 10.1109/ICCV.2015.123.

[49] Tao He et al. "MediMLP: Using Grad-CAM to Extract Crucial Variables for Lung Cancer Postoperative Complication Prediction." In: *IEEE Journal of Biomedical and Health Informatics* 24.6 (2020), pp. 1762–1771. DOI: 10.1109/JBHI.2019.2949601.

[50] Andreas Heinecke, Jinn Ho, and Wen-Liang Hwang. "Refinement and Universal Approximation via Sparsely Connected ReLU Convolution Nets." In: *IEEE Signal Processing Letters* 27 (2020), pp. 1175–1179. DOI: 10.1109/LSP.2020.3005051.

[51] Steven Hicks et al. "Dissecting Deep Neural Networks for Better Medical Image Classification and Classification Understanding." In: *Proceedings of the 2018 IEEE International Symposium on Computer-Based Medical Systems (CBMS)*. 2018, pp. 363–368. DOI: 10.1109/CBMS.2018.00070.

[52] Steven Hicks et al. "Explaining deep neural networks for knowledge discovery in electrocardiogram analysis." In: *Scientific Reports* (May 2021). DOI: 10.1038/s41598-021-90285-5.

[53] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors." In: *CoRR* (2012). arXiv: 1207.0580.

[54] Andreas Holzinger et al. "Causability and explainability of artificial intelligence in medicine." In: *WIREs Data Mining Knowl. Discov.* 9.4 (2019). DOI: 10.1002/widm.1312.

[55] Andreas Holzinger et al. "What do we need to build explainable AI systems for the medical domain?" In: (2017). DOI: 10.48550/arxiv.1712.09923.

[56] Jie Hu, Li Shen, and Gang Sun. "Squeeze-and-Excitation Networks." In: *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 7132–7141. DOI: 10.1109/CVPR.2018.00745.

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

[58] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *CoRR* (2015). arXiv: 1502.03167.

[59] Ali Işın, Cem Direkoğlu, and Melike Şah. "Review of MRI-based Brain Tumor Image Segmentation Using Deep Learning Methods." In: *Procedia Computer Science* 102 (2016), pp. 317–324. DOI: 10.1016/j.procs.2016.09.407.

[60] Paul Jaccard. "THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE." In: *New Phytologist* 11.2 (1912), pp. 37–50. DOI: 10.1111/j.1469-8137.1912.tb05611.x.

[61] Felicia Jacobsen. "Predictive Uncertainty Masks from Deep Ensembles for Automated Polyp Segmentation." In: *Proc. of MediaEval 2021 CEUR Workshop*. 2021.

[62] Shruti Jadon. "A survey of loss functions for semantic segmentation." In: *Proceedings of the 2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. 2020, pp. 1–7. DOI: 10.1109/CIBCB48159.2020.9277638.

[63] Andrew Janowczyk and Anant Madabhushi. "Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases." In: *Journal of pathology informatics* 7 (2016), p. 29. DOI: 10.4103/2153-3539.186902.

[64] Debesh Jha et al. "A Comprehensive Study on Colorectal Polyp Segmentation With ResUNet++, Conditional Random Field and Test-Time Augmentation." In: *IEEE Journal of Biomedical and Health Informatics* 25.6 (2021), pp. 2029–2040. DOI: 10.1109/JBHI.2021.3049304.

[65] Debesh Jha et al. "DoubleU-Net: A Deep Convolutional Neural Network for Medical Image Segmentation." In: 2020, pp. 558–564. DOI: 10.1109/CBMS49503.2020.00111.

[66] Debesh Jha et al. "ResUNet++: An Advanced Architecture for Medical Image Segmentation." In: 2019. DOI: 10.1109/ISM46123.2019.00049.

[67] Ge-Peng Ji et al. "Video Polyp Segmentation: A Deep Learning Perspective." In: (2022). DOI: 10.48550/arxiv.2203.14291.

[68] Daniel S. Kermany et al. "Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning." In: *Cell* 172.5 (2018), 1122–1131.e9. DOI: 10.1016/j.cell.2018.02.010.

[69] J. Kiefer and J. Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function." In: *The Annals of Mathematical Statistics* 23.3 (1952), pp. 462–466. DOI: 10.1214/aoms/1177729392.

[70] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *Proceedings of the 2015 International Conference on Learning Representations (ICLR)*. arXiv, 2015. DOI: 10.48550/arxiv.1412.6980.

[71] Yoriaki Komeda et al. "Computer-Aided Diagnosis Based on Convolutional Neural Network System for Colorectal Polyp Classification: Preliminary Experience." In: *Oncology* 93 (2017), pp. 30–34. DOI: 10.1159/000481227.

[72] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Commun. ACM* 60.6 (2017), pp. 84–90. DOI: 10.1145/3065386.

[73] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles." In: *Proceedings of the 2017 International Conference on Neural Information Processing Systems (NIPS)*. 2017, pp. 6405–6416. DOI: 10.5555/3295222.3295387.

[74] Sebastian Lapuschkin et al. "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation." In: *PLoS ONE* 10 (2015), e0130140. DOI: 10.1371/journal.pone.0130140.

[75] Sebastian Lapuschkin et al. "Unmasking Clever Hans Predictors and Assessing What Machines Really Learn." In: *CoRR* (2019). DOI: 10.1038/s41467-019-08987-4.

[76] Yann LecCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *Nature* 521 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[77] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition." In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[78] R.S. Ledley and L.B. Lusted. "Reasoning Foundations of Medical Diagnosis Symbolic logic, probability, and value theory aid our understanding of how physicians reason." In: *Science* 130.3366 (1959), pp. 9–21. DOI: 10.1126/science.130.3366.9.

[79] Qiaoliang Li et al. "Colorectal polyp segmentation using a fully convolutional neural network." In: *Proceedings of the 2017 International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. 2017, pp. 1–5. DOI: 10.1109/CISP-BMEI.2017.8301980.

[80] Petro Liashchynskyi and Pavlo Liashchynskyi. "Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS." In: (2019). DOI: 10.48550/arxiv.1912.06059.

[81] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network." In: *CoRR* (2014). arXiv: 1312.4400.

[82] Geert Litjens et al. "A survey on deep learning in medical image analysis." In: *Medical Image Analysis* 42 (Dec. 2017), pp. 60–88. DOI: 10.1016/j.media.2017.07.005.

[83] Xiaohua Long et al. "Clinical and endoscopic-pathological characteristics of colorectal polyps: an analysis of 1,234 cases." In: *Int. J. Clin. Exp. Med.* 8.10 (2015). PMID:26770577, pp. 19367–19373.

[84] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Warm Restarts." In: (2016). DOI: 10.48550/arxiv.1608.03983.

[85] Ange Lou et al. "CaraNet: context axial reverse attention network for segmentation of small medical objects." In: *Proceedings of the 2022 International Conference on Medical Imaging: Image Processing (ICMIIP)*. Vol. 12032. SPIE, 2022, pp. 81–92. DOI: 10.1117/12.2611802.

[86] Guo-Chun Lou et al. "A retrospective study on endoscopic missing diagnosis of colorectal polyp and its related factors." In: *The Turkish journal of gastroenterology* (2014), pp. 182–186. DOI: 10.5152/tjg.2014.4664.

[87] Jung Choi Mary Ann Clark Matthew Douglas. *Biology 2e*. OpenStax, 2018. ISBN: 1947172522.

[88] Michael W. Rabow Maxine A. Papadakis Stephen J. McPhee. *CURRENT Medical Diagnosis & Treatment*. McGraw-Hill Education, 2019. ISBN: 9781260117448.

[89] Alireza Mehrtash et al. "Confidence Calibration and Predictive Uncertainty Estimation for Deep Medical Image Segmentation." In: *IEEE Transactions on Medical Imaging* 39.12 (2020), pp. 3868–3878. DOI: 10.1109/tmi.2020.3006437.

[90] Paulius Micikevicius et al. *Mixed Precision Training*. 2017. DOI: 10.48550/arxiv.1710.03740.

[91] Xi Mo et al. "An Efficient Approach for Polyps Detection in Endoscopic Videos Based on Faster R-CNN." In: (2018). DOI: 10.48550/ARXIV.1809.01263.

[92] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. URL: https://christophm.github.io/interpretable-ml-book.

[93] Eric Nalisnick et al. "Do Deep Generative Models Know What They Don't Know?" In: (2018). DOI: 10.48550/ARXIV.1810.09136.

[94] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. "Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks." In: *CoRR* (2016). arXiv: 1602.03616.

[95] Kim NH et al. "Miss rate of colorectal neoplastic polyps and risk factors for missed polyps in consecutive colonoscopies." In: *Intestinal research* 15.3 (2017). DOI: 10.5217/ir.2017.15.3.411.

[96] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. DOI: 10.48550/ARXIV.1511.08458.

[97] Ziad Obermeyer et al. "Dissecting racial bias in an algorithm used to manage the health of populations." In: *Science* 366.6464 (2019), pp. 447–453. DOI: 10.1126/science.aax2342.

[98] Augustus Odena, Vincent Dumoulin, and Chris Olah. "Deconvolution and Checkerboard Artifacts." In: *Distill* (2016). DOI: 10.23915/distill.00003.

[99] Rukundo Olivier and Cao Hanqiang. "Nearest Neighbor Value Interpolation." In: *International Journal of Advanced Computer Science and Applications* 3.4 (2012). DOI: 10.14569/ijacsa.2012.030405.

[100] Yaniv Ovadia et al. "Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty under Dataset Shift." In: *Proceedings of the 2019 International Conference on Neural Information Processing Systems (NIPS)*. Curran Associates Inc., 2019. DOI: 10.48550/arxiv.1906.02530.

[101] Tsuyoshi Ozawa et al. "Automated endoscopic detection and classification of colorectal polyps using convolutional neural networks." In: *Therapeutic Advances in Gastroenterology* 13 (2020), p. 1756284820910659. DOI: 10.1177/1756284820910659.

[102] Harsh Panwar et al. "A deep learning and grad-CAM based color visualization approach for fast detection of COVID-19 cases using chest X-ray and CT-Scan images." In: *Chaos, Solitons & Fractals* 140 (2020), p. 110190. DOI: 10.1016/j.chaos.2020.110190.

[103] Adam Paszke et al. "Automatic Differentiation in PyTorch." In: *Proceedings of the 2017 International Conference and Workshop on Neural Information Processing Systems (NIPS) on Autodiff*. 2017.

[104] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Proceedings of the 2019 International Conference on Advances in Neural Information Processing Systems (NIPS)*. Vol. 32. Curran Associates, Inc., 2019. DOI: 10.5555/3454287.3455008.

[105] Luis Perez and Jason Wang. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. 2017. DOI: 10.48550/arxiv.1712.04621.

[106] Konstantin Pogorelov et al. "KVASIR: A Multi-Class Image Dataset for Computer Aided Gastrointestinal Disease Detection." In: *Proceedings of the 2017 ACM on Multimedia Systems Conference (MMSys)*. 2017, pp. 164–169. DOI: 10.1145/3083187.3083212.

[107] Boris Polyak. "Some methods of speeding up the convergence of iteration methods." In: *Ussr Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17. DOI: 10.1016/0041-5553(64)90137-5.

[108] Justin Tyler Pontalba et al. "Assessing the Impact of Color Normalization in Convolutional Neural Network-Based Nuclei Segmentation Frameworks." In: *Frontiers in Bioengineering and Biotechnology* 7 (2019). DOI: 10.3389/fbioe.2019.00300.

[109] Hemin Ali Qadir et al. "Improving Automatic Polyp Detection Using CNN by Exploiting Temporal Dependency in Colonoscopy Video." In: *IEEE Journal of Biomedical and Health Informatics* 24.1 (2020), pp. 180–193. DOI: 10.1109/JBHI.2019.2907434.

[110] Ning Qian. "On the momentum term in gradient descent learning algorithms." In: *Neural Networks* 12.1 (1999), pp. 145–151. DOI: 10.1016/S0893-6080(98)00116-6.

[111] Vijay Rao et al. "How widely is computer-aided detection used in screening and diagnostic mammography?" In: *Journal of the American College of Radiology* 7.10 (2010), pp. 802–805. DOI: 10.1016/j.jacr.2010.05.019.

[112] Stephan Rasp and Nils Thuerey. "Data-Driven Medium-Range Weather Prediction With a Resnet Pretrained on Climate Simulations: A New Model for WeatherBench." In: *Journal of Advances in Modeling Earth Systems* 13.2 (2021). DOI: 10.1029/2020MS002405.

[113] Annika Reinke et al. "Common Limitations of Image Processing Metrics: A Picture Story." In: (2021). DOI: 10.48550/ARXIV.2104.05642.

[114] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method." In: *Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. DOI: 10.1214/aoms/1177729586.

[115] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *CoRR* 9351 (2015). Ed. by Nassir Navab et al., pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

[116] Santanu Roy, Shyam Lal, and Jyoti R. Kini. "Novel Color Normalization Method for Hematoxylin amp; Eosin Stained Histopathology Images." In: *IEEE Access* 7 (2019), pp. 28982–28998. DOI: 10.1109/ACCESS.2019.2894791.

[117] Sebastian Ruder. "An overview of gradient descent optimization algorithms." In: *CoRR* (2016). arXiv: 1609.04747.

[118] Wojciech Samek and Klaus-Robert Müller. "Towards Explainable Artificial Intelligence." In: *Lecture Notes in Computer Science* (2019), pp. 5–22. DOI: 10.1007/978-3-030-28954-6_1.

[119] Shibani Santurkar et al. "How Does Batch Normalization Help Optimization?" In: *Proceedings of the 2018 International Conference on Neural Information Processing (NIPS)*. 2018. eprint: 1805.11604.

[120] Anton Maximilian Schäfer and Hans Georg Zimmermann. "Recurrent Neural Networks Are Universal Approximators." In: *Proceedings of the 2006 International Conference on Artificial Neural Networks (ICANN)*. Vol. 4131. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 632–640. DOI: 10.1007/11840817_66.

[121] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization." In: *International Journal of Computer Vision* 128.2 (2019), pp. 336–359. DOI: 10.1007/s11263-019-01228-7.

[122] Thomas N. Seyfried. *Cancer as a Metabolic Disease*. John Wiley & Sons, 2012. ISBN: 9780470584927.

[123] Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 640–651. DOI: 10.1109/TPAMI.2016.2572683.

[124] Noam Shussman and Steven D Wexner. "Colorectal polyps and polyposis syndromes." In: *Gastroenterology Report* 2.1 (2014), pp. 1–15. DOI: 10.1093/gastro/got041.

[125] Juan Silva et al. "Toward embedded detection of polyps in WCE images for early diagnosis of colorectal cancer." In: *International journal of computer assisted radiology and surgery* 9 (Sept. 2013). DOI: 10.1007/s11548-013-0926-3.

[126] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *Proceedings of the 2015 International Conference on Learning Representations (ICLR)*. Sept. 2014. DOI: 10.1109/ACPR.2015.7486599.

[127] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1929–1958. ISSN: 1532-4435.

[128] M. Stone. "Cross-Validatory Choice and Assessment of Statistical Predictions." In: *Journal of the Royal Statistical Society. Series B (Methodological)* 36.2 (1974), pp. 111–147. DOI: 10.1111/j.2517-6161.1974.tb00994.x.

[129] Carole H. Sudre et al. "Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations." In: *Lecture Notes in Computer Science* (2017), pp. 240–248. DOI: 10.1007/978-3-319-67558-9_28.

[130] Hyuna Sung et al. "Global Cancer Statistics 2020: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries." In: *CA: A Cancer Journal for Clinicians* 71.3 (2021), pp. 209–249. DOI: 10.3322/caac.21660.

[131] James Surowiecki. *The Wisdom of Crowds*. Anchor, 2005. ISBN: 0385721706.

[132] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning." In: *Proceedings of the 2013 International Conference on Machine Learning (ICML)*. Vol. 28. 3. PMLR, 2013, pp. 1139–1147.

[133] Christian Szegedy et al. "Going Deeper with Convolutions." In: *CoRR* (2014). DOI: 10.1109/CVPR.2015.7298594.

[134] Mingxing Tan and Quoc V. Le. "EfficientNetV2: Smaller Models and Faster Training." In: *CoRR* (2021). arXiv: 2104.00298.

[135] Wallapak Tavanapong et al. "Artificial Intelligence for Colonoscopy: Past, Present, and Future." In: *IEEE Journal of Biomedical and Health Informatics* (2022), pp. 1–1. DOI: 10.1109/JBHI.2022.3160098.

[136] Vajira Thambawita et al. "DivergentNets: Medical Image Segmentation by Network Ensemble." In: (2021). DOI: 10.48550/ARXIV.2107.00283.

[137] Shrawan Kumar Thapa et al. "Task-Aware Active Learning for Endoscopic Image Analysis." In: (2022). DOI: 10.48550/arxiv.2204.03440.

[138] Tina Sanders Valerie C Scanlon. *Essentials of Anatomy and Physiology*. F.A. Davis Company, 2007. ISBN: 0803615469.

[139] Yi Wang et al. "Polyp-Alert: Near real-time feedback during colonoscopy." In: *Computer Methods and Programs in Biomedicine* 120.3 (2015), pp. 164–179. DOI: 10.1016/j.cmpb.2015.04.002.

[140] Sholom M. Weiss et al. "A model-based method for computer-aided medical decision-making." In: *Artificial Intelligence* 11.1 (1978). Applications to the Sciences and Medicine, pp. 145–172. DOI: 10.1016/0004-3702(78)90015-2.

[141] Sanghyun Woo et al. "CBAM: Convolutional Block Attention Module." In: (2018). DOI: 10.48550/arxiv.1807.06521.

[142] Juri Yanase and Evangelos Triantaphyllou. "A systematic survey of computer-aided diagnosis in medicine: Past and present developments." In: *Expert Systems with Applications* 138 (2019). DOI: 10.1016/j.eswa.2019.112821.

[143] Michael Yeung et al. "Focus U-Net: A novel dual attention-gated CNN for polyp segmentation during colonoscopy." In: (2021). DOI: 10.48550/ARXIV.2105.07467.

[144] Jason Yosinski et al. "Understanding Neural Networks Through Deep Visualization." In: *CoRR* (2015). arXiv: 1506.06579.

[145] Aston Zhang et al. "Dive into Deep Learning." In: *arXiv:2106.11342* (2021).

[146] Huamin Zhang and Feng Ding. "On the Kronecker Products and Their Applications." In: *Journal of Applied Mathematics* 2013 (Jan. 2013). DOI: 10.1155/2013/296185.

[147] Ruikai Zhang et al. "Automatic Detection and Classification of Colorectal Polyps by Transferring Low-Level CNN Features From Nonmedical Domain." In: *IEEE Journal of Biomedical and Health Informatics* 21.1 (2017), pp. 41–47. DOI: 10.1109/JBHI.2016.2635662.

[148] Ruikai Zhang et al. "Polyp detection during colonoscopy using a regression-based convolutional neural network with a tracker." In: *Pattern Recognition* 83 (2018), pp. 209–219. DOI: 10.1016/j.patcog.2018.05.026.

[149] Ding-Xuan Zhou. "Universality of Deep Convolutional Neural Networks." In: *Applied and Computational Harmonic Analysis* 48.2 (2018), pp. 787–794. DOI: 10.1016/j.acha.2019.06.004.

[150] A.P. Zijdenbos et al. "Morphometric analysis of white matter lesions in MR images: method and validation." In: *IEEE Transactions on Medical Imaging* 13.4 (1994), pp. 716–724. DOI: 10.1109/42.363096.

Appendix A

Source Code

The source code related to all experiments mentioned in this thesis are made publicly available on the following GitHub repository: https://github.com/feliciajacobsen/PolypSegmentation/. This repository include all preprocessing steps, model classes, training, validation, testing, and the entire deep ensembles- and MC dropout pipelines.

Appendix B

Published Papers

B.1 Paper 1 — Predictive Uncertainty Masks from Deep Ensembles in Automated Polyp Segmentation

Predictive Uncertainty Masks from Deep Ensembles in Automated Polyp Segmentation

Felicia Ly Jacobsen[1]
[1]SimulaMet, Norway
f.l.jacobsen@fys.uio.no

ABSTRACT

This paper presents the submission of team F-HOST for the Medico: Transparency in Medical Image Segmentation task held at MediaEval 2021. We propose a U-Net-based ensemble model for solving the automatic polyp segmentation task and interpret the predictions using a specific method for obtaining uncertainty. Our predicted segmentation masks show a mean Dice score of 45.01% based on the test data. The corresponding uncertainties show systematic errors towards the training data, which indicates overfitting.

I INTRODUCTION

Polyps are abnormal growths inside the lining of the colon or rectum. They can potentially develop into being malignant, leading to colorectal cancer, and thereby act as a precursor for cancer. Detecting and removing polyps with colonoscopic polypectomy during or before further development, will allow for more treatment options and overall improved prognosis [11].

Currently, the gold standard of finding and removing polyps is through a procedure called colonoscopy. This procedure is dependent upon differences in skill, experience, and technique of the endoscopists. However, studies show that up to 28% remain undetected [8]. Automated semantic segmentation based on deep learning frameworks can be used as a tool to detect polyps based on images from colonoscopy examinations. Deep Ensembles can provide an uncertainty quality of the predicted segmentation, even for ensembles with five trained models [7]. This method is known as being easy to implement and being scalable to different deep learning (DL) frameworks and can additionally improve classification error and robustness in terms of dataset shift. In this paper, the results based on the challenge test data are presented and discussed, including their corresponding uncertainty mask estimated from a Deep Ensemble model consisting of five U-Net networks.

II APPROACH

In this section, the approach to the Medico task "Transparency in Medical Image Segmentation" of the MediaEval 2021 challenge is presented. All models were trained using the PyTorch framework [9] on an Nvidia Tesla V100 32GB General-Purpose Graphics Processing Unit (GPGPU).

II.1 Datasets

There is a total of 1,362 images in the development dataset [5]. We randomly select 272 for validation and the rest for training. The test data only consist of a total of 200 images, excluding the ground truth masks. The dataset is based on the HyperKvasir dataset [2], but includes additional images and masks.

II.2 Experimental Setup

We used the U-Net architecture as the base model for the Deep Ensemble, with a total of five U-Nets. The development data was resized into 256×256 pixels before training, due to memory constraints and to reduce training time. The training data was split into batches of 32 images in order to obtain greater training efficiency as opposed to a larger batch size of, e.g., 64. Data augmentation was performed on the fly for each training iteration in order to obtain improved generalization. We use techniques such as blurring, color jitter, horizontal flip, random rotate 90°, and vertical flip. Instead of using transposed convolution in the decoder part of the network as proposed in the original U-Net paper [10], two-dimensional bilinear upsampling is used in order to avoid potential checkerboard artifacts. All models in the ensemble were trained using an initial learning rate of $1 \cdot 10^{-4}$, with a learning rate scheduler with a minimum learning rate of $1 \cdot 10^{-7}$. Each model had a total of 150 training iterations, using the Adam optimizer [6] and the Dice coefficient loss. After the last training iteration, the model weights for each model in the deep ensemble was saved in a *.pt* format. Hyperparameter tuning was done manually by observing the dice loss on the validation data as a function of training iterations, and evaluating the Dice Coefficient (DE), Jaccard Index (JI) and Accuracy.

When performing prediction with the deep ensemble, each individual model is loaded, and each predict on the input image from the test dataset. The element-wise mean is calculated from the output from each of the models in the ensemble. They are later pushed through a Sigmoid activation and thresholded into binary pixel values. The variance provided by the ensemble is used as an approximation for the uncertainty of each prediction mask. This is calculated by taking the squared sum of each probability prediction (Sigmoid output) minus the mean probability prediction from the ensemble. This squared sum is later divided by the number of models in the ensemble, five in this case.

For subtask 2: "Algorithm Efficiency", the time in seconds was calculated for the ensemble to make its overall mean prediction for each of the test images in order to measure the model efficiency of the ensemble. A Docker image is made, and using this image will make a *.csv* file with the image name and its corresponding prediction time in seconds. The Deep Ensemble will be run on the challenge organizers' hardware, and they provide us with the frames per second (FPS), which is the average number of masks from the test dataset the ensemble is able to make per second.

MediaEval'21, December 13-15 2021, Online

For subtask 3: "Transparent Machine Learning Systems", all source code is made publicly available on GitHub[1], which also includes the uncertainty images for the prediction masks.

III RESULTS AND ANALYSIS

Table 1 summarizes the results for the Medico subtask 1, including the mean DC, mean JI and mean Accuracy for the prediction masks on the validation data and the official task test data. These results show that the Deep Ensemble generalize poorly onto the test data, with a decrease of approximately 55% in the DC score and 46% decrease in the mean JI when comparing the results from the validation data on the test data. There is a high variance of DC score in the individual images from the test images, some get a DC as high as 0.8935, whereas some images get as low as 0.0000. Higher performance can be increased by performing more hyperparameter tuning, training the Deep Ensemble on more training examples including similar datasets such as for example the CVC-ClinicDB dataset [4] and the CVC-ColonDB dataset [1]. Additionally, decreasing the number of training iterations can also contribute to a more generalized ensemble model. Also, as proposed in the original paper [7], adding adversarial training and increasing the number of models in the ensemble from 5 to 15, may potentially decrease the prediction error significantly.

Table 1: Results from validation data and test data by the ensemble model of five U-Nets. The results on predicted test data were provided by the task organizers.

	Mean Dice	Mean Jaccard Index	Mean Accuracy
Validation data	0.8226	0.7005	0.9242
Official Test data	0.4501	0.3231	0.8831

For the efficiency subtask, a FPS of 82.9496 was obtained. This means that the time of approximately 2.4111 seconds in total was used to generate the masks on the entire test dataset. This result indicates satisfactory model efficiency, but in return the deep ensemble is both memory- and time consuming to train.

A set of three randomly chosen images from the test data and their corresponding prediction masks and uncertainty heatmaps are shown in Figure 1. The brighter areas in the heatmaps illustrate the pixels where the models in the ensemble disagree the most. These results show that the borders of the detected polyps are where they disagree the most. Furthermore, the two uncertainty heatmaps (from the left) shows an outlining of a rectangle in the bottom left corner. Many of the input images in the HyperKvasir dataset show green rectangles located in the same area, this is information important to the medical experts. Thus, it is common to observe several images with green rectangles in the development dataset. However, note that the input images do not contain these green rectangles. These results indicate that the ensemble expected these rectangles, thus showing systematic bias towards the training data. Increasing the number of training examples, as well as performing corrections to training images where these rectangles appear by, e.g., cropping them out may boost model performance.

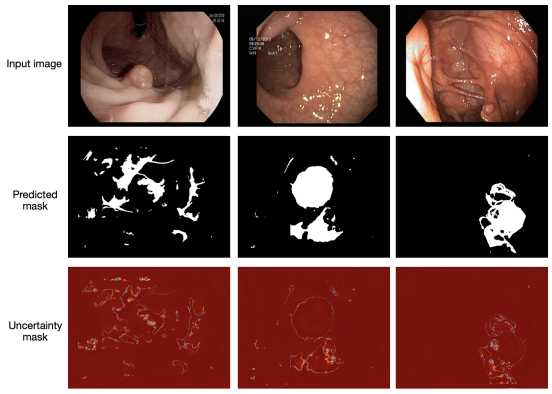

Figure 1: Examples of the input images from the official test dataset are shown on the top row. Their corresponding predicted masks are shown on the middle row, and their uncertainty heatmap representation are on the bottom row. The prediction masks and uncertainty heatmap are calculated using the Deep Ensemble of five trained U-Net networks.

IV CONCLUSION AND FUTURE WORK

In this paper, we presented a method of obtaining the approximate uncertainty values for a set of predicted segmentation masks. The uncertainty masks provide an uncertainty measure of the performance of a U-Net based DL model trained on medical colonoscopy images of polyps.

A mean Dice score of 0.4501 was obtained on the test data, and compared to the Dice score of 0.8226 from the validation data, this indicated that the Deep Ensemble model was being overfitted to the training data, and thus generalizing poorly onto unseen data. Increasing the number of training examples by including similar datasets, decreasing the number of training iterations, increasing the number of models in the ensemble, as well as including adversarial training may improve generalization. A total average FPS of 82.9496 was obtained on the test data, but came at a high computational cost when training the Deep Ensemble. In future work, we will add the aforementioned proposed extensions, as well as experiment and compare to alternative methods such as Masksembles [3] in order to decrease computational cost of obtaining an ensemble model.

REFERENCES

[1] Sánchez J. Vilarino-F Bernal, J. 2012. Towards automatic polyp detection with a polyp appearance model. *Endoscopy* (2012), 3166–3182.

[2] Hanna Borgli, Vajira Thambawita, Pia H Smedsrud, Steven Hicks, Debesh Jha, Sigrun L Eskeland, Kristin Ranheim Randel, Konstantin Pogorelov, Mathias Lux, Duc Tien Dang Nguyen, Dag Johansen, Carsten Griwodz, Håkon K Stensland, Enrique Garcia-Ceja, Peter T Schmidt, Hugo L Hammer, Michael A Riegler, Pål Halvorsen, and Thomas de Lange. 2020. HyperKvasir, a comprehensive multi-class image and video dataset for gastrointestinal endoscopy. *Scientific Data* 7, 1 (2020), 283. https://doi.org/10.1038/s41597-020-00622-y

[1] https://github.com/feliciajacobsen/MediaEval2021

[3] Nikita Durasov, Timur M. Bagautdinov, Pierre Baqué, and Pascal Fua. 2020. Masksembles for Uncertainty Estimation. *CoRR* abs/2012.08334 (2020). https://arxiv.org/abs/2012.08334

[4] Bernal J. López-Cerón M. Córdova H. Sánchez-Montes C. Rodríguez de Miguel C. Sánchez F. J. Fernández-Esparrach, G. 2016. Exploring the clinical potential of an automatic colonic polyp detection method based on the creation of energy maps. *Endoscopy* (6 2016), 837–842.

[5] Steven Hicks, Debesh Jha, Vajira Thambawita, Hugo Hammer, Thomas de Lange, Sravanthi Parasa, Michael Riegler, and Pål Halvorsen. 2021. Medico Multimedia Task at MediaEval 2021: Transparency in Medical Image Segmentation. In *Proceedings of MediaEval 2021 CEUR Workshop*.

[6] Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (12 2014).

[7] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. (12 2017), 6405–6416 pages. arXiv:stat.ML/1612.01474

[8] Kim NH, Jung YS, Jeong WS, and et al. 2017. Miss rate of colorectal neoplastic polyps and risk factors for missed polyps in consecutive colonoscopies. *Intestinal research* 15, 3 (6 2017), 411–418. https://doi.org/10.5217/ir.2017.15.3.411.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, 234–241.

[11] Winawer SJ, Zauber AG, Ho MN, and et al. 1993. Prevention of colorectal cancer by colonoscopic polypectomy. *The New England Journal of Medicine* 329, 27 (12 1993), 1977–1981. https://doi.org/10.1056/NEJM199312303292701

B.2 Paper 2 — Estimating Predictive Uncertainty in Gastrointestinal Polyp Segmentation

Estimating Predictive Uncertainty in Gastrointestinal Polyp Segmentation

Felicia Ly Jacobsen[1,2], Steven A. Hicks[2], Pål Halvorsen[2,3], and Michael A. Riegler[2,4]

[1]*Department of Physics, University of Oslo, Norway*
[2]*Simula Metropolitan Center for Digital Engineering, Norway*
[3]*Department of Computer Science, Oslo Metropolitan University, Norway*
[4]*UiT - Arctic University of Norway, Norway*

***Abstract*—Deep neural networks have achieved state-of-the-art performance on numerous applications in the medical field, with use-cases ranging from automation of mundane tasks to diagnosis of life threatening diseases. Despite these achievements, deep neural networks are considered "black boxes" due to their complex structure and general lack of transparency in their decision-making process. These attributes make it challenging to incorporate deep learning into existing clinical workflows as decisions often need more support than blind faith in a statistical model. This paper presents an investigation of uncertainty estimation for the detection of colon polyps using deep convolutional neural networks (CNNs). We experiment with two different approaches to measure uncertainty, Monte Carlo (MC) dropout and deep ensembles, and discuss the advantages and disadvantages of both methods in terms of computational efficiency and performance gain. Furthermore, we apply the two uncertainty methods to two different state-of-the-art CNN-based polyp segmentation architectures. The uncertainty is visualized as heatmaps on the input images and can be used to make more informed decisions on whether or not to trust a model's predictions. The results show that the predictive uncertainties provide a comparison between different models' predictions which can be interpreted as contrastive explanations where the values are largely influenced by the degree of independence between the models in the ensemble. We also reveal that MC dropout is shown to lack at providing contrastive uncertainty values due to the high correlation between the models' in the ensemble.**

***Index Terms*—computer-aided diagnosis, deep learning, machine learning, segmentation, uncertainty, and explainability.**

I. Introduction

Colorectal cancer (CRC) is currently the third-highest prevalent cancer worldwide and is the second most common cancer-related death [1]. CRC typically starts with a small benign growth inside the lining of the colon or rectum, called a polyp, which can become malignant over time. A polyp often shows few signs of or no symptoms at all [2]. The necessity for high-standard polyp detection techniques is crucial in order to improve the prognosis of CRC in patients. Today, colonoscopy is the gold standard for such preventive screening programs [3]. However, studies show that up to approximately 28% of polyps are undetected [4], [5].

Computer-aided detection (CAD) systems [6] are designed to decrease observational oversights and have already been successfully integrated into diagnostic mammography [7]. Deep learning (DL)-based segmentation can be used to highlight polyps in images and videos. In these cases, DL can become the main component in CAD systems which can be used by medical experts during colonoscopy screenings to reduce the number of undetected polyps. CAD systems can also reduce workloads for medical personnel, increase efficiency at clinics, verify and assess the quality of earlier examinations, and train inexperienced physicians and medical students [6], [8]. Moreover, CAD systems is considered cost-efficient because the use of a computer rather than a second human observer has the advantage of reducing the demand for trained physicians [6].

Despite the potential advantages of DL-based CAD, the lack of explainability and transparency are important limitations in considering their potential future in clinical practice [9]. DL models are considered to have a "black box"-nature because they lack the ability to explain the underlying mechanisms of their decision-making process [10]. Similar to medical experts, CAD systems should also be held accountable in order to learn from their mistakes and to continuously improve.

Explainability and transparency can provide accountability to DL-based CAD systems which can help to secure the acceptance of the integration of DL models in medical clinics [10]. One way of making a model more transparent is by providing additional information about the model's uncertainty regarding a performed prediction, which can aid medical experts in making more informed decisions when using the CAD system. Pixel-level uncertainty values can accentuate spatial areas of contrast between different values of confidence in predictions given by an ensemble of deep neural networks. In these types of applications, the uncertainty values can be interpreted as contrastive explanations that provide a comparison between a prediction given by one model to another. These *explanations* can aid medical experts in evaluating the performance of different DL models.

In this paper, we provide uncertainty representations based on predicted segmentation masks generated by a convolutional neural network (CNN). We experiment with two different uncertainty estimation approaches using a polyp segmentation use-case, one based on deep ensembles and the other using monte carlo (MC) dropout. The two approaches are easily integrated into existing CNN-based prediction frameworks and

can aid in providing more information about the prediction of a given sample. Thus, the main contributions of this paper are as follows:

1) We estimate the uncertainty of current state-of-the-art polyp segmentation models and evaluate their predictive performance using a publicly available dataset.
2) We compare the performance of different segmentation models across ensembles.
3) We evaluate the uncertainty representations between two different uncertainty estimation approaches.

II. Background and Related Work

Although DL has shown to be very successful across several medical domains, there are still cases where it may make poor or unintelligible predictions. Trust in the underlying DL models is paramount if DL-based systems are to be properly integrated into hospitals and clinics. Providing a binary prediction output based on an arbitrary probability threshold is insufficient and does not deliver the information needed for medical professionals to interpret whether the provided prediction can be trusted or not. Uncertainty estimation has recently gained much attention in machine learning, especially for DL-based models [11], [12].

Several approaches to uncertainty estimating of deep neural networks have been explored, including methods using Bayesian neural networks [13], ensemble models [14], and test-time data augmentation [15], [16]. Ghahramani et al. [17] proposed a method called MC dropout, which utilizes dropout during test time to approximate Bayesian inference in deep Gaussian processes. The approach has gained major success due to its simplicity and not requiring significant changes to existing neural network architectures. Another popular method of estimating uncertainty is by using a method called deep ensembles [18], [19]. Traditional ensemble models use a collection of models that each make a prediction to a specific input and come to a consensus using a predetermined set of rules [20]. We can use these independent predictions to measure the agreement between the different models, where a high level of agreement signifies low uncertainty and a low level of agreement indicates a high level of uncertainty. Lastly, test-time data augmentation applies simple transformations to the input data in order to measure the change in prediction when processed by the model and mimic the effect of larger datasets, which can provide more generalizable prediction models [21]. Our approach to uncertainty estimation is motivated by the demand for explainability in DL models.

III. Experiments

Using colon polyp segmentation as a use-case, we trained two deep CNN-based models to automatically segment images collected from real-world colonoscopies. We use a publicly available dataset called Kvasir-SEG [22] which consists of $1,000$ colonoscopy images containing polyps and their corresponding annotations. Examples from this dataset are shown in parts of Figure 2. We randomly split 80% of the total dataset into the training set, 10% into a validation set, and 10% into the unseen test set. The training set is used to calculate the standard deviation and mean across the RBG color channels in order for each dataset to be normalized based on the training set statistics. The result of normalization is color channel intensities bounded by the range $[0, 1]$. Due to RAM restrictions, we resize all input images to 256×256, use a batch size set to 32 and utilize mixed-precision training [23]. We set the total training epochs to 150, which based on our internal testing would ensure convergence during training. We monitored convergence by tracking the training- and validation loss values for each iteration.

As mentioned earlier, Kvasir-SEG only consists of $1,000$ image and mask pairs. We thus extended the dataset using augmentations to increase generalizability. Augmentation is only applied to the training data and is performed on-the-fly for each iteration. Each transformation is assigned a probability, which represents the probability of it being applied to the input image. Thus, multiple transformations can be simultaneously applied to the same image. We use a random rotation transformation method with a limit of 35 degrees counterclockwise with a corresponding probability of 10%, a horizontal flip transformation that flips the image around horizontally with a probability of 50%, and a vertical flip of 10%. We limited the number of possible transformations to three as we avoided the possibility of distorting the images too much such that they completely destroys the features characterizing the polyps. The augmentations were implemented using the Albumentations library [24].

For uncertainty estimation, we adapt the MC dropout [25] and the deep ensemble [18] approach to extract the variance across an ensemble of CNNs trained for automatic polyp segmentation. The variance is visualized as heatmap representations representing spatial predictive uncertainty in the predicted polyp mask, for which a similar approach can be found in [19]. Furthermore, we trained two different state-of-the-art CNN architectures as baseline models for the MC dropout and deep ensemble approach. We compare these models based on their performances and uncertainty representations in the MC dropout- and deep ensemble pipeline.

All experiments were implemented in Pytorch [26] and run on the *eX3 computing cluster*.[1] Among other things, this computing cluster contains NVIDIA Tesla V100 GPUs with 32 GB of RAM which were used for model training. The code used for all experiments is publicly available[2].

IV. Ensemble Models

The primary objective behind deep ensembles is to extract the uncertainty from the neural network using a combination of slightly different baseline models. Intuitively, properly trained models initialized with different weights should converge to a similar endpoint but may behave differently when applied to difficult parts of the image. By comparing the differences in the predictions made by these models, we are able to extract

[1] https://www.ex3.simula.no/

[2] https://github.com/feliciajacobsen/PolypSegmentation

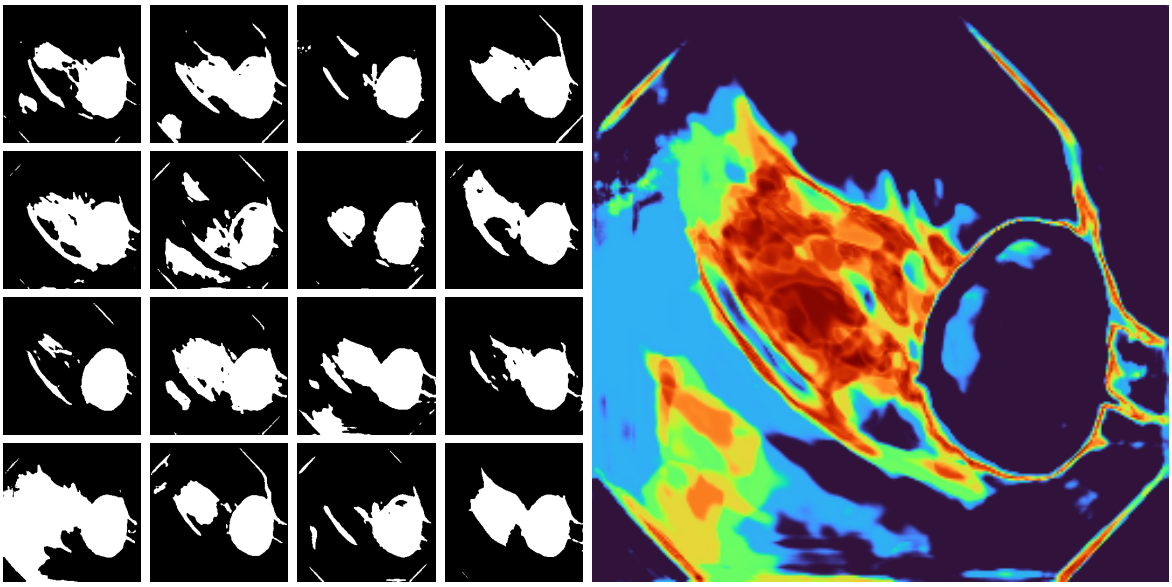

Fig. 1: The left images present the predicted polyp segmentation masks for one image sample generated by each of the 16 different baseline models from the ResUNet++ based deep ensemble. The right image is the corresponding uncertainty map generated using the predicted masks.

information about what regions of an image the ensemble is most uncertain about. Measuring uncertainty using the deep ensemble approach can be broken down into three main steps:

1) Separately train n baseline models.
2) Run the input, y_i, through each respective baseline model to generate n predictions by $\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$, where $\hat{y}$ is the final output of the ensemble.
3) Use the difference across the predictions to measure the uncertainty of the final prediction.

We apply this deep ensemble approach using two sets of baseline model architectures, one based on the popular medical image segmentation architecture U-Net [27], and the other based on ResUNet++ [8], which is an architecture that has previously shown good performance on the Kvasir-SEG dataset. The reason for experimenting with two different baseline model architectures for uncertainty estimation was to identify possible differences in uncertainty representations depending on the baseline model architecture.

The deep ensembles were built using 16 baseline models, each trained on the same training and validation set. Each baseline model is based on the same CNN architecture. The decision to use 16 models was based on manual testing. Starting at 5 baseline models, we gradually increased the number of models while monitoring the gained performance and stopped once the ensemble size did not fit within our memory constraints. For weight and bias initialization of each convolutional layer in the CNNs, we use the default parameter initialization provided in PyTorch, which is the Gaussian He initialization scheme. We use He initialization with the intention of initializing the layers such that the variance of activations across all layers are the same during each forward pass.

The U-Net-based ensemble were implemented using the exact architecture described in the original U-Net paper [27], but replaced all transposed convolutions with bilinear interpolation layers. We did so in order to avoid checkerboard artifacts, which are seen in models containing transposed convolutions. For the ResUNet++ based ensemble, the baseline model were implemented in the same way as described in the original paper [8].

The hyperparameters were manually tuned and held constant between all 16 baseline models, where we used the hyperparameters from the original ResUNet++ paper [8] as a starting point due to the same use-case. For training, weights were optimized using the Adam optimizer with $\beta_1^t = 0.9$ and $\beta_2^t = 0.999$, and dice similarity coefficient (DSC) loss was used to calculate the error. The initial learning rate for all training schemes was set to $1 \cdot 10^{-4}$. This learning rate value corresponded to the fastest convergence during training. We used the trained ensemble to make a collective prediction based on the same set of test data. For one test example, the predicted probabilities across each model are averaged, and variance across the ensemble is computed, representing the estimated predictive uncertainty. This uncertainty is then visualized as a heatmap, which shows the regions of agreement and disagreement between the models in the ensemble (red colors mean higher uncertainty and blue to dark blue means weak to no uncertainty, respectively). Figure 1 shows the predicted masks for each of the 16 baseline models and their corresponding uncertainty representations.

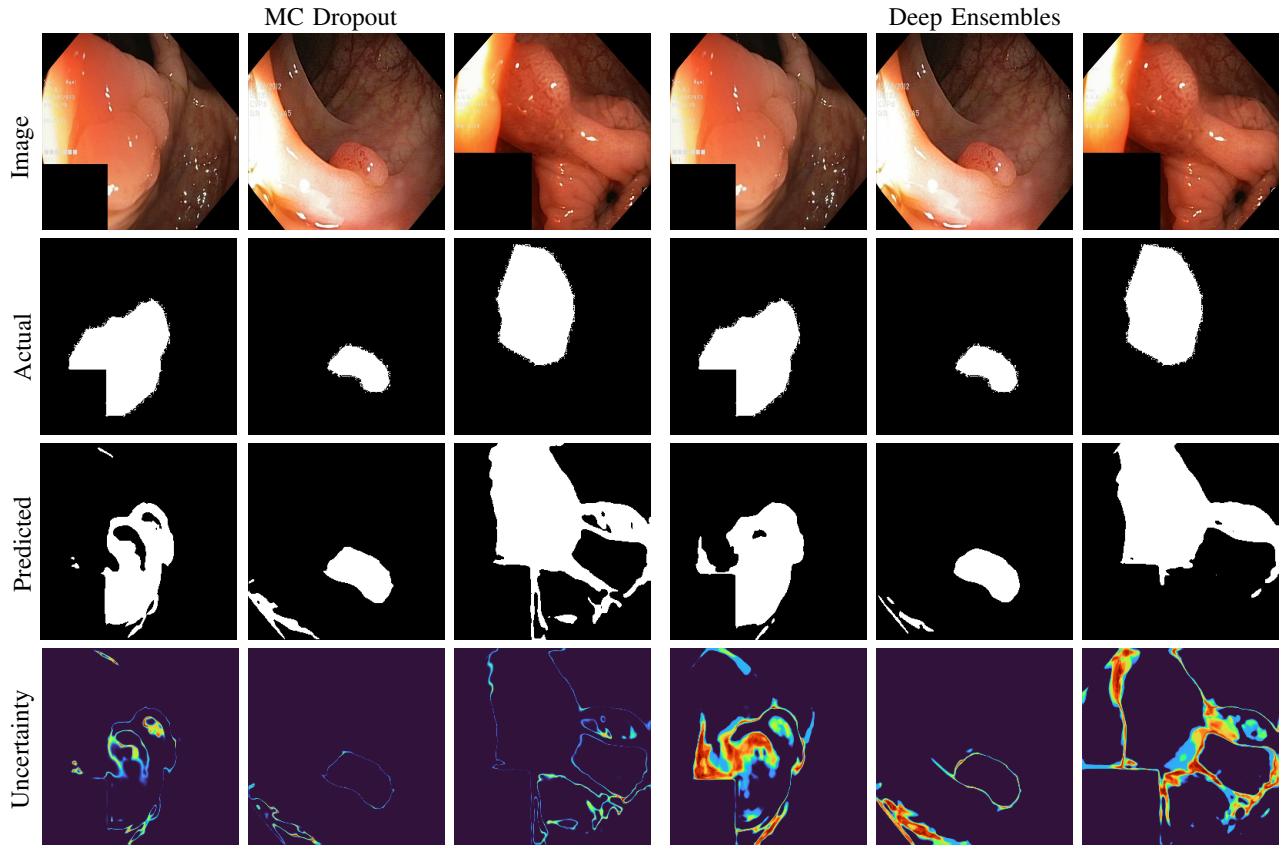

Fig. 2: Three original input images from Kvasir-SEG, the ground truth segmentation masks, the predicted masks, and their corresponding predictive uncertainties. These images were generated using MC dropout (left) and deep ensembles (right) with U-Net as the baseline model.

V. Monte Carlo Dropout

The primary goal of MC dropout is to generate predictions that can be interpreted as samples from the probabilistic distribution, more specifically, a probabilistic model called deep Gaussian Process. A neural network where dropout is applied during test time allows the model to act as an ensemble of classifiers and not just one single model. The MC dropout method can be broken down into three distinct steps:

1) Train a single neural network that contains dropout.
2) Run the input through the model n times with enables dropout to generate n predictions, where $\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the final output of the ensemble.
3) Use the difference in the predictions to measure the uncertainty of the final prediction.

Like the deep ensemble method, U-Net and ResUNet++ are used as baseline models for MC dropout. We adapt the MC dropout approach introduced by DeVries et al. [28] by implementing one dropout layer to each of the five intermediate blocks in the U-Net architecture. Hence, a total of 5 dropout layers are added to the architecture, which represents the dropout-modified U-Net. We do the same approach for ResUNet++, where 4 dropout layers are added to each intermediate block. The dropout rate is held constant, such that we have a global dropout rate for each of the dropout-modified models. The dropout rate is tuned by tracking the training- and validation loss using different dropout rates ranging from 0–0.5. We found 0.1 to give the highest validation score for all models trained with different loss functions.

To mimic the effect of model ensembling, the dropout layers are enabled during test time. The dropout layers provide a source of randomness such that each model during test time be viewed as different to the model in the next iteration. During test time, the model can provide different predictions at each forward pass given an input. The number of forward passes can be interpreted as the number of models' in the ensemble. The variance across the different predictions in the ensemble will represent the predictive uncertainty.

VI. Prediction Stacking

Each trained model in the ensemble is saved and can be loaded into a defined model architecture in order to make a collective prediction. For the case of MC dropout, the prediction for each forward pass corresponding to a specific input test image is stacked along a dimension with the length correspondingly to the ensemble size. Likewise, each of the

Model	Uncertainty	DSC
U-Net	MC dropout	0.7538
	Deep ensemble	0.8172
ResUNet++	MC dropout	0.7190
	Deep ensemble	0.7370

TABLE I: The summarized results of MC dropout and deep ensembles with an ensemble size of 16 in each case. U-Net and ResUNet++ were used as baseline models. The main results are the mean DSC based on Kvasir-SEG test data.

trained models in the deep ensembles is loaded into its corresponding model architecture for each model and stacked along the dimension corresponding to the ensemble size. Loading in a different model architecture for each model in the deep ensemble will require more computational capacity than the process of facilitating MC dropout during test time. It is important to note that the predictions corresponding to the exact same input image must be stacked together. The stacked predictions are averaged across each model in the ensemble by taking the element-wise mean along the ensemble size dimension. The result represents the mean prediction by the ensemble.

The mean prediction corresponding to predicted probabilities is transformed using a threshold equal to 0.5 in order to represent the class values. The positive class corresponds to a pixel that belongs to a polyp, and the negative class corresponds to the background class. Thresholding assigns low probabilities to the background class and high probabilities to the polyp class.

The variance is computed from the stacked predicted probabilities along the ensemble size dimension. Each element in the variance tensor is normalized to lie in the range of $[0, 1]$. We proceed by mapping the variance tensor to a colormap, such that it can be interpreted as spatial predictive uncertainties represented by a heatmap. The Turbo colormap is preferred, which clearly accentuates the areas of high uncertainty as compared to other colormaps in Matplotlib. The red color corresponds to a high uncertainty, the yellow to cyan colors represent the average values of uncertainties, and the dark blue color represents low uncertainty. We observe that this colormap provides an easy visual assessment as it provides high-contrast and smoothly varying color transitions to the images.

VII. Results and Discussion

Looking at Figure 2, we can observe that the predicted masks are more similar to the actual masks using the deep ensemble compared to the ensemble generated by MC dropout. This is also reflected in Table I which displays the different values of mean DSC based on the Kvasir-SEG test dataset, where an ensemble size of 16 gave a mean DSC equal to 0.8172, whereas 16 forward passes generated by MC dropout gave a mean DSC of 0.7538.

In contrast to deep ensembles, we observe from Figure 3 that the MC dropout gave no performance increase in terms of the test DSC with the increasing ensemble size. However,

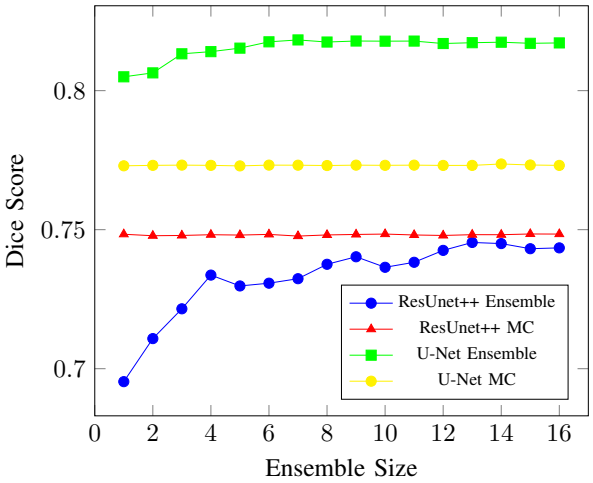

Fig. 3: A graph showing how the performance changed as the ensemble size increased for the different ensembles each of different CNN architectures used as baseline models. Note that the y-axis does not start on 0 to better highlight the differences.

we can see that regardless of the choice of CNN architecture used in the deep ensemble, increasing ensemble size up to a value of 7 significantly increases the performance. For a deep ensemble of ResUNet++ models, we observe an increasing performance to an ensemble size of 14. In contrast, for a deep ensemble of U-Nets, we observe no increasing performance after an ensemble size of 7. But overall, a deep ensemble of 7 U-Nets gave the best DSC score of 0.8182. Because only one model is trained for the MC dropout ensemble, the different models can be interpreted as highly correlated. For deep ensembles, however, each model in the ensemble is initialized and trained separately. This level of independence and diversity is also reflected in the uncertainty values as observed in figures 1, where the quality of the prediction of the independently trained model largely varies depending on the specific model in the deep ensemble. This diversity of models across the an ensemble indicates a favorable property given the performance gain. For the results in Figure 2, the uncertainty representations given by the MC dropout approach are elusive and obscure. Hence, MC dropout lack at providing contrastive explanations for this specific use-case.

Regarding the quality of the uncertainty representations and usability for clinical practice, we conclude that the deep ensembles are more suitable as they provide more precise information about the comparison of the performance across the ensemble. Furthermore, we see that the uncertainties created using the deep ensemble method offer more information that can help interpret the results and make them more trustworthy. By comparing the masks generated by the ensemble model and corresponding uncertainty masks in Figure 2, we see a clear indication of which parts of the segmentation masks are most probably incorrect.

VIII. Conclusion

This paper presented our approach of using deep neural networks to segment polyps in images collected from colonoscopies and measure the predictions' uncertainty using state-of-the-art uncertainty estimation methods. The presented results show that the use of deep ensembles not only provides clear and distinct uncertainty estimates to the predictions, but also improves the performance in terms of the mean DSC during test time. In contrast to deep ensembles, the MC dropout approach has the benefits of high computational efficiency and low memory requirements. Despite this, the resulting uncertainty estimates are elusive and obscure. Based on this comparison, we argue that the uncertainty representations based on deep ensembles can aid medical experts in evaluating and characterizing the model performance compared to other models in the ensemble. Integrating DL-based CAD systems, such as polyp segmentation frameworks, together with predictive uncertainty estimates, can help to build confidence and trust between the medical experts and these systems. We hope that this work motivates the inclusion of uncertainty estimation in medical applications using DL as it could significantly impact whether a model should be trusted or not. For future work, we plan to verify the usefulness of these masks through a user study involving experienced colonoscopists and clinicians.

IX. Acknowledgments

The research has benefited from the Experimental Infrastructure for Exploration of Exascale Computing (eX3), which is financially supported by the Research Council of Norway under contract 270053.

References

[1] H. Sung, J. Ferlay, R. L. Siegel, M. Laversanne, I. Soerjomataram, A. Jemal, and F. Bray, "Global cancer statistics 2020: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries," *CA: A Cancer Journal for Clinicians*, vol. 71, no. 3, pp. 209–249, 2021.

[2] X. Long, X. Li, L. Ma, J. Lu, S. Liao, and R. Gui, "Clinical and endoscopic-pathological characteristics of colorectal polyps: an analysis of 1,234 cases," *Int. J. Clin. Exp. Med.*, vol. 8, no. 10, pp. 19 367–19 373, 2015.

[3] N. Shussman and S. D. Wexner, "Colorectal polyps and polyposis syndromes," *Gastroenterology Report*, vol. 2, no. 1, pp. 1–15, 2014.

[4] N. H. Kim, Y. S. Jung, W. S. Jeong, H.-J. Yang, S.-K. Park, K. Choi, and D. I. Park, "Miss rate of colorectal neoplastic polyps and risk factors for missed polyps in consecutive colonoscopies," *Intestinal Research*, vol. 15, no. 3, pp. 411–418, jul 2017.

[5] G.-C. Lou, J.-M. Yang, Q.-S. Xu, W. Huang, and S.-G. Shi, "A retrospective study on endoscopic missing diagnosis of colorectal polyp and its related factors." *The Turkish journal of gastroenterology*, pp. 182—-186, 2014.

[6] R. Castellino, "Computer aided detection (cad): an overview," *Cancer imaging : the official publication of the International Cancer Imaging Society*, vol. 5, no. 1, pp. 17–9, 2005.

[7] V. Rao, D. C. Levin, L. Parker, B. Cavanaugh, and A. J. Frangos, "How widely is computer-aided detection used in screening and diagnostic mammography?" *Journal of the American College of Radiology*, vol. 7, no. 10, pp. 802–805, 2010.

[8] D. Jha, P. H. Smedsrud, M. A. Riegler, D. Johansen, T. D. Lange, P. Halvorsen, and H. D. Johansen, "Resunet++: An advanced architecture for medical image segmentation," in *2019 IEEE International Symposium on Multimedia (ISM)*, 2019, pp. 225–2255.

[9] W. Samek and K.-R. Müller, "Towards explainable artificial intelligence," in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer International Publishing, 2019, pp. 5–22.

[10] C. Molnar, *Interpretable Machine Learning*, 2nd ed., 2022. [Online]. Available: https://christophm.github.io/interpretable-ml-book

[11] J. Mena, O. Pujol, and J. Vitrià, "A survey on uncertainty estimation in deep learning classification systems from a bayesian perspective," *ACM Comput. Surv.*, vol. 54, no. 9, oct 2021.

[12] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya, V. Makarenkov, and S. Nahavandi, "A review of uncertainty quantification in deep learning: Techniques, applications and challenges," *Information Fusion*, vol. 76, pp. 243–297, 2021.

[13] H. Wang and D.-Y. Yeung, "A survey on bayesian deep learning," *ACM Comput. Surv.*, vol. 53, no. 5, sep 2020.

[14] Y. Yang, H. Lv, and N. Chen, "A survey on ensemble learning under the era of deep learning," 2021.

[15] G. Wang, W. Li, M. Aertsen, J. Deprest, S. Ourselin, and T. Vercauteren, "Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks," *Neurocomputing*, vol. 338, pp. 34–45, 2019.

[16] N. Moshkov, B. Mathe, A. Kertesz-Farkas, R. Hollandi, and P. Horvath, "Test-time augmentation for deep learning-based cell segmentation on microscopy images," *Scientific reports*, vol. 10, no. 1, pp. 1–7, 2020.

[17] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 1050–1059.

[18] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Proceedings of the 2017 International Conference on Neural Information Processing Systems (NIPS)*, 2017, pp. 6405–6416.

[19] A. Mehrtash, W. M. Wells, C. M. Tempany, P. Abolmaesumi, and T. Kapur, "Confidence calibration and predictive uncertainty estimation for deep medical image segmentation," *IEEE Transactions on Medical Imaging*, vol. 39, no. 12, pp. 3868–3878, 2020.

[20] M. A. Ganaie, M. Hu, M. Tanveer, and P. N. Suganthan, "Ensemble deep learning: A review," *arXiv e-prints*, 2021.

[21] M. S. Ayhan and P. Berens, "Test-time data augmentation for estimation of heteroscedastic aleatoric uncertainty in deep neural networks," in *International conference on Medical Imaging with Deep Learning*, 2018.

[22] K. Pogorelov, K. Randel, C. Griwodz, T. de Lange, S. Eskeland, D. Johansen, C. Spampinato, D. T. Dang Nguyen, M. Lux, P. Schmidt, M. Riegler, and P. Halvorsen, "KVASIR: A multi-class image dataset for computer aided gastrointestinal disease detection," in *Proceedings of the 2017 ACM on Multimedia Systems Conference (MMSys)*, 2017, pp. 164–169.

[23] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2017.

[24] A. Buslaev, A. Parinov, E. Khvedchenya, V. I. Iglovikov, and A. A. Kalinin, "Albumentations: fast and flexible image augmentations," *arXiv e-prints*, sep 2018.

[25] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 2016 International Conference on on Machine Learning (ICML)*, vol. 48, 2016, pp. 1050–1059.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[27] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. 9351, pp. 234–241, 2015.

[28] T. DeVries and G. W. Taylor, "Leveraging uncertainty estimates for predicting segmentation quality," *arXiv e-prints*, 2018.