UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

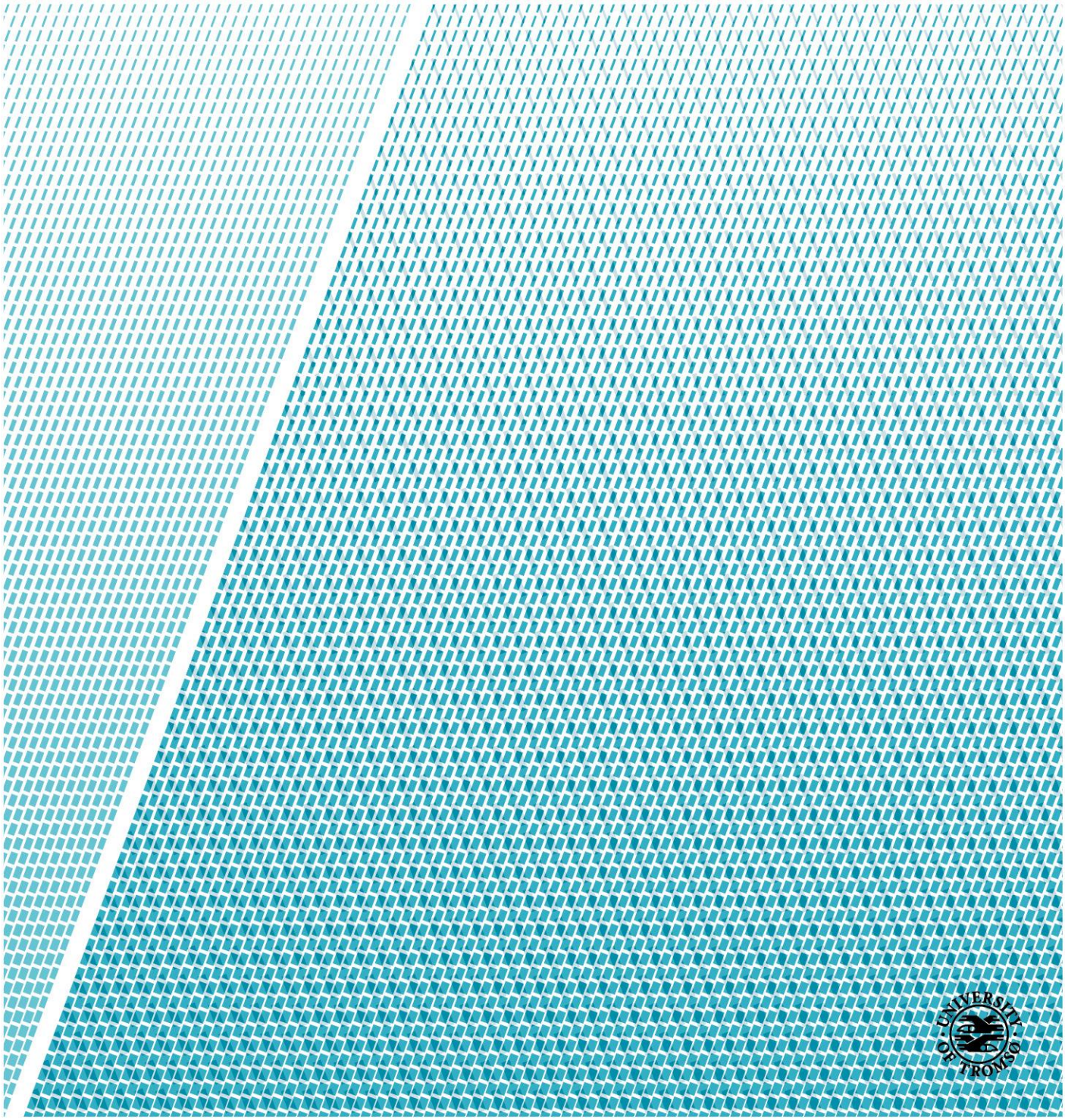
Department of Physics and Technology

Uncertainty Modeling and Interpretability in Convolutional Neural Networks for Polyp Segmentation

—

Kristoffer Wickstrøm

FYS-3900 - Master's thesis in physics 60 SP - Mai 2018



Abstract

Colorectal cancer is one of the leading causes of cancer-related deaths worldwide, with prevention commonly done through regular colonoscopy screenings. During a colonoscopy, physicians manually inspect the colon of a patient using a camera in search for polyps, which are known to be possible precursors to colorectal cancer. Seeing that a colonoscopy is a manual procedure, it can be susceptible to human factors such as fatigue which can lead to missed polyps. As a method to increase polyp detection rate, automated detection procedures which are not affected by such flaws have been proposed to aid practitioners.

Deep Neural Networks (DNNs) are propelling advances in a range of different computer vision tasks such as object detection and object segmentation. These advances have motivated research in applications of such models for medical image analysis. If DNN-based models are to be helpful in a medical context, they need to be precise, interpretable, and uncertainty in predictions must be well understood. In this thesis, we introduce a novel approach for visualizing uncertainty in DNNs and evaluate recent advances in uncertainty estimation and model interpretability in the context of semantic segmentation of polyps from colonoscopy images. We evaluate and enhance several architectures of Fully Convolutional Networks (FCNs) and provide comparison between these models. Our highest performing model achieves a considerable improvement over the previous state-of-the-art on the EndoScene dataset, a publicly available dataset for semantic segmentation of colorectal polyps. Additionally, we propose a novel approach for analyzing FCNs through the lens of information theoretic learning.

Acknowledgements

I would first like to thank my supervisor, Associate Professor Robert Jenssen, for his counsel and guidance throughout my time working on this thesis.

I would also like to thank my co-supervisor Michael Kampffmeyer, for his aid and assistance in all aspects of this thesis.

Furthermore, I would like to thank the entire UiT Machine Learning Group for their academic and social contributions during the last couple of years.

To my friends, my family, and to Sigrid, thank you for your love and support.

Kristoffer Wickstrøm

Contents

List of Figures

List of Tables

Abbreviations

CNN	**C**onvolutional **N**eural **N**etworks
CRC	**C**olo**R**ectal **C**ancer
CRF	**C**onditional **R**andom **F**ields
CVC	**C**omputer **V**ison **C**enter
DL	**D**eep **L**earning
DNN	**D**eep **N**eural **Network**
DPI	**D**ata **P**rocessing **I**nequality
DSS	**D**ecision **S**upport **S**ystem
FCN	**F**ully **C**onvolutional **N**etworks
GPU	**G**raphics **P**rocessing **U**nit
HOG	**H**istogram of **O**riented **G**radients
ITL	**I**nformation **T**heoretic **L**earning
KKT	**K**arush-**K**uhn-**T**ucker
MFCN	**M**odified **F**ully **C**onvolutional **N**etworks
MLP	**M**ulti**L**ayer **P**erceptron
MSE	**M**ean **S**quared **E**rror
RBF	**R**adial **B**asis **F**unction
RF	**R**andom **F**orest
RNN	**R**ecurrent **N**eural **N**etwork
SIFT	**S**cale **I**nvariant **F**eature **T**ransfor
SGD	**S**tochastic **G**radient **D**escent
SMO	**S**equential **M**inimal **Optimization**
SVM	**S**upport **V**ector **M**achine
WCE	**W**ireless **C**apsule **E**ndoscopy

Chapter 1

Introduction

1.1 Colorectal Cancer and Screening Procedures

Colorectal Cancer (CRC) is one of the leading causes of cancer-related deaths worldwide [1–3]. The five-year survival rate for a distant stage CRC diagnosis is estimated to be 14%, whereas the estimated survival rate for early diagnosis is 90% [4]. A common approach to increase the chance of early diagnosis is a process known as screening, where physicians perform tests to detect indications of cancer. Currently, one of the most common screening procedures for CRC is a colonoscopy, where physicians probe for non-cancerous growths referred to as colorectal polyps (shown in Figure 1.1), a possible precursor to CRC. Furthermore, Colorectal cancer has also been estimated to be one of the most expensive diseases to treat [3].

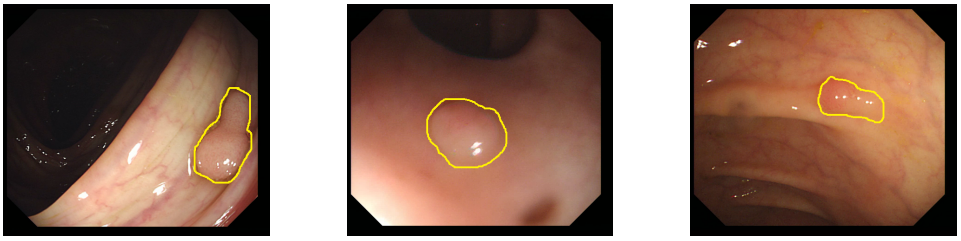


FIGURE 1.1: Images containing polyps, each marked in yellow. Images are obtained from the CVC-Colon Database [5]

A colonoscopy is performed by inserting a thin flexible tube (colonoscope) with a small camera (endoscope) through the anus to visually inspect the colon (see Figure 1.3) and usually takes between 20 minutes and 1 hour. Wireless Capsule Endoscopy (WCE) is an alternative screening method where the patient swallows a small, wireless camera

that transmits images of the intestines to a recorder worn around the waist (an example of a capsule is shown in Figure 1.2). After the procedure, the images are downloaded and assessed by a physician.

The Norwegian Directorate of Health estimates that a screening program would reduce mortality by 27% and occurrences of colorectal cancer by 22% in Norway, prompting a recommendation for a national screening program for persons older than 55 years [6].

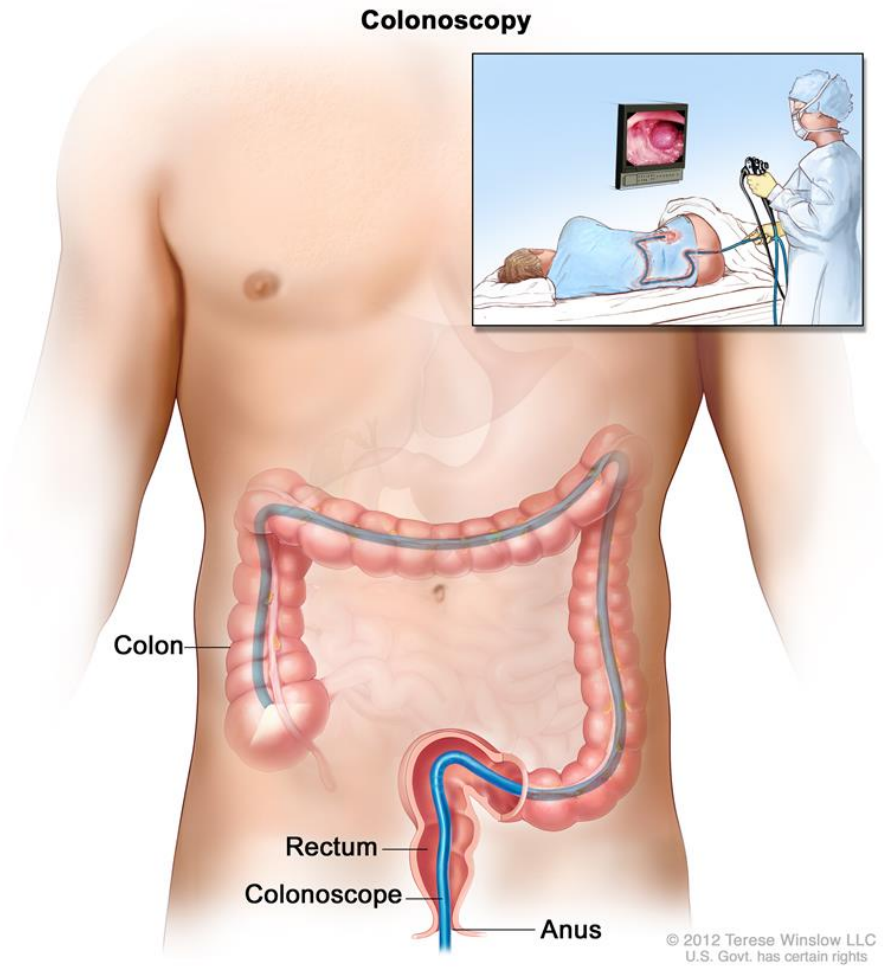


FIGURE 1.3: Figure illustrating a colonoscopy procedure. Image obtained from The National Cancer Institute.[2]

1.1.1 Limitations of colonoscopy and WCE

Despite the benefits, colorectal screening procedures do have their limitations. A colonoscopy is a manual procedure performed by a physician, which will be affected

[1] http://www.gastroenterologist-london.com
[2] https://www.cancer.gov

by human factors such as fatigue and experience. One study has estimated the polyp miss rate during a screening to be between 8-37% depending on the size and type of polyp [7]. Also, patients consider the procedure uncomfortable, both the preparations and the actual process. Furthermore, there is already a long waiting time associated with the procedure, and introducing a screening program could potentially increase the waiting time even further[3].

Utilizing WCE could alleviate some of these problems. It is much less invasive and does not require hospitalizing patients. Also, it has the benefit of being able to examine inaccessible regions, like the small intestine. However, there are difficulties linked with WCE, too. Between 30 and 60 thousand images are produced during the capsules passage through a patient. Processing such an amount would only increase the workload on physicians. Additionally, although it is a rare occurrence, the capsule can get stuck that might necessitate surgery for removal.

FIGURE 1.2: Example of capsule containing camera for WCE. Image obtained from West Thames Gastroenterology.[1]

Some of the limitations associated with both colonoscopy and WCE can be diminished by designing Decision Support Systems (DSSs). These are computer-based information systems that assist in decision making. During a colonoscopy, such a system could be consulted in challenging cases and act as a safeguard against human inaccuracy. For WCE, a reliable DSS could quickly analyze the images obtained from the procedure and single out images that need further investigation.

1.1.2 Designing DSSs

A popular approach for constructing DSSs is based on machine learning, a branch of artificial intelligence that intend for computers to solve tasks based on experience instead of being explicitly programmed. For detection of colorectal polyps, a physician might analyze images from a colonoscopy to provide an experts opinion of the images, often referred to as a ground truth or label. A machine learning algorithm could then be presented with the original images along with the corresponding labels provided by the physician and learn from these examples to produce similar outputs as the physician

[3] https://www.aftenposten.no/norge/i/8wjvd/Inntil-100-ukers-ventetid-for-tarmundersokelse

produced. However, there is a wide range of available machine learning algorithms and determining which to employ can depend on a wide range of factors, such as the data at hand or time-requirements. Furthermore, although results have been promising, machine learning based DSSs for finding colorectal polyps have yet to achieve the precision that physicians require.

1.2 Deep Learning

The performance of machine learning algorithms is very often dependent on the way the data is presented to the algorithm, often referred to as the representation of the data. Finding a good representation can be a time-consuming and challenging procedure, sometimes also requiring specific domain knowledge. Deep learning is a subfamily of machine learning which consists of algorithms that are capable of extracting a useful representation automatically from raw data, many of which have been known in the machine learning community since the late 1960's. However, deep models have, historically speaking, been very difficult to train and are associated with a high computational burden. As a result of these limitations, deep learning saw little research compared to other areas of machine learning. However, in 2012 the SuperVision group won the ImageNet Large Scale Visual Recognition Challenge[4], an annual competition for visual recognition tasks, by a remarkably large margin using a deep learning based approach. Their success spurred an explosion in deep learning research with deep models significantly improving the state-of-the-art in several computer vision tasks such as object location [8] and semantic segmentation [9].

Recent innovations and increased computing power were vital contributions to the deep learning renaissance, but the introduction of extensive datasets, containing millions of images, also played a crucial role. In addition to scaling in a superior way when presented with large-scale data compared to previous machine learning methods, the automatic representation of deep learning algorithms enabled them to harness the full potential of large and complex datasets resulting in an increased performance whenever applied to more massive datasets[10].

[4] http://www.image-net.org/challenges/LSVRC/2012/results.html

1.2.1 Deep Learning Difficulties

Deep learning based methods were quickly applied to different domains and provided state-of-the-art results, but one domain proved difficult to conquer, namely the medical domain. The reason for this difficulty is because several aspects of medical image analysis highlight many of the limitations that deep learning methods suffer from. First of all, patient privacy concerns make data sharing difficult and inhibits the development of large-scale datasets. However, recent years have seen the introduction of several large medical image datasets [11–13] providing more room for deep learning research. But the real limitations lie in the underlying understanding of models based on deep learning. Deep learning models can contain millions of parameters and give little or no indication as to the uncertainty in a prediction or what influenced the prediction in the first place. Such constraints have not obstructed deep learning from being widespread in many industrial application such as voice[5] and face recognition[6] or in music[7] and movie recommendations[8], where a poor decision will have little or no consequences. But for medical application, determining how certain and what influences a prediction is essential as it can be a matter of life and death. If methods based on deep learning are to form a reliable basis for DSSs in the medical domain, it would require better tools for understanding the predictions of the models and the models themselves.

1.3 Scope

This thesis will focus on development and evaluation of deep learning based models on the task of semantic segmentation of colorectal polyps. We seek to assess if deep learning based models can provide the necessary precision to act as a basis for DSSs that aim to benefit physicians. Furthermore, we also want to develop and evaluate methods that seek to increase understanding of deep models.

[5] https://research.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html

[6] https://machinelearning.apple.com/2017/11/16/face-detection.html

[7] https://medium.com/s/story/spotifys-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe

[8] http://www.cs.toronto.edu/~fritz/absps/netflix.pdf

1.4 Contributions

The main contributions of the thesis are the following:

- We develop and improve three recent deep learning based models for the task of semantic segmentation of colorectal polyps, two of which, to the best of our knowledge, has prior to this work not been applied in the field of polyp segmentation.
- We develop and evaluate a recent method for estimating uncertainty in models based on deep learning, a method that has, to the best of our knowledge, not been applied in the medical field previous to this work.
- We develop and evaluate a recent method for visualizing what features in the input data affect the predictions of models based on deep learning, a method that has, to the best of our knowledge, not been applied in the medical field previously.
- We propose a novel method for estimating uncertainty in input feature importance for predictions of models based on deep learning. To the best of our knowledge, we are not aware of any other methods capable of providing such quantities.
- We proposed to utilize an Information Theoretic Learning (ITL) framework for analyzing Fully Convolutional Networks (FCNs), an approach that, to the best of our knowledge, have yet to be analyzed in the field of deep learning prior to this work..

1.5 Notation

Unless otherwise stated, the following notation will be used throughout this thesis:

- Scalars will be written in lowercase, for example, x
- Random variables be written in uppercase, for example, X
- Vectors will be written in lowercase bold, for example, $\mathbf{x}$

- Matrices will be written in uppercase bold, for example, $\mathbf{X}$
- The transpose of a vector $\mathbf{x}$ or a matrix $\mathbf{X}$ will be written as $\mathbf{x}^T$ or $\mathbf{X}^T$
- $\mathbf{I}$ refers to the identity matrix.

1.6 Structure of Thesis

This thesis consists of six chapters, including this introductory chapter.

Chapter 2 presents previous work done on analysis of medical images containing colorectal polyps based on non-deep machine learning methods and describes the methods used in those works, along with the strength and weaknesses of such practices. The chapter continues by presenting studies done on analysis of medical images containing colorectal polyps using deep learning based methods and the prospect and limitation of these methods. This chapter aims to introduce the reader with some works that have been conducted on analysis of medical images containing colorectal polyps, to motivate why it is desirable to build DSSs based on deep learning based method and to highlight some of the issues associated with these methods.

Chapter 3 gives a detailed explanation of Deep Neural Networks (DNNs), followed by an introduction to Convolutional Neural Networks (CNNs). Additionally, the chapter concludes by presenting several techniques that can be used to provide a greater understanding of deep models. This chapter intends to give the reader a rigorous understanding of Deep Neural Networks and in particular Convolutional Neural Networks.

In Chapter 4 we provide a detailed introduction to the two novel methods we propose in this thesis. Additionally, Chapter 4 also describes the architecture we utilize in this thesis as well as the improvement we suggest to said architectures.

Chapter 5 deliver the results of all development and analysis conducted in this thesis, which includes quantitative and qualitative results on publicly available datasets, model comparisons, and results from several different techniques that seek to increase our understanding of DNNs.

Chapter 6 gives an overarching discussion of the results presented in Chapter 5, along with possible paths for future research or other aspects of the thesis that could warrant

further exploration. Additionally, we provide some concluding remarks and summarize the result of this thesis.

1.7 Notes from author

Parts of this work is submitted to the IEEE international Workshop on Machine Learning for Signal Processing in Aalborg, Denmark in September 2018. We are also working on a journal paper for the International Journal of Medical Informatics entitled "Understanding and Uncertainty in Convolutional Neural Networks for Polyp Segmentation" based on the work done in this thesis.

Details regarding data and methods, which may be important but not directly applied in this thesis are moved to the Appendix to benefit the reader. Also, interested readers can obtain all code utilized in this thesis online[9].

[9] https://github.com/Wickstrom/Thesis

Chapter 2

Background and Related Work

This chapter will review previous work done on the development of DSS for colorectal polyps and discuss some of their limitations. First, a general overview of machine learning will be presented, followed by a description of some popular methods for designing DSSs based on machine learning, accompanied by studies where they have been applied. Next, we give a high-level report of deep learning along with work done on DSSs based on such methods.

2.1 Machine Learning

Systems based on machine learning underpin a wide range of technologies regarded as staples of the modern day world: facial recognition systems, online recommendation engines, natural language processing and autonomous vehicles, to name a few. Machine learning is the science of giving computers the ability to learn from data without being explicitly programmed. Algorithms based on machine learning are often divided into three parts:

- **Supervised Learning** is the most common form of machine learning and also the form considered in this thesis. Given a pair of samples (x, y), the goal of supervised learning is to find a function f that maps x to a proper y, i.e. $f(x) = y$. Therefore, supervised learning can be thought of as function approximation where you have some data and want a mapping to a desired output.

- **Unsupervised Learning** is used when presented with unlabeled data or the desired outcome is unknown. Given a collection of N samples $x_1, x_2, ..., x_N$, the goal is to discover a compact description of the data. This could be an estimation of the underlying distribution or an attempt to group the data based on similarity, referred to as clustering.
- **Reinforcement Learning** is a form of machine learning that has seen a surge of research in recent years and stands out compared to supervised and unsupervised learning. In reinforcement learning, an agent interacts with an environment to produce action that can give a reward or punishment. Based on this feedback the agent automatically develops a policy that maximizes its performance.

There is a wide range of different algorithms available for supervised learning, each with their advantages and obstacles. Determining what algorithm to employ can depend on the task at hand, what kind of data is available and the designer's domain knowledge.

2.1.1 Machine Learning Based DSSs

Development of DSSs based on machine learning algorithm is a fruitful research area, with improvements and advances occurring continuously [14–17]. Describing all the different methods would be impractical, so instead, one of the most widely used algorithms is presented, namely the Support Vector Machine (SVM) [18]. There are several popular algorithms, like Random Forests (RFs) [19] or K-nearest-neighbour (KNN) classifier [20], which are capable of achieving comparable results to SVMs. But considering that SVMs has been considered state-of-the-art on many tasks [21, 22], their widespread use, and for brevity, the SVM will be used to highlight the strengths and weaknesses of machine learning algorithms for designing DSSs. Accompanying the description of SVMs is several examples where SVMs have been used to analyze medical images containing colorectal polyps.

2.1.2 Support Vector Machines

Support Vector Machines are supervised learning models commonly used for classification and regression. For classification, an object, described by l measurable quantities

$x_i,\ i = 1, 2, ..., l$, is to be assigned to a class. These measurable quantities are referred to as features, and together they form a feature vector

$$\mathbf{x} = [x_1, x_2, ..., x_l]^T.$$

This feature vector is accompanied by a label that indicates what class the object belongs to. For the two-class case, this label is represented by a scalar y, which takes the value 1 for the first class and -1 for the second class. Given a set of objects, the task of separating them into the correct class can be solved by analyzing what features differ between the objects. For a two-dimensional feature vector, it is possible to visualize the features with a plot, as illustrated in Figure 2.1. This example displays twenty objects, each described by two features, which can be assigned to one of two classes. One possible way of separating the two classes is to create a line and assign an object to either class depending on which side of the line it appears. For a general dimension, such a line is referred to as a hyperplane and can be described mathematically as

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b, \tag{2.1}$$

that multiplies the feature vector with a weight vector, $\mathbf{w} = [w_1, w_2, ..., w_l]^T$ and adds a bias b. The weights and bias, referred to as the parameters of the hyperplane, determine the separation boundary. However, as illustrated in Figure 2.1, there are several possible hyperplanes that separate the classes entirely, denoted by f_1, f_2 and f_3. How does one choose the hyperplane that discriminates these objects in the "best" possible way? Furthermore, how to find the parameters of this hyperplane?

One possible solution to the first question is to note that both the f_1 and f_2 hyperplane shown in Figure 2.1 have features located close to the hyperplane, while the f_3 hyperplane has a large margin to both classes. Intuitively, this leaves more "room" from the hyperplane to the features, which might generalize better if a new object with slightly different features is introduced. Finding the hyperplane with the maximum possible margin from the hyperplane to the features is the central idea of SVMs. It can be shown that[1] the distance of a point from the hyperplane is given by

[1] Showed in [23], for example.

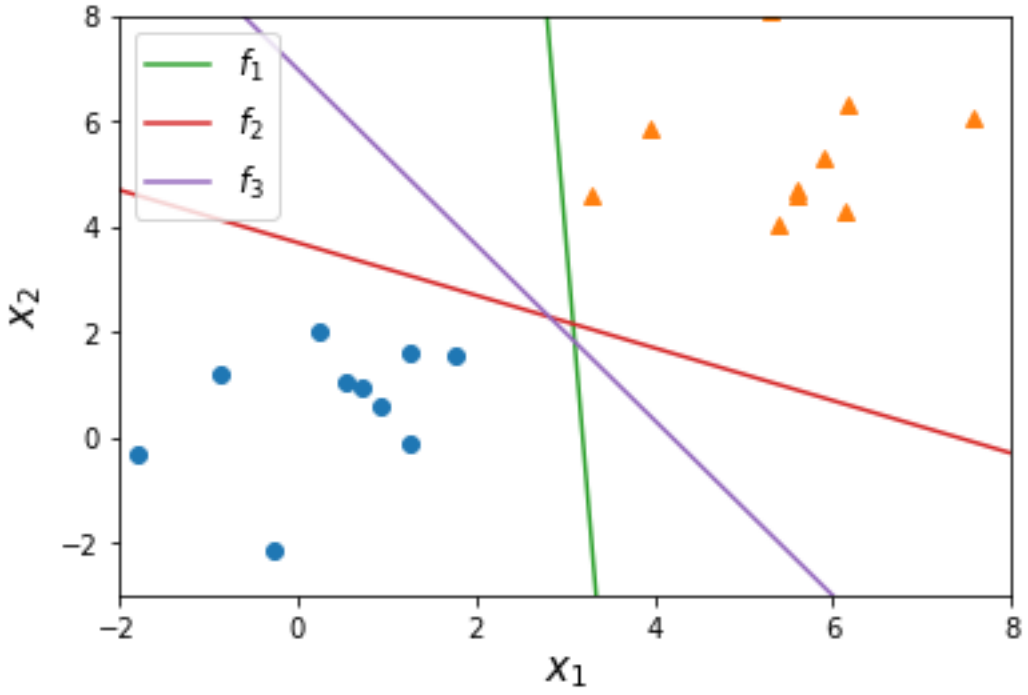


FIGURE 2.1: An example of a two-dimensional, two-class classification problem, where blue circles represents one class and orange triangles represent the other. Several possible hyperplanes that entirely separates the two classes are drawn.

$$z = \frac{|f(\mathbf{x})|}{||\mathbf{w}||} \tag{2.2}$$

However, each hyperplane is determined within a scaling factor $||\mathbf{w}||$. To avoid scaling issues the parameters are sized such that the value of $f(\mathbf{x})$, at the nearest points in class one or class two, is equal to one for class one and two for class two. This description is equivalent with

- Having a margin of $\frac{1}{||\mathbf{w}||} + \frac{1}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}$.
- Require that
  * $\mathbf{w}^T\mathbf{x} + b \geq 1, \quad \forall \mathbf{x} \in$ Class one
  * $\mathbf{w}^T\mathbf{x} + b \leq 1, \quad \forall \mathbf{x} \in$ Class two

So the SVM solution to the question of finding a hyperplane which optimally separates the classes is to find a hyperplane with the maximum possible margin from the hyperplane to the features. Next, we address the problem of finding the parameters

of this hyperplane. In machine learning, this is usually solved by introducing a cost function $\mathcal{C}$ to be optimized. This cost function can take many forms depending on the algorithm and the task[2], but for SVMs the cost is given by

$$\mathcal{C}(\mathbf{w}, b) = \frac{2}{||\mathbf{w}||^2}. \tag{2.3}$$

As explained, the objective of SVMs is to maximize the margin described above, which is equivalent to minimizing the inverted fraction. Minimum is achieved by tuning the weights $\mathbf{w}$ and bias b to the optimal configuration. Furthermore, because of the square in Equation 2.3 the optimization is convex. Convex optimization problems are desirable since a local minimum must be a global minimum. So the answer to the question of how to find the hyperparameters corresponding to the optimal hyperplane becomes; find the parameters of the hyperplane that achieves the maximum possible margin. We can find these parameters by

$$\text{minimze} \quad \mathcal{C}(\mathbf{w}, b) = \frac{2}{||\mathbf{w}||^2} \tag{2.4}$$

$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad i = 1, 2, ..., N \tag{2.5}$$

where N is the number of objects in the set. Finding these parameters is a nonlinear optimization task subject to a set of linear inequality constraints, often referred to as the primal problem. Solving such an optimization problem can be done using Lagrange multipliers, where the Lagrangian function is defined as

$$\mathcal{L}(\mathbf{w}, b, \lambda) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \lambda_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]. \tag{2.6}$$

Here, $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers λ_i. To proceed further it is required that the minimzer of Equation 2.4 and 2.5 satisfy the Karush-Kuhn-Tucker(KKT) conditions:

[2]See Appendix A.1 for a more detailed description of cost functions.

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \lambda)}{\partial \mathbf{w}} = 0 \tag{2.7}$$

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \lambda)}{\partial w_0} = 0 \tag{2.8}$$

$$\lambda_i \geq 0, \qquad i = 1, 2, ..., N \tag{2.9}$$

$$\lambda_i[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1] = 0, \qquad i = 1, 2, ..., N \tag{2.10}$$

Interested readers can turn to the Appendix A.2 for more details about the KKT condition. Combining Equation 2.6 with Equation 2.7 and 2.8 results in

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x} \tag{2.11}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0 \tag{2.12}$$

Equation 2.11 provides an expression to calculate the weights of the desired hyperplane and using 2.10 the bias can also be obtained. But it turns out that computing the parameters from the primal problem is computationally intractable. Instead, the problem can be solved by considering the Lagrangian duality. Stating the problem in its Wolfe dual representation form [23], i.e.

$$\text{maximize} \quad \mathcal{L}(\mathbf{w}, b, \lambda) \tag{2.13}$$

$$\text{subject to} \quad \mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x} \tag{2.14}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0 \tag{2.15}$$

$$\boldsymbol{\lambda} \geq \mathbf{0} \tag{2.16}$$

Substituting Equation 2.14 and 2.15 into Equation 2.13 accompanied by some algebra results in

$$\max_{\boldsymbol{\lambda}} \Big(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{ij} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \Big) \tag{2.17}$$

$$\text{subject to} \sum_{i=1}^{N} \lambda_i y_i = 0 \tag{2.18}$$

$$\boldsymbol{\lambda} \geq \mathbf{0} \tag{2.19}$$

There are many proposed algorithms for finding the optimal Lagrange multiplier, for example, the Sequential Minimal Optimization (SMO) algorithm [24]. Once the optimal Lagrange multipliers have been found, the parameters corresponding to the optimal hyperplane is obtained using Equation 2.14 for the weights and via Equation 2.10 for the bias.

2.1.2.1 Nonlinear SVMs

Figure 2.1 displays an example where the two classes are linearly separable. However, more complex problems can be more difficult to separate, and it might require a nonlinear boundary to separate the classes optimally. At first glance, it might not be obvious how SVMs are extended to enable nonlinear separation, but an elegant approach, know as the kernel trick [23], allows SVMs to transform from a linear to a nonlinear algorithm. The central idea is to map the features to a new higher dimensional space (possibly infinite) where the features are more easier to separate. Equation 2.17 displays that the feature vectors occur in pairs, via the inner product operation. If the separation is to take place in a new k-dimensional space, the only difference would be the additional mapping of the original feature vectors. Suppose that there is some mapping ϕ

$$\mathbf{x} \mapsto \phi(\mathbf{x}) \in H, \quad \mathbf{x} \in \mathcal{R}^l$$

where H is a Hilbert space. The inner product between a pair of feature vectors in Equation 2.17 would now become $\phi(\mathbf{x}_i)^T \phi(\mathbf{x_j})$, which has an equivalent representation

$$\phi(\mathbf{x})^T \phi(\mathbf{x}) = \kappa(\mathbf{x}_i, \mathbf{x}_j) \tag{2.20}$$

Here, $\kappa(\cdot)$ is known as a kernel function, which corresponds to an inner product in some alternative feature space if it satisfies Mercer's conditions [25]:

$$\int \kappa(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_i) g(\mathbf{x}_j) d\mathbf{x_i} d\mathbf{x_j} \geq 0 \tag{2.21}$$

for any $g : \mathcal{R}^l \to \mathcal{R}$ such that

$$\int g(\mathbf{x}_i) d\mathbf{x} < +\infty \tag{2.22}$$

Furthermore, κ must be continuous, symmetric, and have a positive definite Gram matrix, where a Gram matrix is a matrix consisting of the inner products between all pairs of vectors in a set of vectors. So even though the mapping is not known the dot product in that space will have the same value as the kernel function, which is efficient and straightforward to compute. There are several available kernels, but one of the most common is the Radial Basis Function (RBF)

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\Big(- \frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{\sigma^2} \Big) \tag{2.23}$$

where σ is a kernel-parameter that must be tuned by the designer.

2.1.2.2 DSSs Using Support Vector Machines

In general, when analyzing medical images containing colorectal polyps, there are two questions of particular interest:

- Is there a polyp present in the image? (Detection)
- If so, where is it? (Localization)

For detection, the algorithm must classify an image as containing a polyp or not containing a polyp, where images are labeled by physicians such that the ground truth is known. One of the first steps to designing DSSs based on SVMs is deciding what features will be presented to the algorithm. In [26] they divide the original image of resolution (768×576) into sub-images of resolution (40×40), where each pixel

is labeled as polyp or not polyp. Each pixel is represented by 5 features; its RGB components and its coordinated in the sub-image, resulting in 8000 features for a single sub-image. This sub-image is then classified as containing a polyp or not based on the number of pixels classified as a polyp, and if a sub-image is classified as containing a polyp, then the parent image is also classified as containing a polyp. Pixels are classified using a nonlinear SVMs with an RGB kernel.

Merely using the RGB components and the coordinates of a pixel as features can be compelling, but for more challenging problems it might not be sufficient. There is a vast number of algorithms for finding discriminative features to present a classifier with, and we will give a brief description of some popular approaches. One of the most widely used algorithms for extracting feature for a machine learning algorithm is the Scale Invariant Feature Transform (SIFT) algorithm [27]. Extracting SIFT features follow four main stages. First, potential interest points are located through a scale-space extrema detection using edge detecting filter of different size. Secondly, a Taylor expansion is used to determine the location and scale of a potential interest point and key points are selected based on a measure of their stability. Thirdly, each key point is assigned a direction to obtain invariance to image rotation. This is achieved by considering a neighbourhood around each key point and calculating the gradient magnitude and direction. Lastly, our feature vector is constructed by considering a neighbourhood around a key point, a so-called key point descriptor, and calculating the gradient magnitude and orientation at each point in the neighbourhood. This neighbourhood is split into smaller subregions where gradient magnitude and orientation is calculated once again. The size of the neighbourhood and subregions will then decide the length of the feature vector. SIFT features are invariant to scale and rotation and know to produce highly discriminative features. Also, by focusing on key points instead of all samples, it eases the computational burden for the following classifier. Several works have been done with SIFT features for detection of colorectal polyps that have shown improved performance compared to previous methods [28].

Another popular algorithm is the Histogram of Oriented Gradients (HOG) algorithm that was popularized in the machine learning community in 2005 [29]. HOG features are constructed by splitting an image into equal sized neighbourhoods and computing the gradient of each point in that neighbourhood. Next, histograms are calculated for each neighbourhood based on the gradients. These histograms are used to construct the feature vectors that are passed on to the classifier. HOG features have also been explored in the context of colorectal polyp detection and shown good performance [30].

However, in many cases it is not enough to know that the polyp is present, it needs to be located as well. For location, each pixel is classified as polyp or non-polyp, so-called semantic segmentation. Numerous methods have been proposed for segmentation of images [31, 32] and some have been designed specifically for semantic segmentation of colorectal polyp. In [33] they propose a two-step procedure for segmenting polyps that is based on two assumptions. First, the center of a polyp has a negative maximum principal curvature, that is, the colon curves downwards from the center of the polyp. Second, the polyp is delimited by positive values of maximum principal curvature. The first step of the procedure estimates coarse curvature information and the second step refines the coarse prediction to obtain a finer segmentation. Another method is based on the assumption that valleys should surround a polyp in several directions [34], where valleys are detected through a valley detector based on gradient information.

2.1.2.3 Advantages and Limitations of SVMs

Building DSSs based on SVMs provides reliable systems with high precision, especially for detection. Furthermore, SVMs have a strong theoretical foundation that makes model interpretability straightforward. However, there are several complications with SVM-based DSSs. Firstly, the computational burden is high when processing high-dimensional data like images and may require partitioning the images or creating algorithms that extract regions of interest. Additionally, complicated tasks might need more features to obtain acceptable performance, which increases computational issues further. Secondly, features must be extracted manually. Determining what features to choose and how to extract them is a complicated task in itself, and might require domain knowledge to get optimal results. Lastly, SVMs have yet to achieve satisfactory results on segmentation of colorectal polyps, inhibiting DSSs that are supposed to aid physicians in locating polyps. Many of these problems are not unique to SVMs. All traditional machine learning methods require manual crafting of features and are limited by their high computational requirements. Moreover, other machine learning algorithms also struggle with segmentation tasks. These are problems that needs to be addressed if reliable DSSs are to be based on machine learning algorithms.

2.2 Deep Learning

In recent years, deep learning methods have provided significant advances in several computer vision tasks such as image classification [10, 35], object detection [36–38] and image segmentation [9]. Conventional machine learning methods are dependent on the data representation (or features). Transforming raw data into a representation that is suitable for the machine learning algorithm can be time-consuming and might require significant domain knowledge. Deep learning methods tackle the representation issue by stacking multiple processing layers in succession that automatically transforms raw, unprocessed data into a more abstract and useful representation.

2.2.1 Deep Learning Based DSSs

Research on DSS design has shifted toward deep learning based approaches during the last couple of years. This shift is especially true for image analysis, where recent years have seen over 300 contributions to the field [39]. In the following examples we will illustrate how CNNs can perform detection and localization of colorectal polyps. Since the succeeding chapter includes a detailed description of CNNs these examples will be given a "high-level" description, aimed to show the promise of deep methods for further research and their limitations.

2.2.2 DSS using Convolutional Neural Networks

Several studies have been done on polyp detection and localization using CNNs, where the majority has been made on detection. In [40] they employ a CNNs inspired by the LeNet-5 [41] to classify an image as containing or not containing a polyp. To deal with the lack of data they use a patch-based approach, where sub-images are extracted from the original image. This extraction provides more training data and reduces the number of units required in the fully connected layers toward the end of the network. During inference, the final decision is obtained through majority voting of several sub-images extracted from the full-sized test image. This approach yielded superior performance compared to previous methods based on manually extracted features. Additionally, to increase model interpretability, filters from the first convolutional layer is visualized. These filters display that the network has learned a collection of filters

such as edge detectors and texture extractors. In a recent masters thesis they also explored the prospect of detecting polyps using CNNs, but employ a more recent architecture [42], which produced encouraging results. However, this patch-based approach is computationally demanding, particularly during inference, and does not give any information regarding the position of the polyp if it is present.

Another more recent study used an extension of CNNs known as Fully Convolutional Networks (FCNs) [9], particularly suited for per-pixel predictions such as semantic segmentation [11]. These networks resemble typical CNNs but perform upsampling to recover the resolution of the original image, thus enabling per-pixel classification. Another benefit is that FCNs are capable of processing images of arbitrary size and can, therefore, utilize the patch-based approaches for training but process the entire test image in one pass through the network during inference. Results showed a significant improvement over previous approaches, yielding precise segmentation maps with no further post-processing. However, their model lack interpretability and provide no notion of uncertainty.

2.2.2.1 Advantages and Limitations of CNNs

Convolutional Neural Networks tackle many of the issues that traditional machine learning algorithms suffer from. They have significantly improved performance on both detection and segmentation tasks [9, 10], approaching the necessary precision required for medical applications. Also, since the network molds the features into the ideal form for discrimination the time consuming and complicated process of handcrafting features is removed. However, DNNs introduce their own set of obstacles that demand attention. First of all, they require large amounts of data to tune the millions of parameters, which can be challenging in the medical domain. Also, the large number of parameters makes the model capable of learning the training data, so-called overfitting, which might generalize poorly to unseen data. Another problem with a large number of parameters is the lack of transparency. It can be difficult to asses what influence a decision or what parameters are affected by what features, which can make deep models less trustworthy. Furthermore, DNNs have no clear way of representing uncertainty in a prediction, which add to the trust issues. Lastly, there is a wide range of available models aimed at the same task. Deciding which model suits the task at hand lacks a solid theoretical foundation and one must often resort to heuristics. Tackling these

issues is crucial if CNNs are to become a precise and trustworthy component of DSSs and in this thesis, we look at several ways to address the problems stated here.

Chapter 3

Deep Neural Networks and Convolutional Neural Networks

While the previous chapters have motivated the desire to evaluate and develop deep learning methods for medical image analysis, this chapter will give a detailed description of how it is done. First, we look at the general workings of a standard feedforward network, its central components and some essential techniques associated with such networks. Next, CNNs are introduced and explained. Finally, recent methods associated with increasing the interpretability of CNNs and providing uncertainty estimates are presented.

3.1 Deep Feedforward Networks

Feedforward networks also called Multilayer Perceptrons (MLPs), form the bedrock for all of deep learning. Feedforward networks stack layers of simple mappings and transformation in a hierarchical fashion that results in function approximators of universal capabilities under certain assumptions [43]. For a task like classification, $y = f(x)$, an input x is assigned to a class y by a function f. A feedforward network defines a mapping $y = \hat{f}(x;\theta)$ and learns the value of the parameter θ that results in the best function approximation $\hat{f}$ of the true function f. Although not all deep learning models are focused on finding a deterministic function, they all employ the idea of hierarchically stacking mappings and transformations.

We will start by introducing the MLP and the general procedure for deploying such networks, and then the following sections will explore important concepts related to MLPs in further detail.

3.1.1 Multilayer Perceptron

As already stated, MLPs are constructed by stacking layers of mappings and transformations in succession. Figure 3.1 display a typical MLP, which consists of an input layer, three hidden layers and one output layer, where each of these layers contains a number of units, often referred to as neurons. In the general case, we assume that a network consists of L layers, with k_0 units in the input layer and k_l units in the hidden layers, where $l = 1, ..., L$. The input layer represents the data passed into the network; no actual operations are carried out in this layer. In the example network shown in Figure 3.1 the input layer contains four units corresponding to the number of features used to represent a sample of this particular data. At the end of the network, we find the output layer, which corresponds to the network prediction for a given feature vector. In this example, the output layer has two units that could correspond to two classes in a classification problem. Between the input and output layer, we find one or more hidden layers. These are responsible for mapping and transforming the input into a representation where the output layer can optimally perform the desired task. By including more layers, more units or both, we can increase the capacity of the network to handle more complex data. Increasing the number of units is referred to as increasing the "width" of the network while increasing the number of layers is referred to as increasing the "depth" of the network, which is where the "deep" part of Deep Feedforward Networks originates.

For each neuron a weighted sum is computed by multiplying the output of the previous layer with the weight and bias of the current layer, that is

$$z_j^{(l)}(i) = \sum_{j=1}^{k_l} \sum_{k=1}^{k_{l-1}} w_{jk}^{(l)} a_k^{(l-1)}(i) + b_j^{(l)}, \qquad i = 1, ..., N \tag{3.1}$$

where $b_j^{(l)}$ is the bias of the j^{th} unit in the l^{th} layer, $w_{jk}^{(l)}$ is the weight connecting the k^{th} unit in the $(l-1)^{\text{th}}$ layer with the jth unit in the lth layer, k_l is the number of units in the l^{th} layer, k_{l-1} is the number of units in the $(l-1)^{\text{th}}$ layer, N is the total

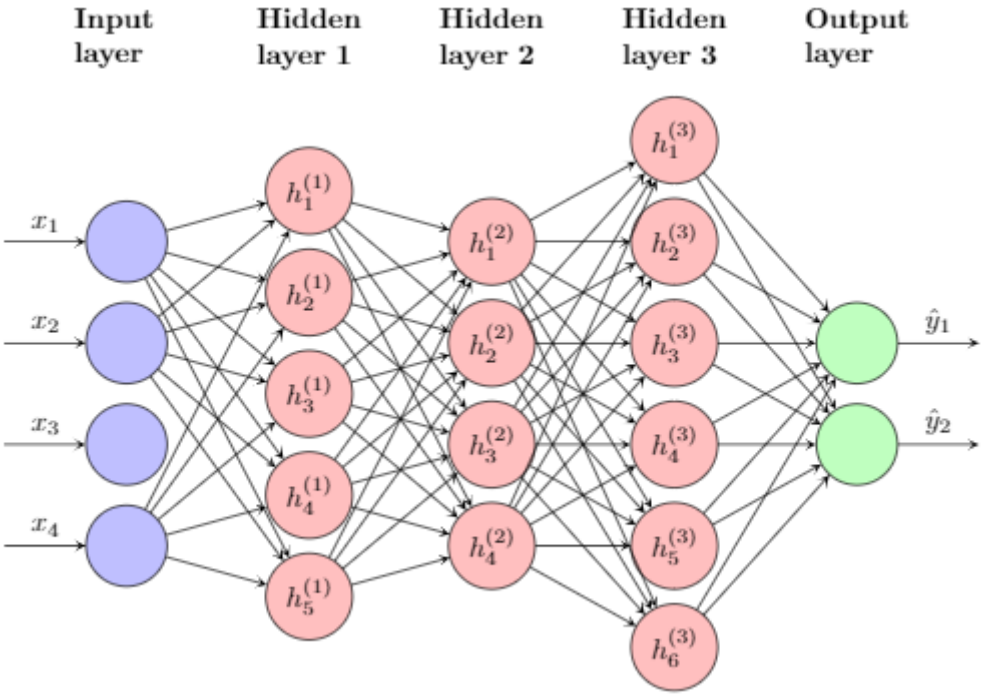


FIGURE 3.1: Figure illustrating a typical MLP with three hidden layers consisting of five, four and six units and two output units.

number of samples and $a_k^{(l-1)}(i)$ is the output of the k^{th} unit in the $(l-1)$th layer. This weighted sum is passed into a non-linearity called an activation function $f(\cdot)$, which acts as approximate unit step function to indicate unit activation and results in the output of a unit,

$$a_k^{(l)}(i) = f(z_j^{(l)}(i)). \tag{3.2}$$

When $l = L$, that is, the network output, $a_k^l(i) = \hat{y}_k(i)$, $k = 1, ..., k_L$, and when $l = 1$, $a_k^l(i) = x_k(i)$, $k = 1, ..., k_0$, that is, the network input.

Multilayer perceptrons can be utilized to a wide range of different tasks, but they all share a common goal, to optimize a cost function on some test data, where the cost function is dependent on the task. Interested readers can read more about cost functions in Section A.1 of the Appendix, but one common choice in neural networks is the sum of squared errors cost function, such that

$$\mathcal{C} = \sum_{i=1}^{N} \mathcal{E}(i) = \frac{1}{2} \sum_{i=1}^{N} e_m^2(i) = \frac{1}{2} \sum_{i=1}^{N} \sum_{m=1}^{k_l} (y_m(i) - \hat{y}_m(i))^2 \tag{3.3}$$

where N is the number of samples, k_L is the number of output units and $\hat{y}_m(i) = a_k^L(i)$, that is network output. If the task is classification, we usually want to minimize the error computed by the cost function. However, since the test data is not available during the training of a model, we use training data to approximate the test error and minimize the error based on the training set instead.

3.1.2 Backpropagation and Gradient Descent

Minimization of the cost function can be acquired by iteratively updating the weights of the network, commonly done using the gradient descent algorithm. However, there is a wide range of modern optimization algorithms that build upon the standard gradient descent algorithm that are discussed in Appendix B.1. The gradient descent algorithm iteratively updated the weights and biases with the update rules

$$w_j^{(l)}(\text{new}) = w_j^{(l)}(\text{old}) - \mu \frac{\partial C}{\partial w_j^{(l)}} \tag{3.4}$$

$$b_j^{(l)}(\text{new}) = b_j^{(l)}(\text{old}) - \mu \frac{\partial C}{\partial b_j^{(l)}} \tag{3.5}$$

where μ is a positive hyperparameter known as the learning rate. A large learning rate results in faster training but might miss a good minimum, while a small learning rate might not reach a good minimum at all. Determining the learning rate is, therefore, an integral part of network design that recent algorithms has sought to automatize (see Appendix B.1). Both Equation 3.4 and 3.5 requires computing the derivative of the cost function with respect to the parameters of the network. Computing these derivatives is done using the backpropagation algorithm, introduced by Werbos [44] and popularized by Rumelhart *et.al* [45]. From Equation 3.3 we can see that the costs dependency on the parameters passes through $z_j^l(i)$. We consider the weights first and use the chain rule, which gives

$$\frac{\partial \mathcal{E}(i)}{\partial w_j^{(l)}} = \frac{\partial \mathcal{E}(i)}{\partial z_j^{(l)}(i)} \frac{z_j^{(l)}(i)}{\partial w_j^{(l)}}. \tag{3.6}$$

Differentiating Equation 3.1 with respect to the weights yields

$$\frac{z_j^{(l)}(i)}{\partial w_j^{(l)}} = a_k^{(l-a)}(i), \qquad k = 1, ..., k_{l-1}. \tag{3.7}$$

Next we define

$$\frac{\partial \mathcal{E}(i)}{\partial z_j^{(l)}(i)} = \delta_j^{(l)}(i), \tag{3.8}$$

which gives the following update rule for the weights,

$$w_j^{(l)}(\text{new}) = w_j^{(l)}(\text{old}) - \mu \sum_{i=1}^{N} \delta_j^{(l)}(i) a_k^{(l-1)}(i) \tag{3.9}$$

and using the same procedure, the following update rule for the bias

$$b_j^{(l)}(\text{new}) = b_j^{(l)}(\text{old}) - \mu \sum_{i=1}^{N} \delta_j^{(l)}(i). \tag{3.10}$$

To update the weights, we need to compute the gradients of each layer. Assuming the sum of squared errors cost function from Equation 3.3 is used, it can be shown that the gradients of the output layer can be computed by

$$\delta_j^{(L)}(i) = e_j(i) f'(z_j^{(L)}(i)), \tag{3.11}$$

and for the remaining layers, the gradients can be computed by

$$\delta_j^{(l-1)}(i) = e_j^{(l-1)}(i) f'(z_j^{(l-1)}(i)), \tag{3.12}$$

where

$$e_j^{(l-1)}(i) = \sum_{k=1}^{k_l} \delta_k^{(l)}(i) w_{kj}^{(l)}. \tag{3.13}$$

The only component that has not been addressed is $f'(z_j^{(l-1)}(i))$, where the derivative of the activation function must be computed. Choice of activation function is an important part of network design and will be explored thoroughly in Section 3.1.4. To summarize, backpropagation can be performed using the following procedure:

1. *Initialization*: Initialize all the weights and biases according to some initialization scheme (see Section 3.2.2 for further details).
2. *Forward pass*: For all training samples, compute the activation of each unit using Eq 3.2 and evaluate the cost using the current parameters.
3. *Backward pass*: Compute the gradients of all layers using Eq 3.11, 3.12 and 3.13.
4. *Update parameters*: Update all parameters using Equation 3.9 and 3.10.
5. *Iterate*: Repeat steps 2-4 until convergence.

3.1.3 Vanishing and Exploding Gradients

One of the fundamental obstacles of DNNs is the vanishing and exploding gradients problem [46]. As we move backward through the network, the gradients tend to get smaller, which causes the units in the early layers to train more slowly. To illustrate the problem we consider a simple neural network, shown in Figure 3.2, consisting of one neuron in each layer and two hidden layers. Seeing that, for this simple case, all quantities are scalars, we simplify the notation for the benefit of the reader and drop the sample index i and neuron indices j and k. For the output layer, where $l = L$, the gradients can be found using Equation 3.11, resulting in

$$\delta^{(l)} = e^{(l)} f'(z^{(l)}) = (y - \hat{y}) f'(z^{(l)}), \tag{3.14}$$

where we have assumed the sum of squared errors loss function from Equation 3.3. To find the gradients of the second hidden layer, we use Equation 3.12, which gives

$$\delta^{(l-1)} = e^{(l-1)} f'(z^{(l-1)}), \qquad \text{Use Eq 3.13 to find } e^{(l-1)} \tag{3.15}$$

$$\delta^{(l-1)} = \delta^{(l)} w^{(l)} f'(z^{(l-1)}) \tag{3.16}$$

$$\delta^{(l-1)} = (y - \hat{y}) f'(z^{(l)}) w^{(l)} f'(z^{(l-1)}). \tag{3.17}$$

Following the same strategy, the gradients of the first hidden layer can be found, and we get

$$\delta^{(l-2)} = (y - \hat{y}) f'(z^{(l)}) w^{(l)} f'(z^{(l-1)}) w^{(l-1)} f'(z^{(l-2)}). \tag{3.18}$$

From Equation 3.18 we can see that each layer adds a factor of $f'(z^{(\cdot)})w^{(\cdot)}$ to the derivatives. As we will see in Section 3.1.4, the derivative of the activation function tend to have a low value. For example, the derivative of the most common activation functions, the sigmoid, is always less than or equal to 0.25. As networks grow deeper and more factors of $f'(z^{(\cdot)})$ are added, the gradients will vanish if the weights are not sufficiently large to counter the effect. But large weights are likely to cause the opposite effect, namely exploding gradients, where the gradients become very large. Development of new activation functions that address the vanishing gradient problem has been one of the key components of DNNs recent success and discovering more effective activation function is still an active field of research.

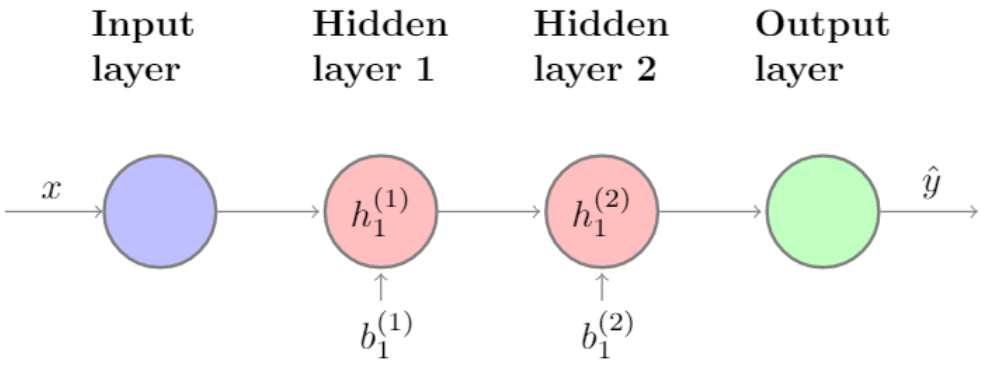


FIGURE 3.2: Figure displays a simple neural network for demonstrating vanishing and exploding gradients.

3.1.4 Activation Function

As mentioned in Section 3.1.1, the activation function acts as an approximation of the unit step function, also referred to as a linear threshold function, which indicates unit activation and enables the hidden layers to discover non-linear transformations of the input. Early research in neural networks were inspired by neurons in the brain, where a neuron would activate if the strength of the received input signal surpassed a certain threshold [47, 48]. Initial models of artificial neurons deployed a linear threshold function as activation function, shown in Figure 3.3, which produces a binary output depending on a set threshold. These early models had potential but were restricted by their binary output and the fixed threshold. Furthermore, backpropagation requires a differentiable activation function which excludes the linear threshold unit.

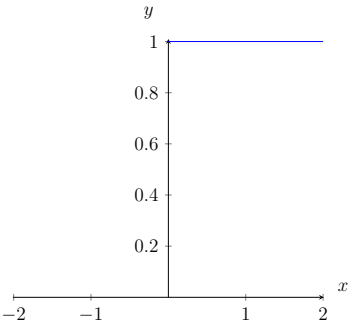


FIGURE 3.3: Linear threshold function with threshold at $x = 0$ and binary output y.

Traditional Activation Functions

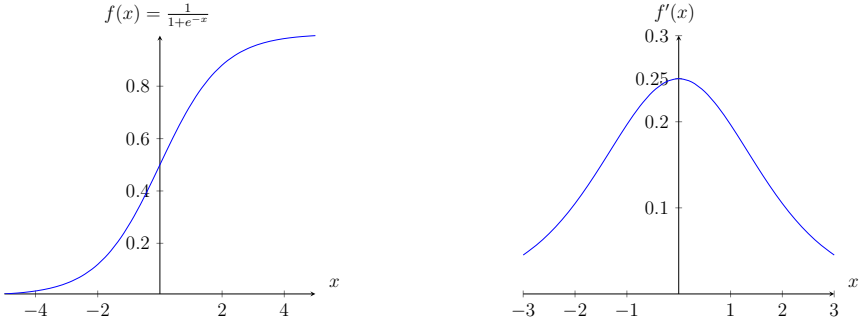
Historically speaking, the most common activation function has been the sigmoid function, which is defined as

$$f_{\text{sig}}(x) = \frac{1}{1 + \exp(-x)} \tag{3.19}$$

and can be seen in Figure 3.4a. Inputs are squashed between 0 and 1 which represents the potential for a neuron to "fire", where we generally assume a neuron to be firing if the output is above 0.5. For large positive or negative input values a sigmoid unit saturates, i.e. the output of the unit approaches 1 and 0, respectively, to indicate that the unit is very certain about firing or not firing. In Section 3.1.2 we needed to compute the derivative of the activation function, which for the sigmoid is

$$f'_{\text{sig}}(x) = f(x)_{\text{sig}}(1 - f(x)_{\text{sig}}) \tag{3.20}$$

and can be seen in Figure 3.4b. From Figure 3.4b we can see the magnitude of the sigmoid derivative will always be less than or equal to 0.25, and as discussed in Section 3.1.3, when the derivatives of the activation is small and many of these small derivatives are multiplied together the gradients tend to diminish or vanish altogether. This effect is reinforced as networks grow deeper and is why the sigmoid activation is rarely used in DNNs.



(A) The sigmoid function. (B) Derivative of the sigmoid function.

FIGURE 3.4: Figure displays the sigmoid function and its derivative.

Another commonly applied activation function is the hyperbolic tangent function, often simply referred to as tanh, which is defined as

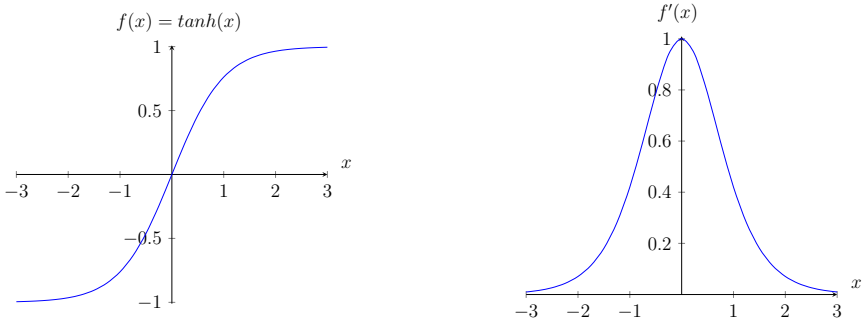
$$f_{\text{tanh}}(x) = \frac{2}{1 + \exp(-2x)} - 1 \tag{3.21}$$

and can be seen in Figure 3.5a. Similarly to the sigmoid it also squashes the input into a fixed range, but for the tanh this range is between -1 and 1, where positive values indicate an active unit and negative values indicate an inactive unit. The tanh and sigmoid function are similar in many ways and related by the expression $\tanh(x) = 2\,\text{sigmoid}(2x) - 1$, that is, a scaled version of the sigmoid. The derivative of tanh is

$$f'_{\text{tanh}}(x) = 1 - f(x)^2_{\text{tanh}} \tag{3.22}$$

and can be seen in Figure 3.5b. Figure 3.5b shows that the tanh allows for larger gradients that could help with the vanishing gradients problem, but experiments [10]

have shown that for deep networks using tanh units exhibits the same problem as the sigmoid and is, therefore, also, rarely used in DNNs.



(A) The tanh function.

(B) Derivative of the tanh function.

FIGURE 3.5: Figure displays tanh function and its derivative.

New Activation Functions

One of the key components to the recent success of DL is the adoption of a new activation function that avoids the vanishing gradient problem, the Rectified Linear Unit (ReLU)[49–51]. If a single value x is considered, the ReLU is defined as

$$f_{\text{ReLU}}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases} \tag{3.23}$$

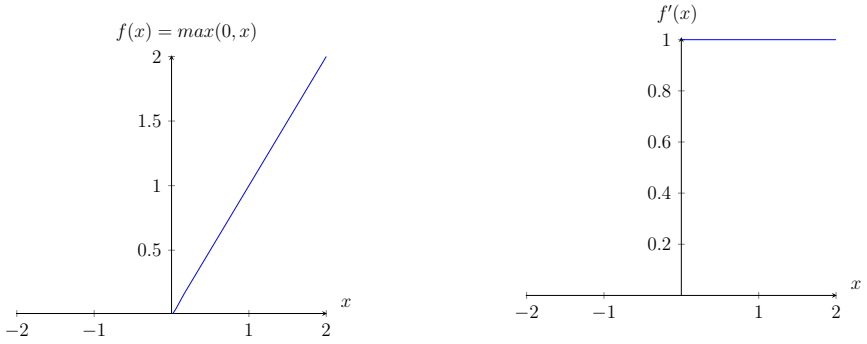
and can be seen in Figure 3.6a. To understand why the ReLU diminish the vanishing gradient problem we look at the derivative, which is

$$f'_{\text{ReLU}}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}. \tag{3.24}$$

For positive inputs, the gradient is always equal to one, which evades the vanishing gradients problem. Another advantageous attribute of the ReLU is sparse activations, as sparse representations are more likely to produce disentangled information disentangled and information that is more likely to be linearly separable [49].

There are some concerns associated with the ReLU. One is the possible blocking of gradients from the hard saturation at $x = 0$, but experimental results have shown

that the opposite is true and that the hard zeros can actually help the supervised training so long as the gradient can propagate along some paths [49]. Another is the unbounded behaviour of the activation that can be handled by restricting the weights of the network, as explained in Section 3.2.1. Lastly, we have the "dying ReLU" issue, which might occur if large gradients flows through the network and updates the parameters in such a way that it never activates again. For example, a large negative bias will effectively render a unit inactive and unable to recover, since its gradient will remain zero. Several modifications of the ReLU has been proposed to solve the "dying ReLU" problem. One extension is the Leaky ReLU, proposed by Maas *et al.,* [52], which allows for a small, non-zero gradient when the unit is saturated and not active. Another popular approach is the Parametric ReLU (PReLU), proposed He *et al.,* [53], which adds a parameter that controls the slope of the negative part for each unit. This extra parameter is learned as part of the backpropagation, adds negligible computation cost and has been shown to increase performance [53].



(A) The ReLU activation function.

(B) Derivative of the ReLU activation function.

FIGURE 3.6: Figure displays the ReLU and its derivative.

Softmax Function

To conclude the description of activation functions we introduce the softmax function that is, strictly speaking, not an activation function yet a standard inclusion in both neural networks and DNNs that perform classification. Softmax is a generalization of the sigmoid function that "squashes" a vector of arbitrary values into the range $(0, 1)$ and is generally used at the output layer of a network to produce a probability distribution over all the classes in the dataset. More formally, for a network consisting of L layers tasked with assigning a data point into one of C classes, the softmax function is defined as

$$\hat{y}_c = \frac{\exp(z_c^{(L)})}{\sum_{c=1}^{C} \exp(z_c^{(L)})}, \ c = 1, 2, ..., C \tag{3.25}$$

where $z_c^{(L)}$ is the output of the network and $\hat{y}_c$ can be interpreted as the probability of the given data point to be assigned to class c. However, this interpretation can be ill-advised, as we will explain in Section 3.5.1.

3.2 Overfitting and Regularization

Deep Neural Networks can have millions of free parameters, enabling the possibility to model a wide range of complex phenomena. However, a sufficiently large network might memorize peculiarities of the training data, achieving high performance on the training set without discovering the actual underlying distribution of the data that results in poor performance on the test set. In such cases, we say that the network is *overfitting*. Figure 3.7 displays a typical example of overfitting, where the training error keeps decreasing while the test error starts increasing. One approach to counter overfitting is to reduce the number of free parameters, i.e. reduce the capacity of the network. But determining the number of parameters needed is not a trivial task, and too few parameters might lead to the opposite effect, namely underfitting, where the network has insufficient capacity to model the data. Instead, we deploy a range of different techniques to reduce overfitting, referred to as regularization techniques, which aim to restrict the network such that strong generalization capability is encouraged. This section will give an overview of the most common techniques and how to apply them.

3.2.1 Parameter Penalties / Weight Decay

One of the most common techniques to prevent overfitting is penalizing large-valued weights since those weights tend to lead to overfitting [54]. Parameter penalization, sometimes also referred to as weight-decay, is achieved by including a penalty term in the cost function, such that Equation 3.3 becomes

$$\mathcal{C} = \sum_{i=1}^{N} \mathcal{E}(i) + \alpha h(\mathbf{W}) \tag{3.26}$$

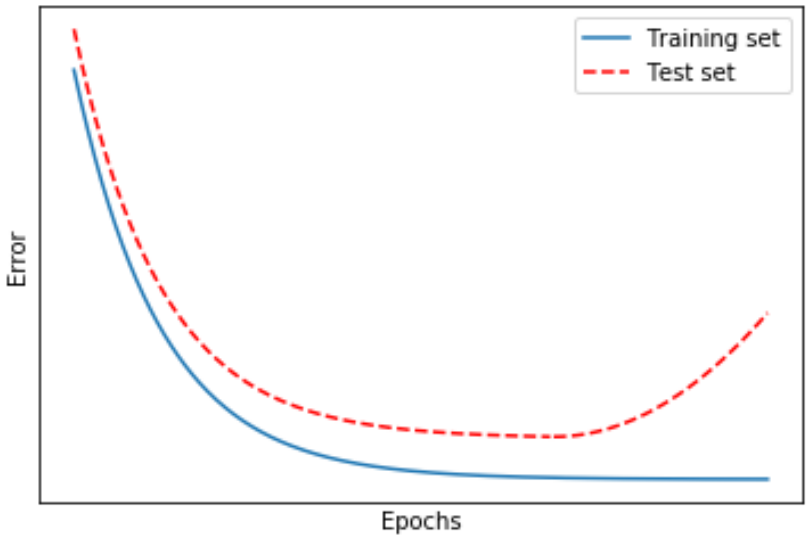


FIGURE 3.7: Error as a function of iteration steps for training and test set.

where $\mathbf{W}$ is a matrix containing the weights of all layers in the network, $h(\cdot)$ is an appropriately chosen differentiable function, and α is a hyperparameter referred to as the regularization parameter that controls how "hard" the weights are penalized. Notice that the bias terms are usually not penalized. Hinton argues that since there are far fewer biases, they are less likely to cause overfitting [55]. Also, in some cases the bias might need to be large and imposing a penalty will only increase the time to reach the required size for the bias. One of the most widely used forms of parameter penalization is penalizing the sum of the squares of the weights, known as Tikhonov regularization or L2-regularization [56], such that the cost function becomes

$$\mathcal{C}_{L2} = \sum_{i=1}^{N} \mathcal{E}(i) + \frac{1}{2}\alpha \sum_{l=1}^{L} \sum_{k=1}^{k_l} w_{lk}^2 \tag{3.27}$$

where w_{lk} refers to lk^{th} element of the weight matrix $\mathbf{W}$, L refers to the number of layers in the model and k_l refers to the number of units in the l^{th} layer. L2-regularization drives the weights closer to the origin that inhibits them from growing to large [57]. Another popular penalization technique is L1-regularization [56], which penalizes the sum of the absolute values of the weights, such that the cost function becomes

$$\mathcal{C}_{L1} = \sum_{i=1}^{N} \mathcal{E}(i) + \alpha \sum_{l=1}^{L} \sum_{k=1}^{k_l} |w_{lk}| \tag{3.28}$$

L1-regularization is known to encourage zero valued weights, which works as feature selection and can work well when large amounts of features are present. Choosing the α parameter is a heuristic process dependent on the data, but 0.0001 is suggested as a sensible initial choice [55].

Why L1 and L2 regularization is referred to as parameter penalties should be evident from the preceding explanations, but it is not clear why they are also referred to as weight decay. To see why we need to look at the derivative of the modified cost function. Taking the derivative of Equation 3.27 with respect to the weights and the biases yields

$$\frac{\partial \mathcal{C}_{L2}}{\partial w_{jk}} = \frac{\partial \sum_{i=1}^{N} \mathcal{E}(i)}{\partial w_{lk}} + \alpha w_{lk} \tag{3.29}$$

$$\frac{\partial \mathcal{C}_{L2}}{\partial b_{lk}} = \frac{\partial \sum_{i=1}^{N} \mathcal{E}(i)}{\partial b_{lk}}, \tag{3.30}$$

where b_{lk} refers to the lk^{th} element of the matrix $\mathbf{B}$ that contains the biases of all layers in the network. For L1 regularization we take the derivative of Equation 3.28 with respect to the weights and the biases, which yield

$$\frac{\partial \mathcal{C}_{L1}}{\partial w_{lk}} = \frac{\partial \sum_{i=1}^{N} \mathcal{E}(i)}{\partial w_{lk}} + \alpha \,\text{sgn}(w_{lk}) \tag{3.31}$$

$$\frac{\partial \mathcal{C}_{L1}}{\partial b_{lk}} = \frac{\partial \sum_{i=1}^{N} \mathcal{E}(i)}{\partial b_{lk}} \tag{3.32}$$

where $\text{sgn}(w)$ represents the sign function that returns -1 for negative input and 1 for positive input. From Equation 3.30 and 3.32 we can see that the derivative of the cost function with respect to the bias results in the derivative of the regular cost function with no modifications. From Equation 3.29 and 3.31 we recognize the derivative of the unmodified cost function as the first term of both equations, but we get an additional term αw for L2 regularization and $\alpha\text{sgn}(w)$ for L1 regularization, which alters the update rule from Equation 3.4 into

$$w_j^{(l)}(\text{new}) = w_j^{(l)}(\text{old}) - \mu\frac{\partial C_{L2}}{\partial w_j^{(l)}} = (1-\mu\alpha)w_j^{(l)}(\text{old}) - \mu\frac{\partial}{\partial w_j^{(l)}}\sum_{i=1}^{N}\mathcal{E}(i) \tag{3.33}$$

for L2-regularization. For L1-regularization we get

$$w_j^{(l)}(\text{new}) = w_j^{(l)}(\text{old}) - \mu\frac{\partial C_{L1}}{\partial w_j^{(l)}} = w_j^{(l)}(\text{old}) - \mu\alpha\,\text{sgn}(w_j^{(l)}(\text{old})) - \mu\frac{\partial}{\partial w}\sum_{i=1}^{N}\mathcal{E}(i). \tag{3.34}$$

From Equation 3.33 we can see that the weights will be multiplied by a factor of $(1-\mu\alpha)$ at each iteration. This factor will be slightly smaller than one, resulting in a slight decay of the weights. Similarly for Equation 3.34 we get an additional term of $\mu\alpha\,\text{sgn}(w_j^{(l)}(\text{old}))$ subtracted at each iteration, also resulting into a slight decrease in the weights. Both modifications have the effect of shrinking the weights but in a slightly different way. For L2-regularization, the shrinkage is proportional to w, while in L1-regularization the shrinkage is constant. When the magnitude of w is large, L1-regularization shrinks the weights less than L2-regularization, and opposite when the magnitude of w is small. This shrinkage explains the previous statement that L1-regularization drives the weights toward zero while retaining some high-importance connections. Choosing which form of weight decay to employ can be a process of trial and error and sometimes they are also used in conjunction.

3.2.2 Weight Initialization

Weight initialization is a procedure for restricting the network by initializing the weights closer to an ideal configuration. Careless initialization of weights and biases can result in a number of different problems. If weights are drawn randomly, and some happen to be very large, the gradients might explode. If many weights and biases end up in a range where a unit is saturated, the gradients might vanish. In DNNs infancy a common heuristic was to initialized the weights from a zero-mean Gaussian distribution with a standard deviation of 0.01 [10] or a uniform distribution in the interval $(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}})$ [58]($n = \#$ units in previous layer.), but several developers encountered convergence difficulties when training very deep models [53, 59]. It was observed that

back-propagated gradients were smaller as one moves from the output layer towards the input layer, just after initialization [60]. Analysis showed that the variance of the back-propagated gradients decreased as they flowed backward through the network [60]. Work done on weight initialization in DNNs is mainly built upon this idea of investigating the variance of the response in each layer.

For the forward pass, the response of a layer was described in Equation 3.1, but for clarity, a simplified formulation is:

$$\mathbf{z}^{(l)} = \mathbf{w}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}. \tag{3.35}$$

where $\mathbf{z}^{(l)}$ is the response vector of layer l, $\mathbf{w}^{(l)}$ is a vector comprised of all weights in layer l, $\mathbf{a}^{(l-1)}$ is a vector comprised of all activation in layer $l-1$ and $\mathbf{b}^{(l)}$ is a vector comprised of all biased in layer l. Assuming that all biases are initialized to zero, the variance of Equation 3.35 is:

$$Var[\mathbf{z}^{(l)}] = n^{(l)}Var\left[\mathbf{w}^{(l)}a^{(l-1)}\right] \tag{3.36}$$

where $n^{(l)}$ is the number of units in layer l. To proceed, several assumption are made:

- Weights are mutually independent and share the same distribution.
- Weights are drawn from a symmetric distribution with a mean of zero.
- Responses of each layer are mutually independent and share the same distribution.
- Responses of each layer have a mean of zero.
- Weights and responses are independent random variables.

With these assumptions, the variance of the product of independent variables in Equation 3.36 becomes:

$$Var[\mathbf{z}^{(l)}] = n^{(l)} Var\left[\mathbf{w}^{(l)}\mathbf{a}^{(l-1)}\right] \tag{3.37}$$

$$= n^{(l)}\Big(Var[\mathbf{w}^{(l)}]Var[\mathbf{a}^{(l-1)}] + \cancelto{0}{Var[\mathbf{w}^{(l)}]E[\mathbf{a}^{(l-1)}]^2} + \cancelto{0}{Var[\mathbf{a}^{(l-1)}]E[\mathbf{w}^{(l)}]^2}\Big) \tag{3.38}$$

$$= n^{(l)} Var[\mathbf{w}^{(l)}]Var[\mathbf{a}^{(l-1)}] \tag{3.39}$$

where the expectation of in last two terms of Equation 3.38 are zero. For a network with L layers Equation 3.39 becomes:

$$Var[\mathbf{z}^{(L)}] = Var[\mathbf{x}]\prod_{l=2}^{L} n^{(l)} Var[\mathbf{w}^{(l)}] \tag{3.40}$$

where $\mathbf{x}$ is the input vector to the network. Now follows the key idea to weight initialization; A proper initialization scheme should neither magnify or diminish the input signal exponentially. To achieve this the product from Equation 3.40 should take a proper scalar (*e.g 1*), which can be accomplished by the condition:

$$n^{(l)} Var[\mathbf{w}^{(l)}] = 1, \ \forall l. \tag{3.41}$$

For the backward pass, the gradients was described in Equation 3.12, but for clarity, a simplified formulation is:

$$\boldsymbol{\delta}^{(l)} = \boldsymbol{\delta}^{(l+1)}\mathbf{w}^{(l+1)} f'(\mathbf{z}^{(l)}). \tag{3.42}$$

Here, $\boldsymbol{\delta}^{(l)}$ is the gradient vector of layer l, $\boldsymbol{\delta}^{(l+1)}$ is the gradient vector of layer $l+1$, $\mathbf{w}^{(l+1)}$ is the weight vector of layer $l+1$ and $f'(\mathbf{z}^{(l)})$ is the derivative of the activation vector of layer l. Assuming unit derivative, i.e. $f'(\mathbf{z}^{(l)}) \approx 1$, the variance of Equation 3.42 becomes:

$$Var[\boldsymbol{\delta}^{(l)}] = n^{(l+1)} Var[\boldsymbol{\delta}_{l+1}\mathbf{w}^{(l+1)}] \tag{3.43}$$

$$= n^{(l+1)}\Big(Var[\mathbf{w}^{(l+1)}]Var[\boldsymbol{\delta}^{(l+1)}] + \cancelto{0}{Var[\mathbf{w}^{(l+1)}]E[\boldsymbol{\delta}^{(l+1)}]^2} + \cancelto{0}{Var[\boldsymbol{\delta}_{(l+1)}]E[\mathbf{w}^{(l+1)}]^2}\Big) \tag{3.44}$$

$$= n^{(l+1)} Var[\mathbf{w}^{(l+1)}]Var[\boldsymbol{\delta}^{(l+1)}]. \tag{3.45}$$

For a network with L layers Equation 3.45 becomes

$$Var[\boldsymbol{\delta}^{(2)}] = Var[\boldsymbol{\delta}^{(L)}]\prod_{l=2}^{L} n^{(l+1)} Var[\mathbf{w}^{(l+1)}] \tag{3.46}$$

Once again, the idea is that the initialization should not magnify or diminish the gradients, which can be achieved by the condition

$$n^{(l+1)} Var[\mathbf{w}^{(l+1)}] = 1, \ \forall l. \tag{3.47}$$

As a compromise between the two constrains, one approach could be

$$Var[\mathbf{w}^{(l+1)}] = \frac{2}{n^{(l)} + n^{(l+1)}}. \tag{3.48}$$

For a uniform distribution, $\mathcal{U}(-a, a)$, this amounts to choosing

$$a = \frac{\sqrt{6}}{\sqrt{n^{(l)} + n^{(l+1)}}},$$

which results in weights being drawn from the following distribution

$$\mathcal{U}\Big(-\frac{\sqrt{6}}{\sqrt{n^{(l)} + n^{(l+1)}}}, \frac{\sqrt{6}}{\sqrt{n^{(l)} + n^{(l+1)}}}\Big), \tag{3.49}$$

which is referred to as Xavier initialization [58]. This initialization scheme was derived when the most common activation functions were sigmoid and tanh. But when ReLUs

began to occur more frequently, the assumption that the response of each layer has an expectation of zero was no longer valid, since $\mathbf{z}^{(\mathbf{l})} = \mathbf{a}^{(\mathbf{l-1})} = \max(\mathbf{z}^{(\mathbf{l-1})}, 0)$. To compensate for the lost of information, [53] proposed to alter the conditions in Equation 3.41 and 3.47 by multiplying with a factor of $1/2$. For a normal distribution, this amount to drawing the weights from one of the two following distributions

$$\mathcal{N}\Big(0, \frac{\sqrt{2}}{\sqrt{n^{(l)}}}\Big) \tag{3.50}$$

$$\mathcal{N}\Big(0, \frac{\sqrt{2}}{\sqrt{n^{(l+1)}}}\Big), \tag{3.51}$$

which is referred to as HeNormal initialization [53]. Equation 3.50 ensure that the forward pass is scaled properly while Equation 3.51 scales the backward pass properly. Nevertheless, both approaches are sufficient to aid model convergence.

3.2.3 Early Stopping

For the development of DNNs it is common to split the data into the following three parts:

- **Training set**: A set of examples used to find the optimal parameters to perform the desired task.
- **Validation set**: A set of examples used for model selection and hyperparameter tuning.
- **Test set**: A set of examples used to evaluate the performance of the network.

Although we must always consider the available data, a rule of thumb is to split the complete data set such that the training set make up 50% of the data while the validation and test set make up on 25% each. It is essential that the test set is kept entirely separate and treated as unknown up until evaluation. This separation is to ensure we are testing the model on unseen data to asses if the network has found a good general description of the underlying structure of the data or overfitted on the training set.

A validation set is needed for several reasons. Firstly, since the weights and biases are initialized randomly we cannot be confident that a decrease in error on the training set was a result of an adjustment to some hyperparameter or just a particularly good initial weight configuration. By monitoring the validation set, we can see how the adjustment of hyperparameters affects data separate from the training set. Secondly, it is not guaranteed that the model corresponding to lowest error on the training set will be the model that yields the best performance on the test set, as it might be overfitted like in Figure 3.7. Instead, we monitor the validation set, stop training when the error on the validation set starts increasing and save the model corresponding to the lowest error on the validation set. This procedure is referred to as early stopping [61], and act as a regularization technique since it limits the amount of iteration and ends the training before it overfits. However, the example shown in Figure 3.7 is idealized, and when working with real data the output error might fluctuate like illustrated in Figure 3.8, which means that the error of the validation set can increase for some steps before it decreases to a new, lower value. To solve this predicament, a new hyperparameter called patience is introduced, which determines how many iterations we are willing to wait before training is stopped. If the validation error starts increasing and does not reach a lower validation error than the previous lowest validation error before the number of iterations since the previous lowest value becomes larger than the number of iteration allowed by the patience parameter we stop the training and save the model corresponding to the lowest validation error, as illustrated in Figure 3.8. Determining the value of the patience parameter is currently a process of trial and error, but there has been recent studies into more theoretical approaches to setting the patience parameter [62].

3.2.4 Dropout

One of the more recent regularization techniques is a technique referred to as Dropout [63], which has been particularly successful in DNNs [64]. Dropout is performed by randomly dropping units (along with their connections) during the training procedure. Computing the forward pass with Dropout included is performed by using the following procedure:

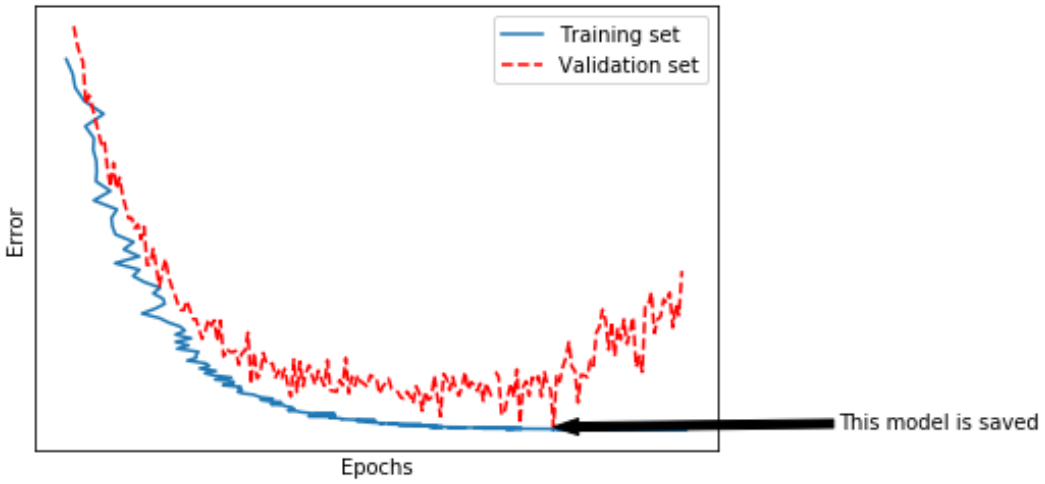


FIGURE 3.8: Error as a function of iteration steps for training and validation set.

$$
\begin{aligned}
r_k^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{a}_k^{(l-1)} &= r_k^{(l)} a_k^{(l-1)}, \\
z_j^{(l)} &= w_{jk}^{(l)} \tilde{a}_k^{(l-1)} + b_j^{(l)}, \\
a_k^{(l)} &= f(z_j^{(l)}).
\end{aligned}
$$

For each layer we sample a Bernoulli random vector $\mathbf{r}^{(l)}$, where each element of the vector has a probability p of being 1, with the same size as the activation vector $\mathbf{a}^{(l-1)}$ from the previous layer. Element-wise multiplication is performed between $\mathbf{r}^{(l)}$ and $\mathbf{a}^{(l-1)}$, which produces the thinned activation vector $\tilde{\mathbf{a}}^{(l-1)}$. Next we apply Equation 3.1 and 3.2 to obtain the output $\mathbf{a}^{(l)}$. Figure 3.9 displays a possible configuration of the network shown in Figure 3.1 when the Dropout procedure is applied. Note that nodes in the input can be dropped but nodes in the output layer is always preserved.

At each training step, a thinned network is trained and the weights that remained after Dropout are updated. For each training step, we might obtain a new, unique network that is trained and updated. Dropout can interpreted as an ensemble method where many different networks are combined. At test time we want to utilize the full capability of the network, therefore we retain all units ($p = 1$) and scale the weights such that $\mathbf{w}_{\text{test}}^{(l)} = p\mathbf{w}_{\text{train}}^{(l)}$ before the test input is presented to the network. Also note that determining the value for the hyperparameter p, the probability of retaining a unit, depend on several factors. For the input layer, it depends on the input data.

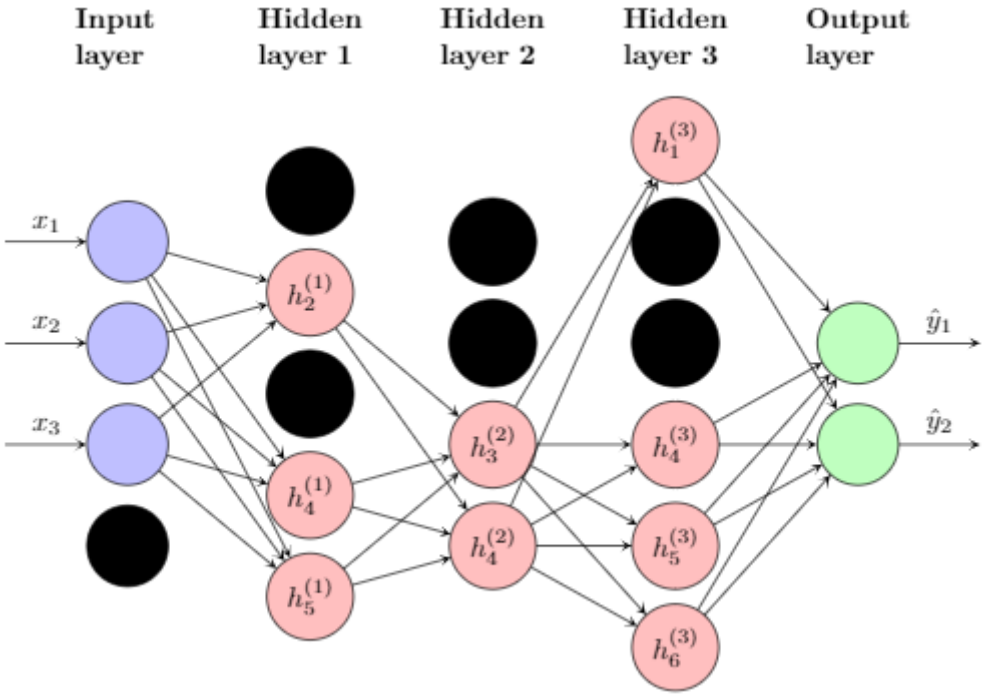


FIGURE 3.9: A possible configuration of the network from Figure 3.1 when the Dropout procedure is applied.

For real-valued data, like images or speech frames, p is usually chosen to be 0.8 [63]. This ensures that most of the information contained in the input is kept while still introducing some noise to prevent overfitting. But for very low dimensional or sparse input it might be necessary to set p equal to one, to keep the limited amount of information in the input. For the hidden layers, the choice of p is associated with the number of units in a given layer. A small value of p will result in the majority of units being dropped, so the number of units must be large. But a large number of units can make the network difficult to train that can lead to underfitting. A large value of p will result in the majority of units being kept, which might lead to overfitting. However, a common convention supported by empirical studies is to set p equal to 0.5 [63].

3.2.5 Transfer Learning

One of the key components to deep leanings success is the availability of very large datasets, like the ImageNet dataset [65], which enable training of very deep models with millions of parameters without overfitting. It was also discovered that these deep models trained on large datasets learned filters that generalized well to other data as well [66], and reusing these weights on different datasets is what we refer to as transfer

learning. But training a model on millions of images can take many days and require costly hardware. A solution is to use publicly available, pretrained networks as the basis for a new network and adjust the parameters based on the new data, a process referred to as fine tuning. Such pretrained networks can be particularly useful in cases where data is limited, or you have data that is similar to the data your network was pretrained on. However, transfer learning is not always viable. Using a pretrained network constrains the choice of architecture since the structure has to match the network where weights are imported from. Another issue might arise if the new data is too different from the original data.

3.2.6 Data Augmentation

Data augmentation is a technique to reduce overfitting by artificially increasing the amount of training data. Such methods are especially popular when training CNNs that mostly process images, since a wide range of image transformations are available. Typical transformations include cropping, rotation, zoom, and shearing. Figure 3.10 displays an example where all the transformations mentioned above are applied. Note that the transformations are applied randomly, except for the cropping that is always applied. For example, an image can be rotated within a certain range specified by the designer in advance. Since samples are presented to the network many times during training and data augmentation is performed every time an image is presented the network might see a new version at each iteration.

Data augmentation has shown to increase performance [67], but there is a limit to the effectiveness of such techniques. Since the augmented training samples are obtained from the original training data they are not statistically independent an does not have a comparable effect to gathering more real data.

3.2.7 Batch Normalization

Batch Normalization [68] is a recent technique that aims to accelerate training of DNNs by reducing internal covariate shift. Internal covariate shift refers to the fact that the distribution of each layers input changes during training, as the parameters of the previous layer changes. When a network grows deeper, these shifts can become amplified, and the learning rate must be kept small to avoid large adjustments. But

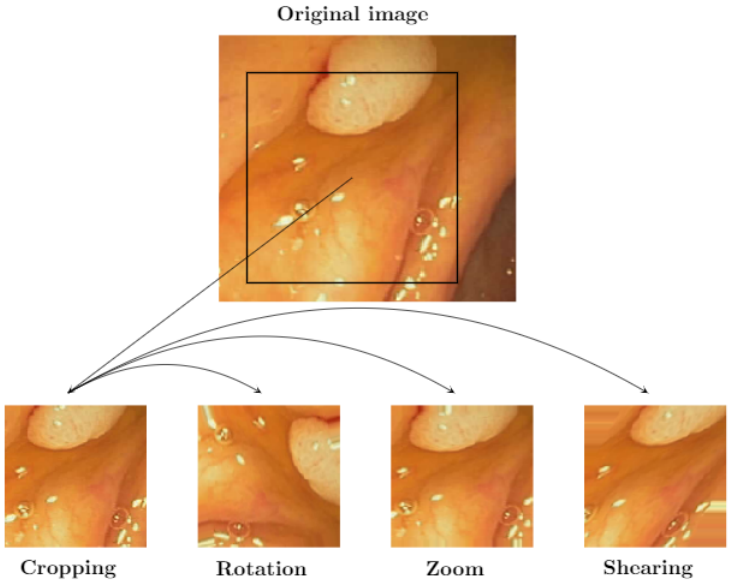


FIGURE 3.10: Illustration of data augmentation procedure with images as input. Rotation, zoom, and shearing are applied randomly a cropped region of the original image. Note that the size of the transformed images shown below the original image is not preserved in order to save space. Image obtained from [11].

a small learning rate slows down training considerably and might prevent the model from finding a good minima. Batch Normalization aims to diminish the effect of internal covariate shift by normalizing the input of each layer, such that the mean and variance is approximately 0 and 1, respectively. In doing so, the gradient's dependence on the scale of the parameters is reduced which allows a higher learning rate without divergence issues. Additionally, Batch Normalization can act as a regularizer since it restricts the activations to a certain range.

For a layer l with k_l units, the activation vector $\mathbf{a}^{(l)}$ will be normalized by

$$\widehat{\mathbf{a}}^{(l)} = \frac{\mathbf{a}^{(l)} - \mathsf{E}[\mathbf{a}^{(l)}]}{\sqrt{\mathsf{Var}[\mathbf{a}^{(l)}]}} \tag{3.52}$$

where the expectation and variance are computed over the training set. However, since networks are often trained using mini-batches (see Appendix B.1), estimates of the mean and variance are produced from each mini-batch. But by normalizing the input, information about the absolute scale of activations is discarded, which limits the networks ability to represent data. In [68], they exemplify this limitation by considering the case where inputs to a sigmoid are normalized and not scaled or shifted, which would result in the input being constrained to the linear range of the sigmoid. To

preserve information, two trainable parameters, $\boldsymbol{\gamma}^{(l)}$ and $\boldsymbol{\beta}^{(l)}$, are added to each layer l that scales and shift the normalized activation vector in the following way:

$$\widetilde{\mathbf{a}}^{(l)} = \boldsymbol{\gamma}^{(k)}\widehat{\mathbf{a}^{(l)}} + \boldsymbol{\beta}^{(k)}, \tag{3.53}$$

where $\boldsymbol{\gamma}^{(l)}$ and $\boldsymbol{\beta}^{(l)}$ are of equal size as the activation vector. These normalized activations are dependent on the mini-batch used to estimate the mean and variance. However, during inference, the output should only depend on the input. There are two approaches for estimating a mean and variance that can be utilized for model testing. One approach is to use the population mean and variance during inference. Another is to keep a running average of the mean and variance during training, which enables monitoring of the accuracy during training.

The algorithm is illustrated by focusing on an activation vector $\mathbf{a}^{(l)}$ and considering a mini-batch $\mathcal{B}$ of size m,

$$\mathcal{B} = \{\mathbf{a}_1^{(l)}, ..., \mathbf{a}_m^{(l)}\}.$$

Normalized values are denoted $\{\widehat{\mathbf{a}}_1^{(l)}, ..., \widehat{\mathbf{a}}_m^{(l)}\}$ and their linear transformations are $\{\widetilde{\mathbf{a}}_1^{(l)}, ..., \widetilde{\mathbf{a}}_m^{(l)}\}$. This transform is referred to as

$$\mathrm{BN}_{\boldsymbol{\alpha}^{(l)},\boldsymbol{\beta}^{(l)}}(x_i) : \{\mathbf{a}_1^{(l)}, ..., \mathbf{a}_m^{(l)}\} \rightarrow \{\widetilde{\mathbf{a}}_1^{(l)}, ..., \widetilde{\mathbf{a}}_m^{(l)}\},$$

and the algorithm is presented in Algorithm 1.

3.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a particular kind of DNNs designed to process data in a grid-like structure, such as images. Traditional DNNs like the MLP introduced in Section 3.1.1 consists of layers where all units in the previous layer are connected to all units in the current layer and are commonly implemented using matrix multiplication. Applying an activation function to the result of the matrix multiplication creates a layer referred to as a fully connected layer. In CNNs we replace

Algorithm 1 Batch Normalizing Transform, applied to activation $\mathbf{a}^{(l)}$ over a mini-batch.

Input: Values of $\mathbf{a}^{(l)}$ over the mini-batch $\mathcal{B}$
Parameters to be learned: $\boldsymbol{\alpha}^{(l)}, \boldsymbol{\beta}^{(l)}$
Output: $\{\widetilde{\mathbf{a}}_i^{(l)} = \mathbf{BN}_{\boldsymbol{\alpha}^{(l)},\boldsymbol{\beta}^{(l)}}(\mathbf{a}_i^{(l)})\}$

1: $\mu_{\mathcal{B}} \longleftarrow \frac{1}{m}\sum_{i=1}^{m}\mathbf{a}_i^{(l)}$ % Mini-batch mean

2: $\sigma_{\mathcal{B}}^2 \longleftarrow \frac{1}{m}\sum_{i=1}^{m}(\mathbf{a}_i^{(l)} - \mu_{\mathcal{B}})^2$ % Mini-batch variance

3: $\widehat{\mathbf{a}}_i^{(l)} \longleftarrow \frac{\mathbf{a}_i^{(l)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$ % Normalize

4: $\widetilde{\mathbf{a}}_i^{(l)} \longleftarrow \boldsymbol{\alpha}^{(l)}\widehat{\mathbf{a}}_i^{(l)} + \boldsymbol{\beta}^{(l)} \equiv \mathrm{BN}_{\boldsymbol{\alpha}^{(l)},\boldsymbol{\beta}^{(l)}}(\mathbf{a}_i^{(l)})$ % Scale and shift

this matrix multiplication with a convolution operation in one or more layers. Applying an activation function to the result of the convolution results in a convolutional layer. This section will introduce the essential components of a CNN along with some key ideas that CNNs benefit from.

3.3.1 Convolution

A convolution is an integral that expresses the overlap of two functions g and f as g is shifted over f, defined as

$$s(t) = (f * g)(t) = \int f(a)g(t-a)da \tag{3.54}$$

or in the discrete case

$$s[t] = (f * g)[t] = \sum_{a=-\infty}^{\infty} f[a]g[t-a]. \tag{3.55}$$

A typical use of the convolution operation is to filter an image I using a kernel K, often referred to as a filter, which requires computing the two-dimensional discrete convolution, given by

$$s[i,j] = (I * K)[i,j] = \sum_n \sum_m I[m,n]K[i-m, j-n]. \tag{3.56}$$

Figure 3.11 displays an example where small filters for detecting horizontal and vertical edges are applied to an image that produces two filtered images. In the context of deep learning, such filtered image are commonly referred to as the feature maps.

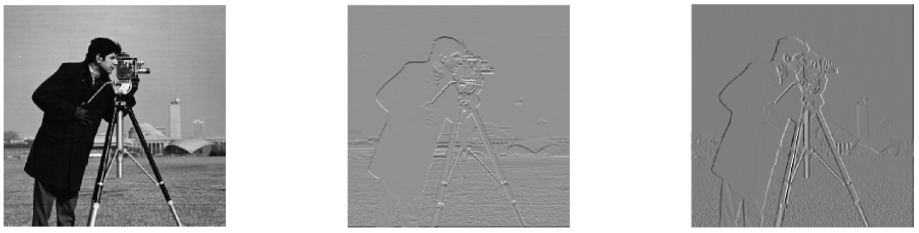


FIGURE 3.11: Example of image convolved with simple edge detector filters. From left to right: Original image, image filtered with horizontal edge detector and image filtered with vertical edge detector. Original image obtained from Scitkit-Image[1].

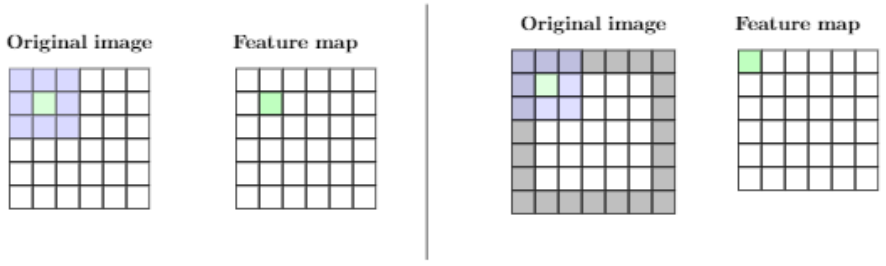
A filter is applied to an image by considering a small neighbourhood and calculating the weighted sum of the pixel values contained in this neighbourhood, where the weights are dependent on the choice of kernel. This filter is initially placed in the upper-left corner where the first weighted sum is calculated, where the resulting value corresponds to the upper-left pixel in the feature map. This filter is then shifted to the right and the process is repeated. For the example shown in Figure 3.11 the following kernels, known as Sobel filters [69], were applied:

$$\mathbf{K}_h = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \qquad \mathbf{K}_v = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix},$$

where $\mathbf{K}_h$ detects horizontal edges and $\mathbf{K}_v$ detects vertical edges. In this example, the feature maps end up with the same size as the original image, but that is not necessarily the case. There are three factors that affect the size of the feature map, namely kernel size, stride, and zero-padding. Kernel size refers to the size of the considered neighbourhood and is generally chosen as a square matrix of size (d, d), where d is usually an odd number. Choosing d as an odd is to ensure that the filter has a center, which will correspond to the pixel at the same position in the feature

[1] http://scikit-image.org/

map. However, if the filter is placed in the upper-left corner to obtain a value for the upper-left corner pixel it results in boundary issues since there are no pixels on its left or above it. Such a scenario is shown on the left of Figure 3.12, where a 3×3 kernel is applied to a 6×6 image, resulting in a feature map with both its height and width reduced by two pixels. Solving this predicament is commonly done by padding the border around the image with zeros, which does not affect the weighted sum but ensure that the resolution is preserved. Displayed on the right of Figure 3.12 is an example where a 3×3 kernel is applied to a 6×6 image, but with padding included, resulting in a feature map of equal size as the original image. Also, note that a larger kernel would require more zero-padding to keep the resolution of the input. Lastly, one must also consider the number of pixels the filter is shifted, referred to as stride. In the example shown in Figure 3.12, the resulting feature map would only have the same size if the filter was shifted one pixel at the time. If the filter was shifted two pixels after computing a weighted sum, it would result in a feature map with half the size of the original image.



(A) Convolution without padding. (B) Convolution with padding.

FIGURE 3.12: Illustration of convolution operation with and without zero-padding.

Taking all these factors into account, the size of a feature map can be calculated using the following equation:

$$n_{out} = \frac{n_{in} + 2p - k}{s} + 1, \tag{3.57}$$

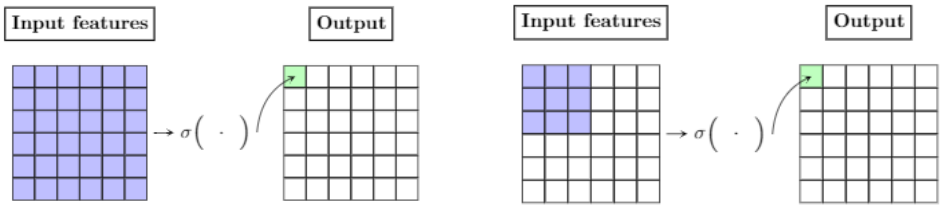
where n_{out} is the number of output features, n_{in} is the number of input features, p is the amount of padding, s is the stride, and k is the size of the kernel. Determining the kernel size, zero-padding and stride is an important part of network design that is dependent on several details. A large stride might be desirable to reduce the resolution and ease the computational burden when large images are concerned. Small images,

on the other hand, might require zero-padding to keep the images from becoming too small to process. As for kernel size, small kernels like 3×3 kernels are common since they contain few parameters that allow more filters and deeper network. However, large kernels have also seen use, particularly in early layers to reduce the size of the feature maps [10].

3.3.2 Motivation

From the outset, it might not be obvious how convolution improves DNNs, but it is a crucial component of networks designed for computer vision task. How convolution improve DNNs is based on two central ideas. Firstly, convolution exploits the idea that, particularly in images, local groups of values are often highly correlated and form distinct patterns, detectable by small filters that consider local neighbourhoods. In a fully connected layer, all pixels in the input image are connected to a single unit, which produces the value for a single pixel. A convolutional layer only considers a small neighbourhood around the pixel in question, referred to as sparse connectivity. Figure 3.13 illustrates the difference between the two approaches when applied to a small 6×6 image. In this example, a fully connected layer would require 36 parameters to produce a single pixel value. Using convolution and considering a 3×3 neighbourhood, only 9 parameters would be required to produce the same value. Secondly, local regions in structured data (such as images) tend to be invariant to location, that is, an edge is an edge regardless of where in the image it appears. As seen in Figure 3.11, a small filter can be used to process an entire image. Returning to the example in Figure 3.13, a fully connected layer would require 36 additional parameters for each new pixel value. If a picture of equal size was to be produced, this would result in $36 * 36 = 1296$ parameters. A convolutional layer would use the same 9 parameters for the entire image, an idea referred to as parameter sharing, thus reducing the number of parameters needed significantly. Also, notice that a fully connected layer require the number of inputs to be known, such that the number of weights can be specified. This limits a fully connected layer to only processing images of equal size, while a convolutional layer can tackle input of arbitrary size.

Convolution allows for deeper with fewer parameters, thus reducing the potential for overfitting. Also, studies have shown that as a network grows deeper its performance increases [53, 59]. The increase in performance is often explained by considering how the data is transformed when processed by the network. As previously stated,



(A) Matrix multiplication. (B) Convolution.

FIGURE 3.13: Illustration of the difference between processing a 6×6 image using matrix multiplication and convolution. For matrix multiplication, shown on the left, 36 parameters are required to produce the value for the top-left pixel of the feature map. If a feature map of equal size was desired it would require $36 * 36 = 1296$ parameters. For convolution, showed on the right, only 9 parameters are required to produce the value for the top-left pixel of the feature map. These 9 parameters would also be used to process the remaining pixel values, thus dramatically reducing the number of parameters needed to produce a feature map of equal size as the input.

the network tries to transform the data into a representation where discrimination is optimal. Early layers of deep CNNs tend to learn simple filters that extract general features such as edges [66]. Succeeding layers build upon these general features to create a more complicated representation, obtaining what is known as a distributed representation. However, this increased depth also brings challenges concerning model training as a result of the vanishing gradient problem discussed earlier in this chapter. To harness the true benefits of convolution it should be complemented by recent innovations such as ReLUs and Batch Normalization.

3.3.3 Pooling

Another component that is often included in CNNs is pooling. A pooling function replaces a region of a feature map with a summary statistic of said region. A common choice is the max pooling function, which returns the maximum value within the region. Other options are available, for instance, taking the average value of the region, but max pooling is by far the most used pooling function in the context of CNNs. Because the pooling operation is applied individually on each feature map the total number of feature maps will remain the same before and after the pooling operation. Figure 3.14 illustrates the max pooling operation when applied to an image of size 4×4. The image is separated into non-overlapping neighbourhoods of size 2×2, a stride equal to 2 and each region is replaced by the maximum value of each neighbourhood, thus reducing the size of the original image by 2.

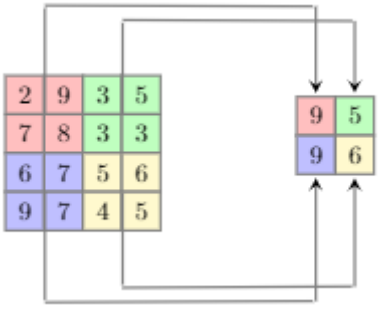


FIGURE 3.14: Illustration of a 2×2 max-pooling operation with stride $= 2$ applied to a 4×4 grid resulting in a new image of size 2×2.

Pooling is motivated by a desire to make the representation approximately invariant to small translations in the input. If a feature is present in the input image, we want most of the pooled outputs to remain the same even if the feature is shifted slightly. Pooling also has the effect of reducing the size of the feature maps, which reduces the computational burden. However, in some cases, we might be concerned about small shifts in the input which means that the amount of pooling must be considered in the context of the problem at hand.

3.3.4 Architecture

As CNNs have been applied to an increasing amount of tasks so has the number of different architectures also increased. Modern CNNs can consist of hundreds of layers [70] with different tweaks and adjustments included to enhance performance. With that in mind, we consider a simple network inspired by one of the first successful CNNs to illustrate the structure of a typical CNN, namely the LeNet-5 [41], displayed in Figure 3.15. LeNet-5 was originally used to classify grey-scaled, 28×28 images of hand-written digits from the MMNIST dataset[2], which is why the output layer is shown to have ten output nodes, one for each digit. Images are commonly presented to the network as a multidimensional-array on the form (N, C, H, W), where N refers to the number of images, C refers to the number of channels in an image, H is the height of an image and W is the width of an image. For the MNIST dataset, N can be chosen by the designer, C is equal to 1, and both H and W is equal to 28.

The first layer of the network shown in Figure 3.15 is a convolutional layer, composed of 6, 5×5 filters with a stride of 1 and no zero-padding. After an activation function is applied to all pixels in the resulting feature maps, the output of the first convolutional

layer is a multidimensional-array on the form $(N, 6, 26, 26)$. Following the first layer is a pooling layer, where a 2×2 max-pooling operation with stride equal to 2 is applied to each feature map, which results in the feature maps from the second layer to have the form $(N, 6, 12, 12)$. Next follows another convolutional layer, consisting of 16, 5×5 filters with a stride of 1 and no zero-padding that, after an activation function, results in feature maps on the form $(N, 16, 8, 8)$. Succeeding the third layer is another pooling layer that, again, applies a 2×2 max-pooling operation with a stride equal to 2, resulting in feature maps on the form $(N, 16, 4, 4)$. The next layer is a fully connected layer consisting of 84 units, which requires some modification of the feature maps. Since fully connected layers process data through matrix multiplication, the feature maps are converted from the form $(N, 16, 4, 4)$ to the form $(N, 16 * 4 * 4)$. After matrix multiplication and an activation function the resulting features are on the form $(N, 84)$. Finally, the last layer of the network is a fully connected layer consisting of 10 units, one for each digit, where the output of these units is passed through a softmax function. Computing the cost is done with the MSE cost function, and training is performed using the backpropagation algorithm and gradient descent.

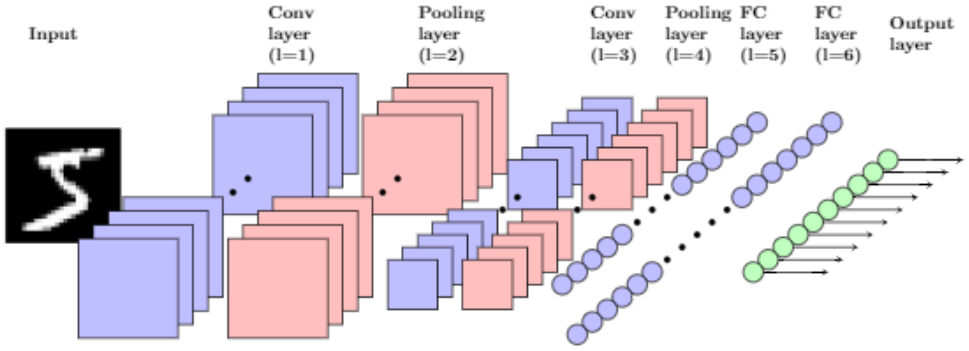


FIGURE 3.15: Architecture inspired by the LeNet-5 [41]. Each convolutional layer performs convolution with the input and applies an activation function. Each pooling layer performs max-pooling using a 2×2 kernel with stride equal to 2. Fully connected layers consists of matrix multiplication followed by an activation function. At the end of the network, a softmax function is applied.

From the preceding description one should have a general idea of how a CNN works, but some aspect of the network might still seem arbitrary. Why is the number of filters and units chosen as they are? Why are there two convolutional layers and three or more? Should convolutional layers always be followed by pooling layers and why do these pooling layers always reduce the resolution by a factor of 2? At this point, DNNs lack the theoretical framework to answer such questions accurately, and

[2] http://yann.lecun.com/exdb/mnist/

network architectures are usually chosen empirically by evaluating networks on different datasets.

3.4 Fully Convolutional Networks

Early CNNs achieved impressive results on both object classification and detection tasks, but there was no obvious way they could perform tasks where each pixel had a corresponding label, such as semantic segmentation. As a result of the pooling layers commonly applied in CNNs, the resolution of the feature maps is gradually decreased throughout the network. The decrease in resolution leave the feature maps with fewer pixels than the labeled image, making per-pixel predictions difficult. However, a recent extension to CNNs referred to as Fully Convolutional Networks (FCNs) are particularly suited to tackle per-pixel prediction problems [9]. FCNs employ an encoder-decoder architecture and are capable of end-to-end learning. The encoder network consists of one or more encoders that extract useful features from an image and maps it to a low-resolution representation. The decoder network consists of one or more decoders that are tasked with mapping the low resolution representation back into the same resolution as the input image. This section will look closer at the different components needed to construct FCNs.

3.4.1 Encoder Network and Decoder network

Similar to ordinary CNNs, the encoder network of FCNs is tasked with extracting useful features from the input and mapping it to a low-resolution representation. An encoder network consists of one or more encoders, where each encoder is comprised of one or more convolutional layers. Feature maps within a single encoder are usually zero padded such that the resolution remains constant within the encoder, but an encoder is commonly followed by a pooling layer that reduced the resolution of the feature maps. Only decreasing the resolution in the pooling layers provides clarity as to how much the resolution has been decreased throughout the encoder, which is helpful when recovering the original resolution.

To enable per-pixel prediction the low-level representation provided by the encoder network must be mapped into the same resolution as the original image. This task

is performed by the decoder network, which consists of one or more decoders. Each decoder is comprised of one or more convolutional layers and an upsampling layer at the end of the decoder. This upsampling layer is tasked with increasing the resolution of the feature maps, which can be achieved in several ways, some of which are presented in the following subsection.

3.4.1.1 Upsampling

Increasing the resolution of an image requires interpolating values for pixels that are not in the original image, based on pixels in the original image. One widely used method is nearest-neighbour interpolation, where a new pixel is assigned the same value as the nearest point. Nearest-neighbour's strength lies in its simplicity, but it is known to produce pixelated images when used for upsampling of image. Another popular method is bilinear interpolation, where a new pixel is assigned a value based on a weighted average of nearby pixels. For instance, given four pixels with values $(x_1,\ y_1)$, $(x_1,\ y_2)$, $(x_2,\ y_1)$ and $(x_2,\ y_2)$ and a new pixel with unknown values $(x,\ y)$, bilinear interpolation would determine its value by calculating

$$(x,\ y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} (x_1,\ y_1) & (x_1,\ y_2) \\ (x_2,\ y_1) & (x_2,\ y_1 2 \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}.$$

In contrast to nearest-neighbour interpolation, bilinear interpolation can create new values for pixels and therefore generate a smoother looking image after upsampling. However, it does come at the cost of performing a number of calculations, which can be demanding for large image data. A third option is known as transposed convolution, an approach that is often employed in FCNs [9, 71, 72]. Transposed convolution performs ordinary convolution, but by controlling the kernel size, padding and stride of the operation we can increase the resolution of the image. A new pixel is therefore assigned a value based on the weighted sum of nearby points and the weights of the kernel. Compared to nearest-neighbour and bilinear interpolation, which is constant, transposed convolution has the advantage that it can learn the weights for the upsampling procedure, thus providing greater flexibility. Nevertheless, this introduces more parameters to the network, which might lead to overfitting. Empirical evaluation of the

different methods has shown that using transposed convolution for upsampling in FCN improve performance and is generally the approach employed in most FCNs [9, 72].

3.4.2 Architecture

As with CNNs, a variety of FCNs have been developed with different modifications and adjustments. To illustrate the general structure of FCNs we consider one of the first and most basic networks, namely the FCN-32, displayed in Figure 3.16. The encoder network is composed of five encoders, each followed by a pooling layer. Each encoder applies convolution followed by an activation function, where the two initial encoders repeat convolution followed by activation function twice, and the three succeeding encoders apply convolution followed by activation function three times. Each pooling layer applies a max-pooling function with a 2×2 neighbourhood with a stride equal to two, thus reducing the resolution by a factor of two for each encoder, and by a factor of 32 in total.

The decoder network consists of only a single decoder, made up of two convolutional layers and followed by an upsampling layer. Upsampling is performed using transposed convolution, where the resolution is increased by a factor of 32, hence the name FCN-32. These feature maps are then passed into a softmax function to obtain the final prediction of the network.

3.5 Uncertainty and Interpretability in DNNs

Despite DNNs success on a large variety of computer vision tasks they are not without flaws. Most deep models are unable to represent the uncertainty associated with their predictions and they give no indication as to which features are affecting their decisions. These limitations have not stopped deep learning from being the tool of choice for tasks like facial recognition and machine translation, but such impediments are more problematic if deep learning aims to make a difference in the medical field. A physician will be reluctant to make a diagnosis based on a singular prediction with no notion of uncertainty or indication of which features the prediction is based on. This section introduces several recent methods that address exactly these issues.

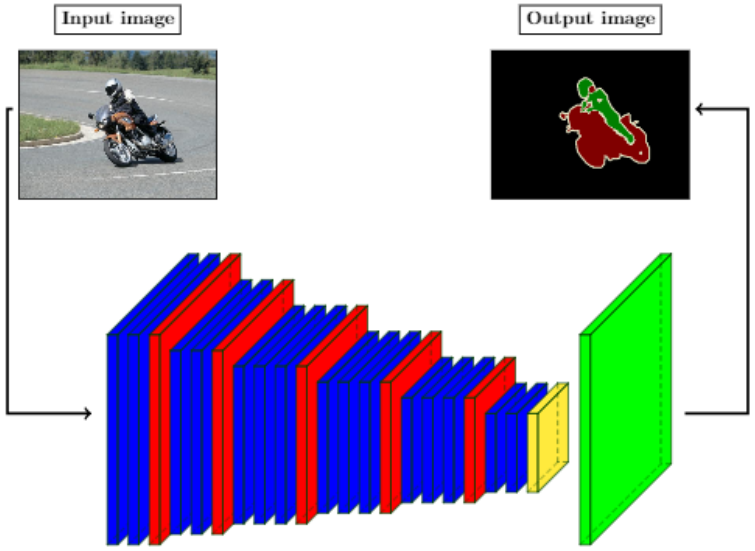


FIGURE 3.16: Illustration of the Fully Convolutional Network-32 architecture from [9]. Images are obtained from [73]. Color code description:
Red: Pooling.
Green: Soft-max.
Yellow: Upsampling.
Blue: Convolution, Batch Normalization and ReLU.

3.5.1 Uncertainty Estimation

Uncertainty modeling is a crucial component of any model, deep or not. Determining how confident a model is can bolster trust in high certainty cases or allow experts to asses special cases with high uncertainty. For example, when segmenting colorectal polyps, a model might output some uncertainty measure to accompany its prediction, which physicians could use to judge if a case requires further investigation before making a diagnosis. For network designers, determine which cases result in high uncertainty predictions can provide valuable information about the model. For instance, if a model classifies an object correctly but fails after the object is rotated, one might include data augmentation to artificially inflate the number of rotated examples in the training set. Or if the model consistently struggles with one particular kind of cases it would indicate which kind of data is lacking in the training set.

Unfortunately, deep learning based models do not have any inherent notion of uncertainty. Although the softmax output at the end of the network is often treated as model confidence this is generally ill-advised [74]. A simple example that illustrates the limitations of treating the softmax output as model confidence is to consider a network trained to classify an image as containing a cat or a dog. If this network is

given an image showing a car, the softmax output might still indicate high confidence for one of the classes when it should have produced a low probability for both classes or indicate a high degree of uncertainty. However, a recently proposed framework offers a simple approach to the problem of uncertainty modeling inspired by Bayesian probability theory [74, 75].

Bayesian models are accompanied by an intrinsic notion of uncertainty provided by the mathematical framework that Bayesian probability theory is built upon. To evaluate the probability of a hypothesis, such as the value of a parameter, a Bayesian approach would be to assign a prior probability for the hypothesis and then update it to a posterior probability as new data is presented. Updating the posterior is done using Bayes' rule that can be expressed as

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}, \tag{3.58}$$

where H represents the hypothesis and D represents the data. The term $P(D|H)$ in Equation 3.58 is known as the likelihood function and expresses the probability of the observed data, given that the hypothesis is true. A likelihood function is chosen by assuming some model based on knowledge about the data. The term $P(H)$ is referred to as the prior probability and describe our prior knowledge about the data. The term $P(D)$ is known as the marginal probability of the data, a term that is often omitted, in that case Bayes' rule takes the following form

$$P(H|D) \propto P(D|H)P(H). \tag{3.59}$$

The marginal distribution of the data is something we are given thus it does not depend on the hypothesis we wish to investigate and only acts as a normalizing constant, which is why it is often excluded. Lastly, the left side of Equation 3.58 is called the posterior distribution and, as mentioned, indicates the probability of the hypothesis after the data has been examined.

To see how a Bayesian approach would provide a notion of uncertainty to neural networks, we first revisit how a typical neural network solves a task. Given a dataset $\mathbf{X} = \{\mathbf{x_1}, ..., \mathbf{x_N}\}$ and the corresponding labels $\mathbf{Y} = \{\mathbf{y_1}, ..., \mathbf{y_N}\}$, the task is to find a function $f : X \rightarrow Y$ using some parameters $\boldsymbol{\theta}$. Finding this function would be

done using the procedure we have outlined throughout this chapter, by designing a neural network and finding the parameters of the network using backpropagation and gradient descent. Using the trained network on a new input vector, $\mathbf{x}^*$, to predict the label vector, $\mathbf{y}^*$, would amount to a forward pass through the network $\hat{\mathbf{y}}^* = f(\mathbf{x}^*; \theta)$ using the parameters found during training of the network. But using a fixed estimate for $\boldsymbol{\theta}$ ignores the uncertainty associated with the parameters, which could result in a function that produces more extreme predictions than is probable. For a Bayesian neural network, the predictive distribution is given by

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}) d\boldsymbol{\theta}. \tag{3.60}$$

The first term of Equation 3.60, $p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\theta})$, is just the output of the network after applying the softmax function, that is $p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\theta}) = \text{Softmax}(f(\mathbf{x}^*; \boldsymbol{\theta}))$. The second term, $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y})d\boldsymbol{\theta}$, is the posterior of the parameters and can be written as

$$p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}) \propto p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) \tag{3.61}$$

We recognize $p(\boldsymbol{\theta})$ as the prior distribution of the model parameters and by revisiting our assumptions about the distribution of the parameters from Section3.2.2 it is natural to assume a Gaussian prior distribution, $p(\boldsymbol{\theta}) \sim \mathcal{N}(0, \mathbf{I})$. However, there is no natural choice for the likelihood function, which means that the integral in Equation 3.60 must be evaluated numerically through Monte Carlo integration. But, evaluating the posterior to find plausible model parameters is computationally intractable, so instead we replace the posterior with an approximate variational distribution $q(\boldsymbol{\theta})$ that is simple to evaluate. Using $q(\boldsymbol{\theta})$ we obtain a new, approximate predictive distribution given by

$$q(\mathbf{y}^*|\mathbf{x}^*) = \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\theta}) q(\boldsymbol{\theta}) d\boldsymbol{\theta}. \tag{3.62}$$

Nevertheless, this begs the question, how to choose $q(\boldsymbol{\theta})$? The idea put forth in [74] was to utilize the Dropout procedure presented in Section 3.2.4 in order to sample from $q(\boldsymbol{\theta})$. Assuming that $\boldsymbol{\theta} = \{\mathbf{W}_i\}_{i=1}^L$, i.e. the weights of a neural network with L layers (biases could be incorporated but omitted for clear notation), we wish to sample plausible weights from the approximate variational distribution. Recalling the Dropout procedure, we sample a set of vectors, $\{\mathbf{r}_i\}_{i=1}^L \sim \text{Bernoulli}(p)$, each with similar size

as the weights of the corresponding layer. By taking the Hadamard product, denoted by $\circ$ between the weights of each layer and their corresponding Bernoulli vector we obtained a new set of weights

$$\{\widehat{\mathbf{W}}_i\}_{i=1}^{L} = \{\mathbf{W}_i \circ \mathbf{r}_i\}_{i=1}^{L}. \tag{3.63}$$

By sampling T sets of $\{\mathbf{r}_i\}_{i=1}^{L}$ we can obtain T sets of sampled weights that can be used in the approximate predictive distribution from Equation 3.60 by

$$q(\mathbf{y}^*|\mathbf{x}^*) = \frac{1}{T}\sum_{t=1}^{T} \text{Softmax}(f(\mathbf{x}^*; \widehat{\mathbf{W}}_t)) \tag{3.64}$$

Practically, Equation 3.64 amounts to performing T forward passes and gathering the results, which in turn can be used to estimate mean and uncertainty of the prediction. The authors [74] referred to this method for approximating samples from the predictive distribution as Monte Carlo Dropout.

We utilize Monte Carlo Dropout to estimate the uncertainty in FCNs polyp predictions, producing novel uncertainty maps in the context of semantic segmentation of colorectal polyps. In Section 5.3 of Chapter 5 we present the results of this uncertainty estimation.

3.5.2 Interpretability

Understanding what influences the prediction of a model is not only crucial for building trustworthiness, it can also be helpful for analyzing the shortcomings of a network. For instance, if we notice that a model can detect an object at some position but fails if the object is rotated, we might be encouraged to perform data augmentation to artificially inflate the train set with rotated examples. Interpretability, or rather the lack of it, has been one of the main criticisms directed at DNNs and they have often been accused of being "black boxes" [76], capable of high performance but with no possibility to understand its inner workings. Such criticism is certainly justified to some degree, but recent works have started addressing the lack of interpretability in DNNs. One of the first approaches used a deconvolutional network to visualize what features the network deemed important for a particular prediction [66] . Aided by this new

technique they won the ILSVRC[3] in 2013 and opened the door for more interpretable DNNs. Nevertheless, deconvolutional networks are known to produce visualizations that can, in some cases, be difficult to interpret and can be complicated to construct.

Therefore, this thesis will concentrate on a different approach that utilizes the gradients of the network prediction with respect to the input feature map to visualize what the network considers important; a technique first explored in the context of DNNs in [77]. One can think of this as a way of examining which pixels need to be changed the least to affect the prediction the most since those pixels should have the greatest impact on the prediction. Gradient-based visualization techniques are less complicated and can, with certain modifications, produce distinct visualizations of important features for the model. As a motivational example[4], consider a linear score model for the class c:

$$S_c(\mathbf{I}) = \mathbf{w}_c^T \mathbf{I} + b_c \ , \tag{3.65}$$

where S_c is a score function for class c, $\mathbf{I}$ is an image represented on vector-form(one-dimensional), $\mathbf{w}_c$ is the weight vector of the model and b_c is the bias of the model, respectively. By inspecting Equation 3.65 it is possible to see that the magnitude of different elements of $\mathbf{w}_c$ affect how important the model considers a pixel of the image $\mathbf{I}$ for the class c. If each pixel is evaluated an image visualizing the importance of each pixel to that class can be constructed, called a class saliency map. However, in the context of DNNs, the score function S_c would be the output of the network, which is a highly non-linear function of I, making such interpretations very difficult. But $S_c(\mathbf{I})$ can be approximated with a linear function in the neighbourhood of an image $\mathbf{I}_0$ by computing the first-order Taylor expansion:

$$S_c(\mathbf{I}) \approx \mathbf{w}^T \mathbf{I} + b, \tag{3.66}$$

where $\mathbf{w}$ is the derivative of S_c with respect to the image $\mathbf{I}$ at the point $\mathbf{I}_0$:

$$\mathbf{w} = \frac{\partial S_c}{\partial \mathbf{I}}\Big|_{\mathbf{I}_0}. \tag{3.67}$$

[3] http://www.image-net.org/challenges/LSVRC/
[4] Inspired by Section 3 of [77].

From Equation 3.67 one can obtain the class score derivatives for the given image, and one can interpret the magnitude of these derivatives as indicators of each input pixels importance. Obtaining the derivatives from Equation 3.67 is done using the backpropagation algorithm. Constructing saliency maps is done by presenting an image the trained network that produces a score. Note that, the softmax function is not applied when these maps are computed. This is to preserve the relative magnitudes of the class scores, which will give more a distinct indication as to what pixels are important. Next, the gradients from Equation 3.67 are computed using the backpropgation algorithm. If the gradients are propagated all the way back to the beginning of the network we end up with similarly shaped gradients as the original image, such that the magnitude of each element of the gradients indicate the importance of the corresponding element in the input image. Images are often presented in RGB form, so the gradients will also have three channels. To obtain a single number for a pixels importance we take the maximum across the color channels, resulting in an image with the same height and width as the original image but with only a single channel.

Figure 3.17 display an example where the procedure just described has been applied to a CNN tasked with classification. In this specific example, the network is presented with an image containing a dog, shown at the top of Figure 3.17, and computes a score for the given image. From this score vector we extract the score for the class in question, in this case, the dog class that is used to compute the gradients. The image displayed at the bottom of Figure 3.17 indicates what pixels are deemed important by the network to assign this image to the dog class. It shows of background pixels are irrelevant to the prediction while pixels associated with the dog is considered important. Accompanying network predictions with visualization of discriminative features increase model interpretability and bolster the trustworthiness of deep networks. Furthermore, saliency maps can be consulted in poor prediction cases to asses what features caused the confusion.

3.5.2.1 Guided Backpropagation

Saliency maps are easy to compute and give insight into the networks inner workings, but there are difficulties associated with the approach. Consider the following feature map from layer $l-1$ of some network

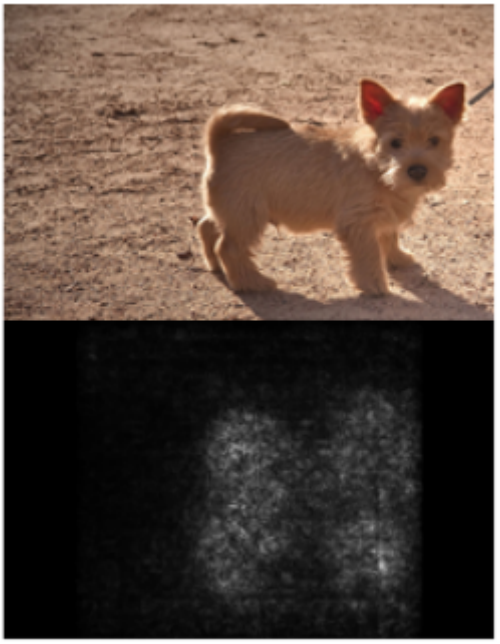


FIGURE 3.17: Example obtained from [77] that illustrates what pixels a network deems important. Top image display an image belonging to the dog class and the bottom image is saliency map constructed by propagating the gradients associated with the top image backward through the network.

$$f^{(l-1)} = \begin{bmatrix} 1 & -1 & 5 \\ 2 & -5 & -7 \\ -3 & 2 & 4 \end{bmatrix}. \tag{3.68}$$

Assuming ReLU will produce the following activation in layer $l-1$

$$\max\left(f^{(l-1)}, 0\right) = \begin{bmatrix} 1 & 0 & 5 \\ 2 & 0 & 0 \\ 0 & 2 & 4 \end{bmatrix}, \tag{3.69}$$

and the following derivative for layer $l-1$

$$\max\left(f^{(l)}, 0\right)' = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}. \tag{3.70}$$

Next, assume that the error of the succeeding layer l is found to be

$$e^{(l)} = \begin{bmatrix} -2 & 3 & -1 \\ 6 & -3 & 1 \\ 2 & -1 & 3 \end{bmatrix}. \tag{3.71}$$

Combining the results from Equation 3.70 and 3.71 with Equation 3.12

$$\delta^{(l-1)} = \max\left(f^{(l-1)}, 0\right)' \circ e^{(l)} = \begin{bmatrix} -2 & 0 & -1 \\ 6 & 0 & 0 \\ 0 & -1 & 3 \end{bmatrix}, \tag{3.72}$$

where $\circ$ denotes the Hadamard product. Equation 3.72 shows that both negative and positive gradients are propagated backward through the network, where positive gradients are associated with discriminative features and negative gradients are associated with features that should be suppressed. Both negative and positive gradients are important during the optimization procedure but not necessarily during network analysis. We are concerned about the features that the network deems important, which means that negative gradients might contribute to noisier visualizations. A recent technique that address the problem of noisy visualizations is Guided Backpropagation [78], which propose to impute the gradient such that only positive gradients flow backward through the network. Consider Equation 3.72 again, but with negative gradients zeroed out:

$$\delta^{(l-1)} = \max\left(f^{(l-1)}, 0\right)' \circ e^{(l)} \circ (e^{(l)} > 0) = \begin{bmatrix} 0 & 0 & 0 \\ 6 & 0 & 0 \\ 0 & 0 & 3 \end{bmatrix}, \tag{3.73}$$

Comparing Equation 3.72 and 3.73 it is straightforward to see that fewer values are propagated backward through the network, which should provide more distinct visualizations of discriminative features. Implementation of Guided Backpropagation follows the same procedure as for saliency maps, but the backward pass of the trained network is modified such that negative gradients are canceled out. To provide further comparison we present an example from [78], shown in Figure 3.18 that exemplify the difference between deconvolutional networks, saliency maps, and Guided Backpropagation. Notice how both the results produced through a deconvolutional network and

the saliency map highlight a large number of features while Guided Backpropagation single out some important features located in the center of the image.

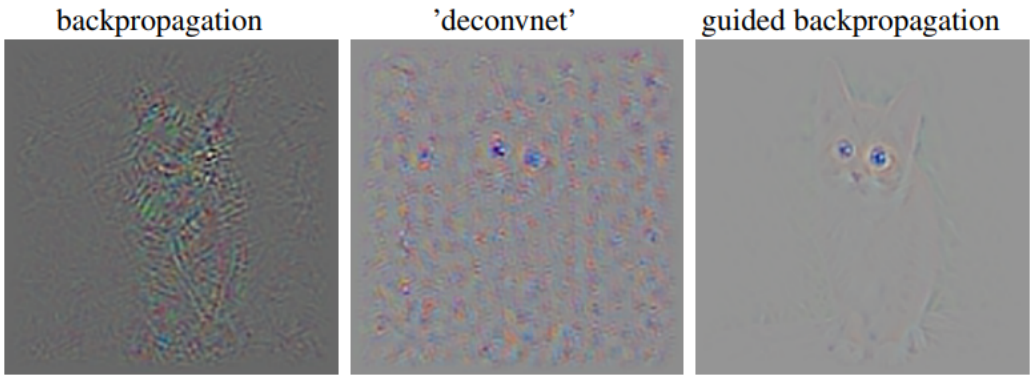


FIGURE 3.18: Example obtained from [78], which illustrate the difference between saliency maps (backpropagation), deconvolutional networks (deconvnet) and Guided Backpropagation. From left to right

We utilize Guided Backpropagation to determine which features in the input motivate a FCN to produce a particular prediction, resulting in novel interpretability maps in the context of semantic segmentation of colorectal polyp. In Section 5.4 of Chapter 5 we present the result of this Guided Backpropagation procedure.

Chapter 4

Innovations and Network Details

Chapter 3 provided the theoretical foundation required to grasp deep models and several techniques associated with increasing our understanding of such models. In this chapter, we propose several novel methods that aim to improve our understanding even further. Additionally, we give a detailed description of the models employed in this thesis and outline the modifications we propose in order to improve these models.

4.1 A Proposed Method for Estimating Gradient Uncertainty

In Section 3.5.1 of Chapter 3 we discussed DNNs inability to produce any notion of uncertainty and described Monte Carlo Dropout that provides a method to obtain approximate measures of uncertainty for DNNs by utilizing dropout during inference. Accompanying a model's prediction with an uncertainty estimate add options to assess if a particular prediction is highly certain or a case that could require further analysis from a human expert. In Section 3.5.2 of Chapter 3 we described Guided Backpropagation, a technique developed to visualize the relative importance of input features for CNNs by considering the positive gradients from a backward pass through the network. But, determining the importance of the input features based on gradients from a single backward pass runs into the same problems we discussed regarding decisions based on predictions from a single forward pass. How confident are we that these features are important for the decision of the network?

To determine the uncertainty associated with input feature's importance we propose a novel approach inspired by Monte Carlo Dropout combined with Guided Backpropagation. Given a new sample $\mathbf{x}^*$, we want to find the gradients that correspond to the input features, denoted by $\boldsymbol{\delta}^0$. Taking a similar approach as in Section 3.5.1 of Chapter 3, the approximate predictive distribution for the gradients of the input features is given by

$$q(\boldsymbol{\delta}^0|\mathbf{x}^*) = \int p(\boldsymbol{\delta}^0|\mathbf{x}^*, \boldsymbol{\theta})q(\boldsymbol{\theta})d\boldsymbol{\theta}. \tag{4.1}$$

Calculating $p(\boldsymbol{\delta}^0|\mathbf{x}^*, \boldsymbol{\theta})$ is done through the backpropagation algorithm described in Section 3.1.2 of Chapter 3, i.e. computing the gradients with respect to the output of the network and then using the chain rule to work backward toward the input gradients. Also, we want to modify the backward pass such that negative gradients are canceled, following the Guided Backpropagation procedure. For clear notation, we denoted this procedure as $\nabla_{\boldsymbol{\theta}} f_{gb}(\mathbf{x}^*; \boldsymbol{\theta})$, where $\nabla_{\boldsymbol{\theta}}$ indicated finding the gradients of each layer with respect to the parameters of the network and $f_{gb}(\mathbf{x}^*; \boldsymbol{\theta})$ is the prediction of the model with the modified backward pass. Again, we assume that $\boldsymbol{\theta} = \{\mathbf{W}\}_{i=l}^{L}$ is the set of weights of a neural network with L layers, sample a set of vectors $\{\mathbf{r}_i\}_{i=1}^{L} \sim \text{Bernoulli}(p)$ and take the Hadamard product $\{\mathbf{W} \circ \mathbf{r}_i\}_{i=l}^{L}$ to obtain a sampled set of weights from the network. Sampling T sets of weights give the following approximate predictive distribution for the gradients of the input features

$$q(\boldsymbol{\delta}^0|\mathbf{x}^*) = \frac{1}{T}\sum_{t=1}^{T} \nabla f_{gb}(\mathbf{x}^*; \widehat{\mathbf{W}}_t). \tag{4.2}$$

In practice, this amount to performing T forward and backward passes with dropout applied and storing the gradients, a method we refer to as Monte Carlo Gradients.

We utilize Monte Carlo Gradients to estimate the uncertainty of which features in the input motivate a FCN to produce a particular prediction, resulting in novel gradient uncertainty maps. In Section 5.5 of Chapter 5 we present the result of the Monte Carlo Gradients procedure.

4.2 Towards Analysis of FCNs Through Information Theoretic Learning

Up until now, we have considered several approaches to address the lack of methods for analyzing DNNs that have focused on network gradients or a Bayesian approach to DNNs. Yet, recent works have begun examining the possibility of understanding DNNs through the lens of information theory and Information Theoretic learning (ITL) [79–84]. Information theory was originally proposed by Claude Shannon and studies the properties of data using measures such as entropy and mutual information [85]. In information theory, entropy also referred to as Shannon entropy, is a measure of the uncertainty associated with a random variable and defined as

$$H(X) = E[-\log_b(P(X))] \tag{4.3}$$

where X is discrete random variable with possible values $\{x_1, ..., x_n\}$ and $P(X)$ is the probability mass function. Mutual information is a measure of the mutual dependence between two variables and can be expressed as

$$I(X;Y) = H(X) + H(Y) - H(X,Y), \tag{4.4}$$

where Y is discrete random variable with possible values $\{y_1, ..., y_m\}$, $H(X)$ is the entropy of X, $H(Y)$ is the entropy of Y and $H(X,Y)$ is the joint entropy of X and Y defined as

$$H(X,Y) = -\sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i, y_i)\log_b(P(x_i, y_i)), \tag{4.5}$$

where $P(x_i, y_i)$ is the joint probability of the values x_i and y_i occurring together. Another quantity that often occurs in information theory is the conditional entropy, defined as

$$H(Y|X) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i, y_i)\log_b\Big(\frac{P(x_i)}{P(x_i, y_i)}\Big). \tag{4.6}$$

The units of the different quantities are determined by the base of the logarithm, denoted as b in Equation 4.3, 4.5 and 4.6, which is usually chosen to be 2, resulting in the various quantities being measured in bits. Figure 4.1 displays a diagram with the different information quantities and the relationship between them. The red circle to the left represents the entropy of the random variable $\mathbf{X}$ while the yellow circle to the right represents the entropy of the random variable $\mathbf{Y}$. The total region covered by the two circles represent the joint entropy while the non-overlapping regions of the circles are the conditional entropies. In the overlapping region between the two circles we find the mutual information between the two random variables, represented in orange. The theory of Shannon was originally developed based on discrete random variables, but was eventually extended to include continuous random variables [86].

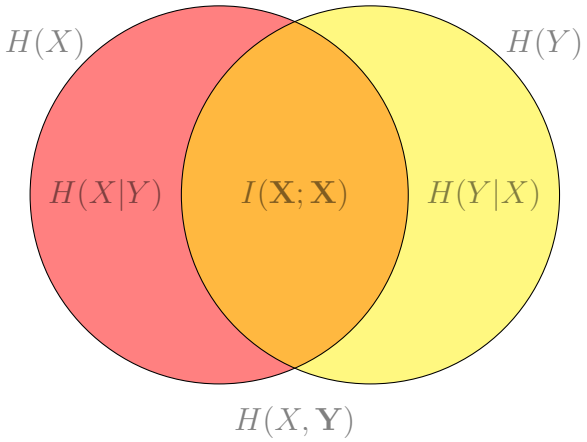


FIGURE 4.1: Diagram illustrating mutual information. The red circle to the left represents the entropy of the random variable X while the yellow circle to the right represents the entropy of the random variable Y. The total region covered by the two circles represent the joint entropy while the non-overlapping regions of the circles are the conditional entropies. In the overlapping region between the two circles we find the mutual information between the two random variables, represented in orange.

One of the first works on analyzing DNNs using information theory investigated the role of learning in deep architectures [80] by analyzing the information plane of the network. The information plane refers to the plane of the mutual information values that layer preservers on the input and output variables, i.e. compute $I(X, H_l)$ and $I(Y, H_l)$, where H_l refer to the l^{th} layer of a network with L layers treated as a random variable, and plot $I(X, H_l)$ vs $I(Y, H_l)$ for all layers. They showed that most of the training procedure is spent on compression of the input into a useful representation and not on fitting the training labels and that overfitting only occurs when the training error becomes small. However, following their findings a discussion submerged that questioned if the results generalized to arbitrary networks [87], which suggest that

further studies are needed. Nevertheless, they opened the door toward a new approach for analyzing DNNs with many promising directions going forward.

Another promising approach proposed during the last couple of months is using ITL [88] for analyzing DNNs [83]. As an extension to Shannon-based information theory, ITL employs Renyi's α-entropy [89] and Parzen windowing [90] to estimate information quantities. Renyi's α-entropy is defined as

$$H_\alpha(p) = \frac{1}{1-\alpha}\log \int p_X^\alpha(\mathbf{x})d\mathbf{x} \tag{4.7}$$

where $p_X(\mathbf{x})$ is the probability density function of the random variable X generating the data set $\boldsymbol{\mathcal{D}} = \{\mathbf{x}_1, ..., x_N\}$. Modelling $p_X(\mathbf{x})$ in the ITL framework is usually done in a non-parametric fashion using Parzen window density estimation, given by

$$\hat{p}_X(\mathbf{x}) = \frac{1}{N}\sum_{\mathbf{x},\in\mathcal{D}} \kappa_\sigma(\mathbf{x}, \mathbf{x}_t). \tag{4.8}$$

In Equation 4.8, $\kappa_\sigma(\mathbf{x}, \mathbf{x}_t)$ is know as the Parzen window, or kernel, centered as $\mathbf{x}_t$ with the width of the window controlled by the parameter σ. Note that for $\hat{p}_X(\mathbf{x})$ to be a proper density $\kappa_\sigma(\mathbf{x}, \cdot)$ must also be a density function [91], where a typical example of such a kernel is the Gaussian kernel $G_\sigma(\cdot)$ (Equation 2.23 from Section 2.1.2.1 of Chapter 2). Generally in machine learning, and particularly in deep learning, features often live in a high-dimensional space and few samples are available that can make density estimation difficult. However, $\alpha = 2$ gives rise to what is know as Renyi's quadratic entropy given by

$$H_2(p) = -\log \int p_X^2(\mathbf{x})d\mathbf{x}, \tag{4.9}$$

which, when combined with Parzen window density estimation, yield a convenient expression for the estimated entropy. Assuming a Gaussian kernel $G_\sigma(\cdot)$ with standard deviation σ for the Parzen window in Equation 4.8 and plugging the estimated probability distribution function into Equation 4.9 gives the following estimate for Renyi's quadratic entropy:

$$\begin{aligned}
\widehat{H}_2(\mathbf{x}) &= -\log \int_{-\infty}^{\infty} \Big(\frac{1}{N}\sum_{i=1}^{N} G_\sigma(\mathbf{x}-\mathbf{x}_i)\Big)^2 d\mathbf{x} \\
&= -\log \frac{1}{N^2} \int_{-\infty}^{\infty} \Big(\sum_{i=1}^{N}\sum_{j=1}^{N} G_\sigma(\mathbf{x}-\mathbf{x}_i) G_\sigma(\mathbf{x}-\mathbf{x}_j)\Big) d\mathbf{x} \\
&= -\log \frac{1}{N^2} \sum_{i=1}^{N}\sum_{j=1}^{N} \int_{-\infty}^{\infty} G_\sigma(\mathbf{x}-\mathbf{x}_i) G_\sigma(\mathbf{x}-\mathbf{x}_j) d\mathbf{x} \\
&= -\log \Big(\frac{1}{N^2} \sum_{i=1}^{N}\sum_{j=1}^{N} G_{\sigma\sqrt{2}}(\mathbf{x}_i-\mathbf{x}_j)\Big).
\end{aligned} \tag{4.10}$$

To obtain this result, notice that the integral of the product of the two Gaussian in the third line of Equation 4.10 is exactly evaluated as the value of the Gaussian computed at the difference of the arguments and whose variance is the sum of the variances of the two original Gaussian functions. In practice, this means that estimating Renyi's quadratic entropy can be done by only considering pairs of samples, thus avoiding the potentially difficult high dimensional density estimation that would have been necessary if we wanted to estimate Shannon or α-Renyi entropy for all $\mathbf{x}$. From this quadratic entropy, we can go on to find mutual information and other quantities that allows for further analysis.

However, the last couple of months have seen the development of an alternative approach to entropy estimation that bypass the probability density estimation but still wield the ITL framework [92]. The authors of [92] wanted to define an entropy measure directly from data, which they achieved by defining functionals on matrices with certain properties. For a function to be considered a measure of entropy, it must fulfill a set of axioms provided by Renyi [89] that the authors of [92] reformulated to be suitable for matrices. They could then describe matrices that would result in functionals that can be considered measures of entropy. To obtain these matrices, they evaluate a kernel $\kappa_\sigma(\cdot,\cdot)$ on all pairs of data points, where the kernel must be positive definite and also infinitely divisible [93]. Given a data set $\boldsymbol{\mathcal{D}} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, we obtain a Gram matrix $\mathbf{K}$ by evaluating $\kappa_\sigma(\cdot,\cdot)$ on all pairs of samples and can be utilized to define a quantity with properties similar to those of an entropy functional but without the need to estimate any probability density function. Explicitly, a matrix-based analogue to Renyi's α-entropy for a normalized positive definite matrix A of size $N \times N$ can be

given by the functional

$$S_\alpha(\mathbf{A}) = \frac{1}{1-\alpha}\log_2\Big(\sum_{i=1}^{N}\lambda_i(\mathbf{A})^\alpha\Big) \tag{4.11}$$

where $\lambda_i(A)$ denotes the i-th eigenvalue of A, a normalized version of K:

$$A_{ij} = \frac{1}{N}\frac{K_{ij}}{\sqrt{K_{ii}Kjj}}. \tag{4.12}$$

We can now go on to define a matrix-based equivalent for the joint-entropy, defined as:

$$S_\alpha(\mathbf{A},\mathbf{B}) = S_\alpha\Big(\frac{\mathbf{A}\circ\mathbf{B}}{\mathsf{tr}(\mathbf{A}\circ\mathbf{B})}\Big). \tag{4.13}$$

Utilizing Equation 4.11 and 4.13 we can obtain an analogous quantity to the mutual information from Equation 4.4 expressed as

$$I_\alpha(\mathbf{A};\mathbf{B}) = S_\alpha(\mathbf{A}) + S_\alpha(\mathbf{B}) - S_\alpha(\mathbf{A},\mathbf{B}), \tag{4.14}$$

which allows us to estimate the mutual information between two random variables via their gram matrices without estimating any probability density functions. However, there is still the matter of determining a kernel width (σ) that captures the structure of the data. One method for selecting the kernel width is using Silverman's rule of thumb [94], defined as

$$\sigma = h \times n^{-1/(4+d)} \tag{4.15}$$

where n is the number of samples, d is the dimensionality of the samples and h is an empirical value determined by evaluating the data. An alternative, empirically based approach is to consider the mean distance from each data point to its five nearest neighbours to determine the kernel width [95].

In [81] they put this novel ITL framework to use and examined the information plane

of an autoencoder, a particular kind of DNN designed for unsupervised learning. As in [80] they investigate the information plane of the input/output and the hidden layers, but by utilizing the matrix based entropy approach they could investigate much deeper models and more complex datasets. Another recent work investigated CNNs [84] using ITL in a similar fashion as in [81]. Figure 4.2 displays a simple MLP to illustrate how we would utilize an ITL framework for network analysis. The simple MLP consists of an input layer, two hidden layers and an output layer, where the output of each layer is treated as a random variable denoted by X, H_1, H_2 and Y, respectively. We pass N samples through the network that produce N realizations of each random variable and evaluate a kernel k on all pairs of realizations to produce four matrices, K_X, K_{H_1}, K_{H_2} and K_Y of size $N \times N$. Using Equation 4.12 we normalize each matrix to obtain A_X, A_{H_1}, A_{H_2} and A_Y and estimate the entropy of each random variable using Equation 4.11 that in turn is used to estimate the mutual information between all layers of the network.

We propose here to extend the ITL framework for FCNs. To do so, we need to address certain obstacles. Experiments performed in [81] and [84] were deeper and more complex than those used in [80], but are still fairly small compared to many FCNs that commonly have over twenty layers. The increased complexity of FCNs can produce computational difficulties, particularly for large images. Therefore, we propose to consider the encoders and the decoders of the network as random variables instead of each layer, thus reducing the number of entropy estimates. Also, in [84] they validate the Data Processing Inequality (DPI) [86], which states that for any three random variables that form a Markov chain $X \rightarrow Y \rightarrow Z$,

$$I(X;Y) \geq I(X;Z). \tag{4.16}$$

For a neural network, Equation 4.16 tell us that, for instance, the mutual information between the input and the first layer should be equal or larger than the mutual information between the input and any layer succeeding the first layer. However, FCNs often include skip connections between encoders and decoders, which means that the DPI is not necessarily valid.

We utilize this proposed ITL framework as a novel approach for analyzing FCNs and for investigating the DPI. The result of our evaluation is presented in Section 5.6 of Chapter 5.

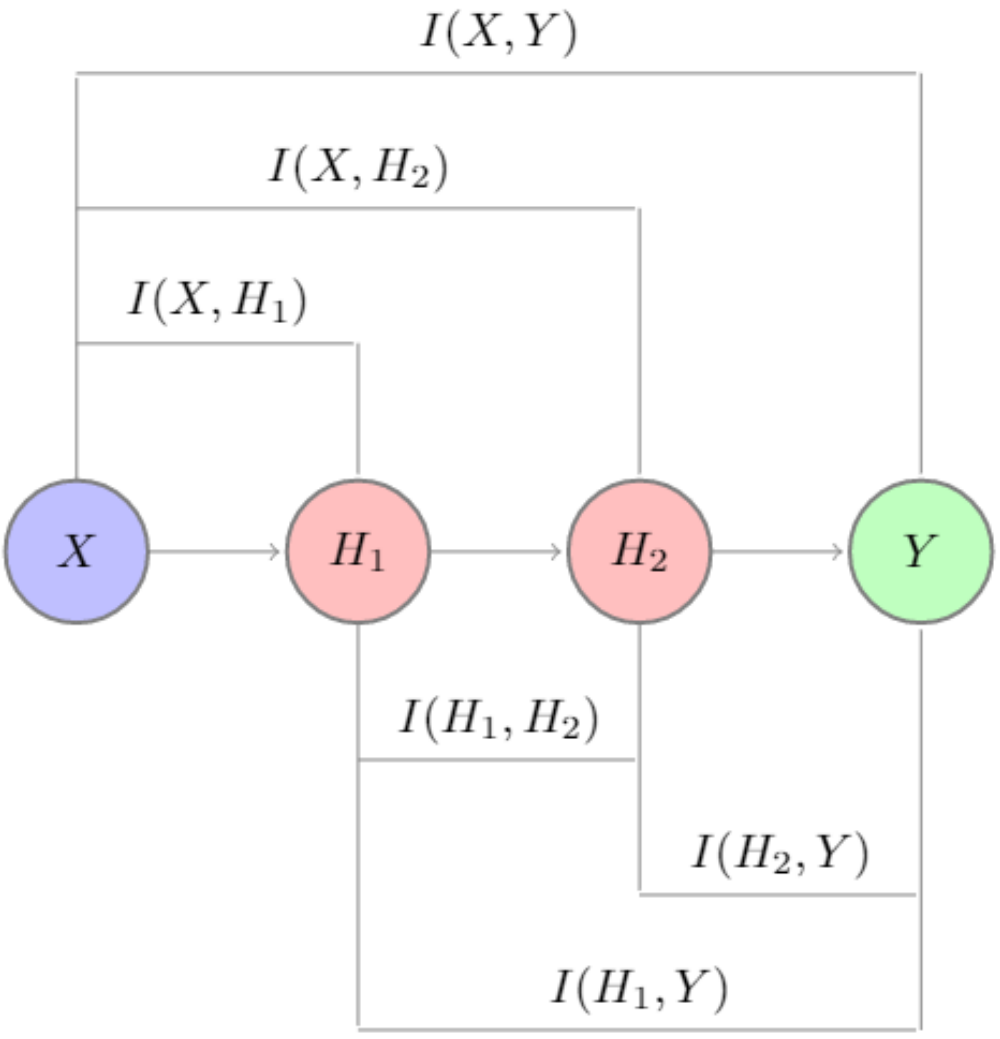


FIGURE 4.2: Figure displays an example of mutual information setup for a simple MLP, where each layer is treated as a random variable and we investigate the mutual information between all layers.

4.3 Network Details and Proposed Improvement

Deep learning research is progressing at a rapid speed with model development and improvements occurring regularly. A recent model might improve the state-of-the-art on a certain task but find no improvement on another, which makes deciding on a model a process of trial and error. Coupled with the number of available architectures and the required training time, the choice of model can become an arduous task. With this in mind, we chose to employ three established networks that have similar encoders but tackle the upsampling procedure differently, providing a diverse foundation of established deep models for developing DSSs for the task of polyp segmentation. Note that our implementation of the networks tries to be as faithful as possible to the original

model, but some modifications are made to include recent techniques or adjustment to suit the data. Such adjustments will be addressed in the description of each network.

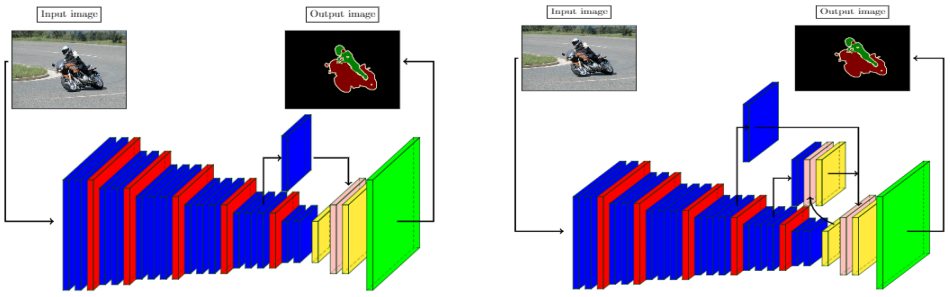
We would also like to add that we evaluate all models presented in this section on the task of semantic segmentation of colorectal polyps. The result of this evaluation is presented in Section 5.2.1 of Chapter 5.

4.3.1 Fully Convolutional Network

As mentioned in Chapter 3, Fully Convolutional Networks were introduced to tackle per-pixel prediction problems such as semantic segmentation. In [9] they proposed several architectures, where the most basic one was the FCN-32 displayed in Figure 3.16. Similar for all the architectures is the encoder network, which is based on the VGG-16 [59] and consists of five encoders. Each encoder performs convolution followed by Batch Normalization and a ReLU, a process that is repeated several times to produce a set of feature maps. Following each encoder is a 2×2 max-pooling operation with stride equal to two. In the VGG-16, the last encoder is composed of fully connected layer, but as discussed in Section 3.4, such layers are limited by their inability to process image of arbitrary size. Therefore, the fully connected layers at end of the VGG-16 is replaced with convolutional layers similar to the preceding encoders, which make up the only decoder of this network. The decoder network performs transposed convolution on the feature maps produced by this decoder to recover the resolution of the input feature maps. For the FCN-32, the output of the decoder is directly upsampled by a factor of 32 and then passed into a softmax function.

However, upsampling directly from last feature map of the encoder network limited the scale of detail in the final prediction. This problem was addressed by adding skips between the encoder network and the decoder network, thus providing features of different scales to produce the final prediction. Feature maps from the first encoder are upsampled by a factor of two and then summed with the feature maps of the final encoder, after that they are processed by an additional convolutional layer. If these feature maps are upsampled by a factor of 16, the FCN-16 displayed in Figure 4.3a is produced. If the same procedure is repeated once more, combining information from the fourth encoder before upsampling by a factor of 8, the FCN-8 is produced. The FCN-8 showed superior results compared to the FCN-32 and FCN-16, whilst combining information for earlier encoders did not improve results significantly. Dropout is

included between all layers of the first decoder with a probability of dropping a unit equal to 0.5. Convolutional layers preceding a upsampling layer is not followed by Batch Normalization or a ReLU. A detailed description of our FCN-8 implementation is presented in Table C.1 in Appendix C.1.



(A) Illustration of the Fully Convolutional Network-16 architecture from [9].

(B) Illustration of the Fully Convolutional Network-8 architecture from [9].

FIGURE 4.3: Color code description:
Pink: Sum.
Red: Pooling.
Green: Soft-max.
Yellow: Upsampling.
Blue: Convolution, Batch Normalization and ReLU.

4.3.1.1 Author Contributions and Motivation

Albeit the FCN-8 has been applied to the task of segmentation of colorectal polyp in colonoscopy images [11], we believe that utilizing recent techniques can improve results and ease the training procedure. Batch Normalization was not included in the original FCN-8 and neither in previous work [11], but is included in our implementation. Furthermore, transfer learning is applicable since the first 13 convolutional filters of the encoder network are identical to the first 13 layers of the VGG-16, enabling us to initialize the model with weights trained on larger datasets. Both these techniques have shown to improve performance and speed up training and have yet to be employed in semantic segmentation of colorectal polyps before now, to the best of the author's knowledge.

We choose to use the FCN-8 for our experiments for several reasons. It has shown impressive results on a number of different tasks, including medical image analysis analysis [9, 11, 96]. Being based on the VGG-16 architecture, it enables transfer learning that might be useful in medical image analysis where data can be sparse.

Lastly, even with many recent architecture available the FCN-8 is still widely used and serves as a natural foundation for our experiments.

4.3.2 U-Net

One of the first networks to build upon FCNs was the U-Net [71], which proposed an alternative method to recover the resolution of the data. The encoder network consists of five encoders, each of which performs 3×3 convolutions with a filter bank to produce a set of feature maps. These feature maps are batch normalized and passed into a ReLU. Every encoder performs this process twice followed by a 2×2max-pooling operation with a stride equal to two, reducing the size of feature maps by a factor of two. U-Nets decoder network consists of four decoders that process in the same manner as the encoders. The feature maps produced in the fifth encoder is upsampled by a factor of two using transposed convolution and concatenated with the feature maps produced by the fourth encoder. These combined feature maps are passed into the first decoder, which in turn is upsampled and concatenated with the feature maps of the third encoder. This process is repeated until the resolution of the input feature map is recovered. After the final decoder follows a 1×1 convolutions that map the feature vector into the desired number of classes and a softmax function. Dropout is applied at two final encoders, with a probability of dropping a unit equal to 0.5. A detailed description of our U-Net implementation is presented in Table C.2 in Appendix C.1.

4.3.2.1 Author Contributions and Motivation

As with the FCN-8, Batch Normalization was not included in the original U-Net but included by us to accelerate training and possibly increase performance. Additionally, to best of our knowledge, U-Net has yet to be used for semantic segmentation of colorectal polyps.

We consider U-Net particularly interesting for segmentation of colorectal polyp as it was originally introduced for processing biomedical image segmentation. Also, concatenating feature maps from the encoder network with feature maps from the decoder network presents and alternative method for propagating context information to higher resolution layers compared to the FCNs. Furthermore, since it is not based on another

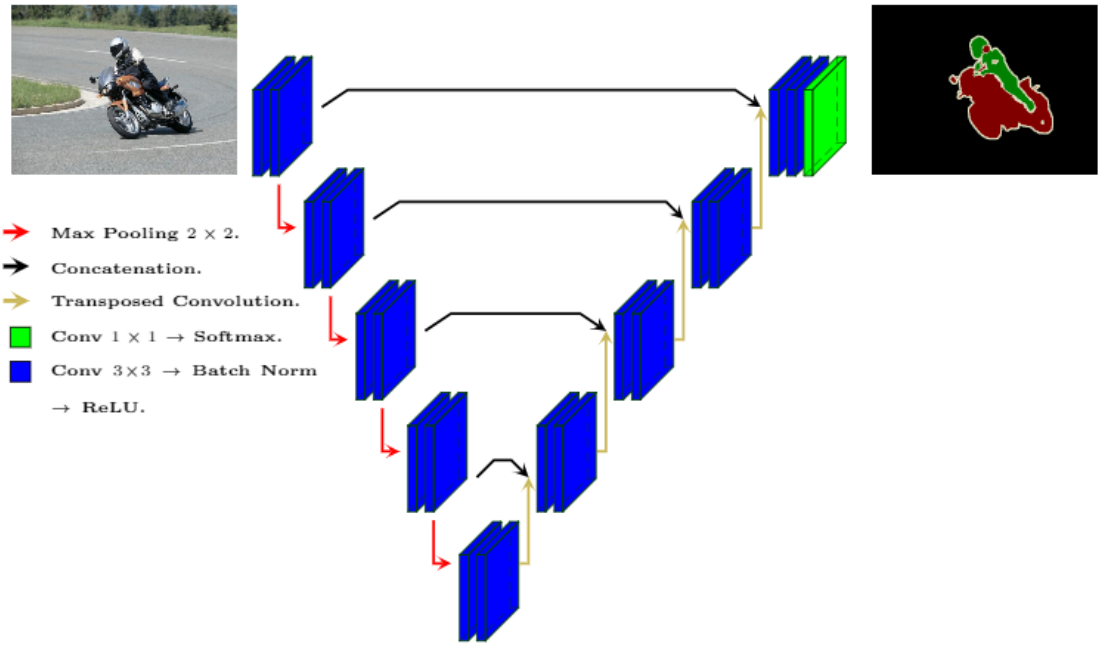


FIGURE 4.4: Illustration of the U-Net architecture from [71]. Images are obtained from [73].

CNN transfer learning is not applicable, which enables comparison between models initialized with weights trained on larger datasets. Finally, U-Nets architecture contains significantly fewer parameters than the FCN-8, and could offer a lightweight alternative if performance is comparable.

4.3.3 SegNet

Both FCNs and the U-Net rely on transposed convolution to recover feature maps with the same resolution as the input features, but SegNet [97] presents another option. SegNet consists of an encoder network and a decoder network, where the encoder network consists of five encoders and the decoder consists of five decoders. Each encoder performs convolution with a filter bank to produce a set of feature maps that are batch normalized and passed into a ReLU, a process that is repeated two times for the two initial encoders and three times for the three central encoders. Following each encoder is a 2×2 max-pooling operation with a stride equal to two, reducing the size of feature maps by a factor of two. This encoder corresponds to the first 13 convolutional layers of the VGG-16, which enables weights to be initialized from networks trained on larger datasets in a transfer learning fashion. SegNet is constructed symmetrical, such that the decoder network is identical to the encoder network but with

the max-pooling operation replaced by a max-unpooling operation. Figure 4.5 displays an example to illustrate how this max-unpooling operation is carried out. When a feature map is downsampled the max-pooling indices are stored and used at a later stage to perform non-linear upsampling, a procedure with several advantages. Firstly, it produces sparse feature maps that are computationally attractive and implicit feature selectors. Secondly, it removes the need to learn additional filter for upsampling, thus reducing the number of parameters in the model. In the final convolutional layer of the final decoder there is no Batch Normalization or a ReLU, but instead a softmax function. A detailed description of our SegNet implementation is presented in Table C.3 in Appendix C.1.

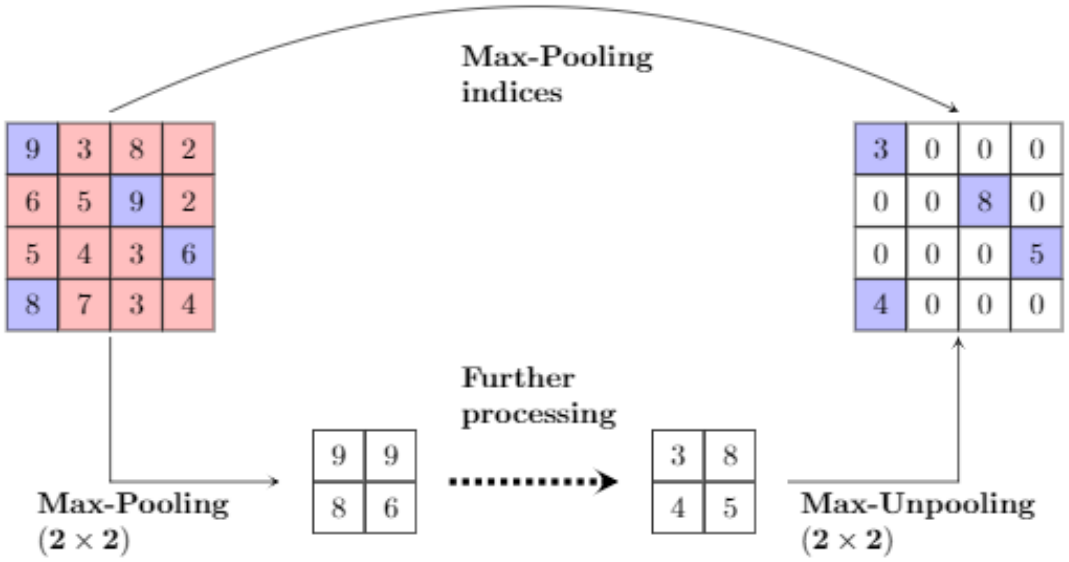


FIGURE 4.5: Illustration of the upsampling procedure employed in SegNet. A (2×2) max-pooling operation is applied to a (4×4) feature map, producing a (2×2) features map. This feature map is processed further, and upon upsampling the new feature map is inserted into an empty feature map using the indices from the max-pooling operation.

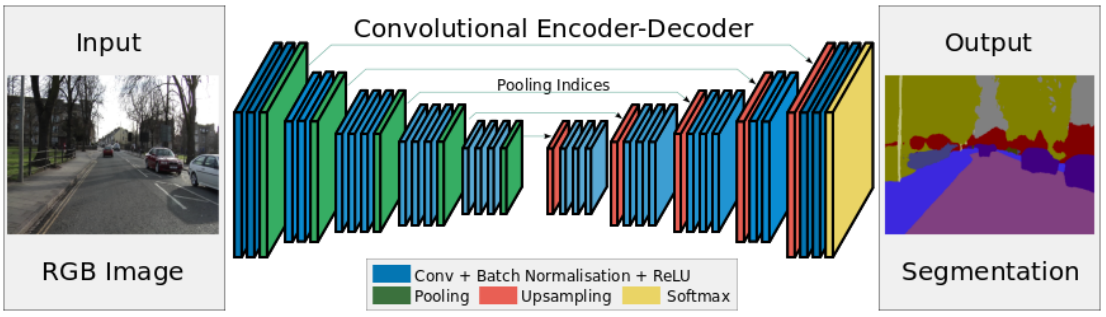


FIGURE 4.6: Illustration of the SegNet architecture from [97]. Figure obtained from [97].

4.3.3.1 Author Contributions and Motivation

In contrast to the FCN-8 and U-Net, SegNet has Batch Normalization already included. But we chose to add Dropout in the three central encoders and decoders, similar to [98]. Adding Dropout helps with regularizing the model and enables Monte Carlo Dropout and Monte Carlo Gradients. Furthermore, to best of our knowledge, SegNet has yet to be used for semantic segmentation of colorectal polyps.

SegNet was primarily motivated by scene understanding application and has yet to see wide use in medical image analysis. Including SegNet in this work will show if the novel upsampling procedure can bring advantages in the processing of medical images. Furthermore, has a comparable number of parameters to U-Net, providing another lightweight alternative to the FCN-8.

Chapter 5

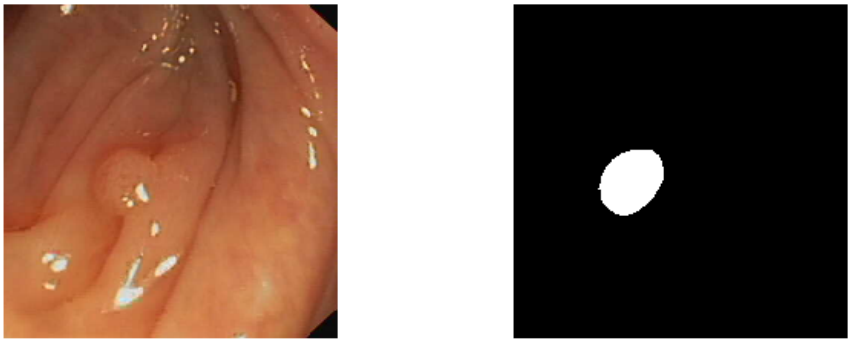
Results

At this point, the preceding chapters have aimed at providing the reader with all the tools necessary for understanding the models and techniques utilized to conduct the investigations presented in this chapter. Analysis carried out in this thesis are aimed at determining the potential of deep models as the foundation for DSSs and investigating techniques that seek to increase the interpretability of such models. But before we present our analysis, we give a general overview of components which are common for all models implemented.

5.1 Experimental Setup

All evaluations were conducted on the publicly available Endoscene dataset [11], which consists of 912 images of colorectal polyps obtained from 44 video sequences acquired from 36 patients with annotated images included as ground truth. We split the data into training, validation, and test set following the procedure shown in [11], and the interested reader can find a detailed description of this split in Section D.1 of the Appendix. Figure 5.1 displays an example of a pair of samples from the dataset, where the leftmost image is the original image obtained during the colonoscopy and the rightmost image is the annotated image. Each pixel is labeled as belonging to a polyp or the background, where white pixels correspond to the polyp class, and black pixels correspond the background class. Moreover, Figure 5.1 illustrates that the polyp generally occupies a much smaller proportion than the background class, making the

dataset unbalanced. We experimented with class balancing using Median Frequency Balancing[1]. (MFB) [99], but found no significant improvement.



(A) Original image. (B) Annotated image.

FIGURE 5.1: Example of sample pair from the Endoscene dataset. Leftmost image is the original image take during a colonoscopy and the rightmost image is annotated image. Each pixel is labeled as pixel or background, where white correspond to a pixel being labeled as polyp and black as a pixel belonging to the background.

We report our results using two metrics, global accuracy and Intersection over Union (IoU). For a given class c, prediction $\hat{y}_i$, and label y_i, accuracy is defined as

$$\text{Acc} = \frac{1}{N}\sum_i (\hat{y}_i == c \wedge y_i == c), \tag{5.1}$$

where $\wedge$ denotes the logical *and* operation, N is the total number of pixels in the dataset and the sum goes over all pixels in the dataset. Intersection over Union is defined as

$$\text{IoU} = \frac{\sum_i (\hat{y}_i == c \wedge y_i == c)}{\sum_i (\hat{y}_i == c \vee y_i == c)}, \tag{5.2}$$

where $\vee$ denotes the logical *or* operation. To calculate the global accuracy and mean IoU we compute the score for each class and average over the classes. We employ two measure of evaluation because the global accuracy can be misleading in cases where one class is highly over-represented in a sample. For instance, an image containing a small polyp will have a mostly black annotated image, which means that a model can produce a completely black output and still achieve a high global accuracy. But the

[1]See Section A.1 of the Appendix for description of MFB.

IoU metric punish erroneous predictions harder, which gives a more precise description of the model's actual performance.

Data augmentation was performed during the training of all models, which include cropping, rotation, flipping, shearing, and zooming. A 224×224 patch is randomly extracted from either the center or one of the four corners of an incoming training image and its corresponding ground truth. Next, rotation between -60 and 60 degrees, zoom from 0.7 to 1.4, shearing between -30 and 30 degrees is applied randomly to the cropped image. Also, after these transformations, the resulting image is flipped with 0.5 probability. Another thing to note is that such a heavy augmentation procedure can inhibit convergence if images are distorted too much. We experienced that applying all transformations from the beginning of the training caused convergence difficulties. Therefore, we only cropped the samples for the first 100 epochs and included the transformations after this point, which allowed for simpler training samples during the initial training when the weights need the largest adjustment. We chose to start transforming the training samples after 100 epochs by monitoring the results on the validation set and observing that they had begun to level for all models at this point.

Early stopping was utilized during the training of all models. We monitor the IoU score for the polyp class on the validation set with a patience of 50 epochs. Monitoring the IoU score for the polyp class instead of, for example, the mean IoU score was done to encourage the selection of models with high polyp detection, a desirable goal for two reasons. First, as the dataset is unbalanced, a high score can be achieved by classifying most pixels to the background class. Second, from a medical point of view, missing a polyp is a more severe mistake than assigning parts of the background to the polyp class.

All models were trained using the backpropagation algorithm and the ADAM optimizer [100], a recent adaptive version of gradient descent that is presented in Section B.1 of the Appendix. We use the cross-entropy cost function, also presented in Section B.1 of the Appendix for the benefit of the reader, with a batch size of 10. All evaluations were performed using the deep learning framework Pytorch[2] on a single Titan X Graphics Processing Unit (GPU).

[2] http://pytorch.org/

5.1.1 Training Approach Discussion

There are several decisions reported in this section that could have warranted a more thorough analysis but were taken based on former studies or experimental results during the design of the networks that are not presented. This is partly to benefit the reader, as some details are not considered crucial parts of the models and partly because some details would require such an in-depth analysis that it would be beyond the scope of this thesis. However, we will discuss some of these points briefly to justify our decisions.

First, picking the range for the different transformations can be considered a set of hyperparameters that could be experimentally determined by monitoring the validation set. A high degree of augmentation can result in models that generalize better to unseen data, but it might also distort the images too much that could worsen performance. Determining the ideal range for all transformations would require tuning the range of each transformation separately and for each model, which we consider outside the scope of this thesis. Instead, we rely on a previous study that examined the effect of data augmentation that showed increased performance resulting from data augmented in the ranges described above [11].

Second, the number of epochs chosen for the patience hyperparameter was determined experimentally during the design of the models. A large patience might allow the network to find a better configuration resulting in a higher performance, while a smaller patience might prevent the models from overfitting. We experience that 50 epochs gave a sufficiently good stopping criterion, but acknowledge that it could benefit from further study.

Third, using the ADAM optimizer to update the weight might seem arbitrary. As described in Section B.1, there are several recent optimization techniques available that make interesting alternatives, such as RMSprop [101]. Common for these recent optimization techniques and the ADAM optimizer is their automatic adjustment of update parameters, which removes the need to fine-tune the learning rate, a procedure that can be difficult and time-consuming. Determining which optimizer provides the greatest benefit could therefore be an interesting study but we consider it outside the scope of this thesis.

Fourth, the batch size is sometimes considered a hyperparameter of its own. A large batch size results in a better estimate of the gradients and could result in faster and

more stable convergence, but comes at a higher computational cost. In our case, we were restricted by the amount of memory available on the GPU and landed on 10 images in each batch for practical reasons.

Lastly, the limited effect of the MFB is curious, seeing that it has shown improved performance in other cases [102]. We experience similar results with and without class balancing, which might be an effect of the cropping procedure. By extracting patches we remove a large amount of pixels from the image, where the majority corresponds to the background class, thus indirectly balancing the classes.

5.2 Development of DSSs for Semantic Segmentation of Colorectal Polyps

We evaluated the three models introduced and enhanced in Section 4.3.1, Section 4.3.2, and Section 4.3.3 of Chapter 4 on semantic segmentation of colorectal polyps that we refer to as the Enhanced FCN-8 (EFCN-8), Enhanced SegNet (ESegNet), and Enhanced U-Net(EU-Net). The primary aim was to investigate the potential of FCNs as DSSs for colorectal polyp segmentation and to compare results from different models.

5.2.1 Results

In Table 5.1, we report our results for the models mentioned above along with results from both hand-crafted and deep learning based approaches. The hand-crafted method computes a histogram based on the pixel values and uses peaks and valleys information from the histogram to perform their segmentation and is referred to as the Segmentation from Energy Maps (SDEM) algorithm [34]. For the deep learning approach, segmentation is performed using the FCN-8, but without Batch Normalization or transfer learning. This approach is referred to as FCN-8.

Figure 5.2 displays the cost versus epochs for EFCN-8, ESegNet and EU-Net. Notice the sudden jolt at the 100 epoch when the data augmentation commences. A single epoch for EFCN-8 took 1 minute and 40 seconds, 1 minute and 30 for ESegNet and 2 minute and 35 seconds for EU-Net. For FCN-8, it took 326 epochs before the training was stopped using early stopping, which resulted in a total training time of

Model	# Parameters(M)	IoU background	IoU polyp	Mean IoU	Global Accuracy
SDEM [34]	-	0.799	0.221	0.412	0.756
EU-Net	27.5	0.945	0.516	0.723	0.945
ESegNet	29.5	0.933	0.522	0.727	0.935
FCN-8 [11]	134.5	**0.946**	0.509	0.727	**0.949**
EFCN-8	134.5	**0.946**	**0.587**	**0.767**	**0.949**

TABLE 5.1: Results on test set.

approximately 9 hours. ESegNet ran for 338 epochs before being stopped, resulting in a total training time of approximately 8 hours and 45 minutes. EU-Net ran for 239 epochs, resulting in a total training time of approximately 6 hours and 30 minutes.

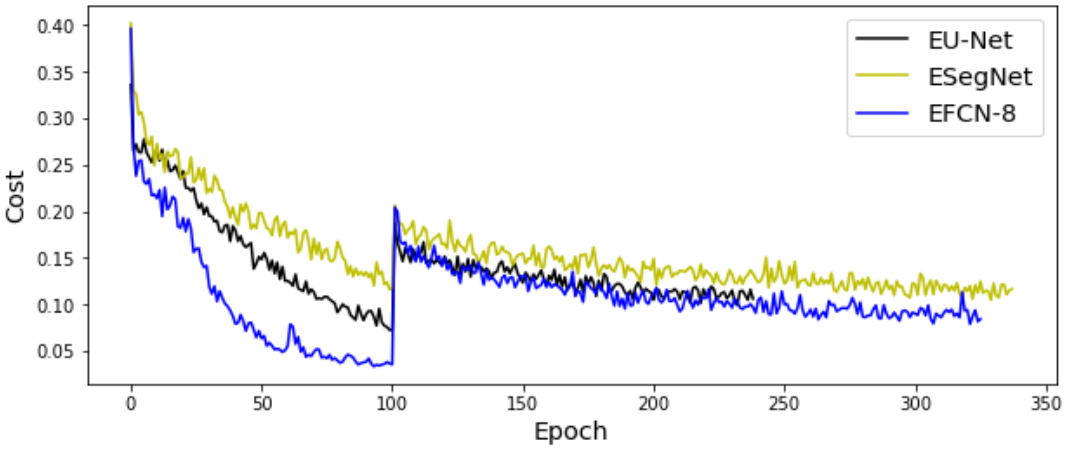


FIGURE 5.2: Cost convergence of EFCN-8, ESegNet and EU-Net on the training set.

As previously stated, we monitor the IoU score for the polyp class on the validation for model selection. Figure 5.3 displays the IoU score for the polyp class versus epochs on the validation set for EFCN-8, ESegNet, and EU-Net, where the red markers indicate which model was selected. Figure 5.4 displays all metrics evaluated on the test set. It shows, from top to bottom, the mean accuracy, IoU for background class, IoU for polyp class and mean IoU score.

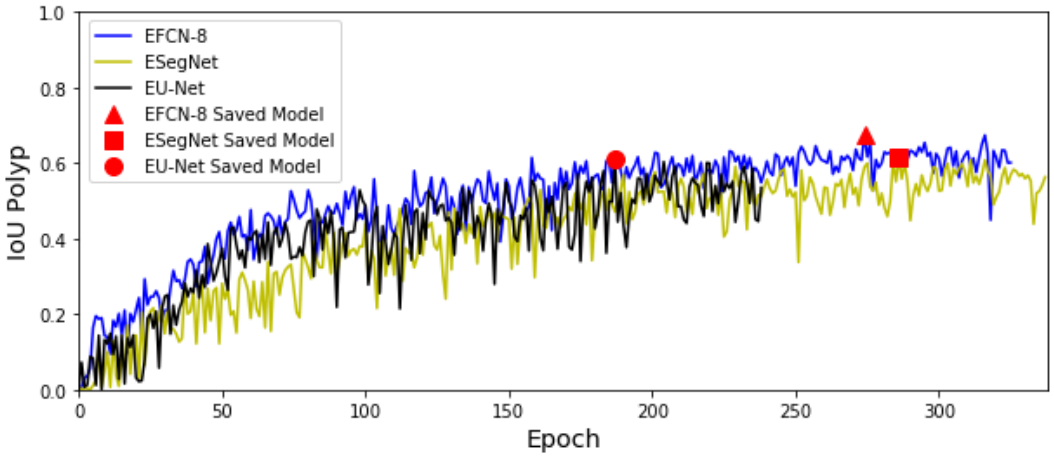


FIGURE 5.3: Graph displaying the IoU score for the polyp class on the validation set.

Finally, we present some qualitative results from each model, displayed in Figure 5.5 and 5.6. Each row in both Figures represents, from top to bottom, original image, ground truth, EFCN-8 prediction, ESegNet prediction and EU-Net prediction. In Figure 5.5, the first three columns show examples where all models successfully segment the polyp present in the image, while the last two columns show examples where all models fail to segment the polyp. In Figure 5.6, the first two columns present examples where the EFCN-8 successfully segments the polyp, and both ESegNet and EU-Net produce poor segmentation. Column three in Figure 5.6 presents an example where EFCN-8 and ESegNet both successfully segment the polyp, but EU-Net fails. Column four in Figure 5.6 shows an example where both ESegNet and EU-Net achieve a good segmentation, but this time, EFCN-8 produce a somewhat flawed prediction. The fifth and final column in Figure 5.6 displays an example where all models fail to find the actual polyp. At the same time, all models segment a part of the column belonging to the background class as a polyp.

5.2.2 Discussion

There are multiple aspects of these results that are interesting to address, and we start by looking at the quantitative results shown in Table 5.1. Before FCNs were introduced, the SDEM was one the highest performing algorithms on polyp segmentation, among others. What is evident from Table 5.1 is the difference in performance between the hand-crafted method and the deep methods, particularly for the IoU score for the polyp class and mean IoU. Achieving a high global accuracy or high IoU score for

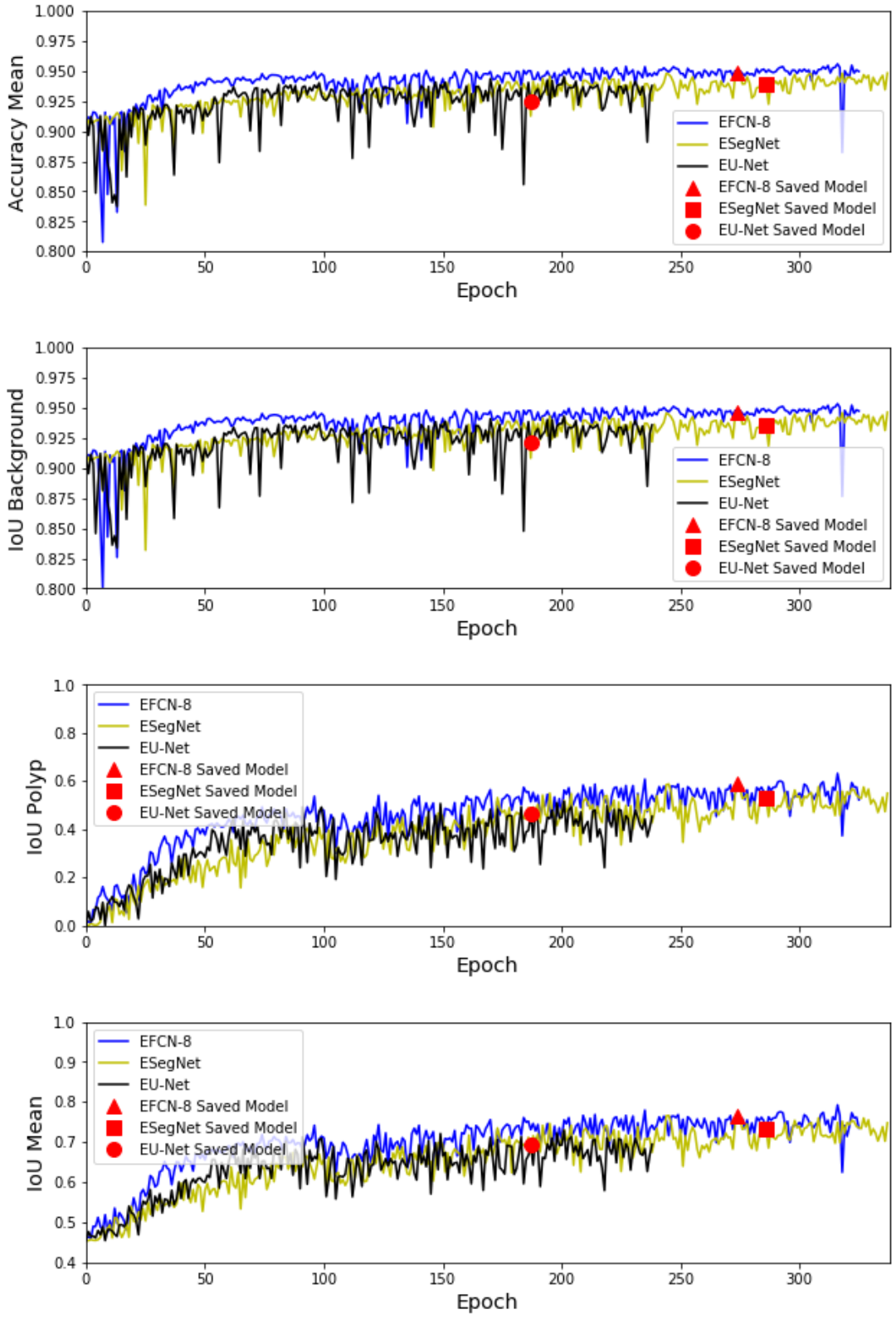


FIGURE 5.4: Figure displays, from top to bottom, mean accuracy, IoU score for background class, IoU score for polyp class and mean IoU score for EFCN-8, ESegNet and EU-Net on the test set.

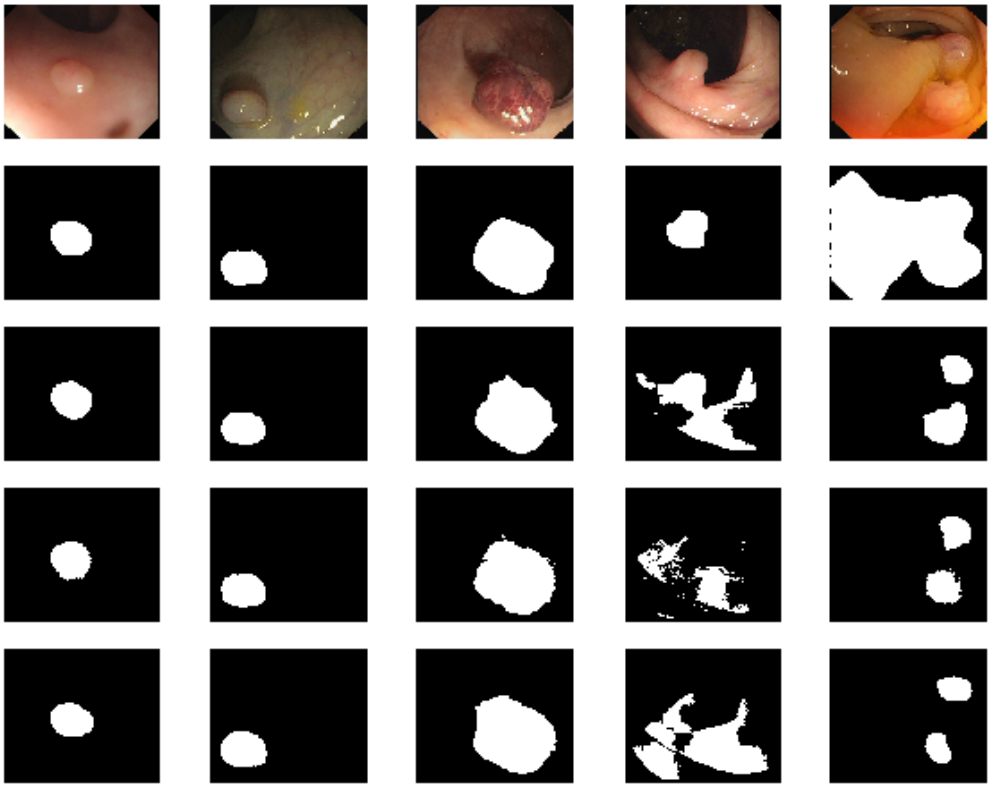


FIGURE 5.5: Qualitative results on the Endoscene dataset. Pixels labeled as white correspond to the polyp class and pixels labeled in black correspond to the background class. Each row represents, from top to bottom, original image, ground truth, EFCN-8 prediction, ESegNet prediction and EU-Net prediction. First three columns display examples where all models successfully segment the polyp, while the last two columns show examples where all models fail to segment the polyp.

the background class is easier since simply predicting a black canvas will result in a fairly high global accuracy and high IoU score for the background class because of the majority of background pixels. Achieving a satisfying IoU score for the polyp class is more challenging but also what we desire. After all, the goal is to detect and locate polyps. This is also why we chose to monitor the IoU score for the polyp class for the early stopping. Interestingly, EU-Net and ESegNet achieve comparable results to the FCN-8 from previous work but with much fewer parameters. Our EFCN-8 implementation, on the other hand, EFCN-8, significantly increases performance, which we attribute to the inclusion of Batch Normalization and transfer learning.

Another interesting point is the similar results of EU-Net and ESegNet that have a comparable number of parameters but different upsampling procedures. As presented in the previous section, ESegNet had the shortest training time per epoch but took

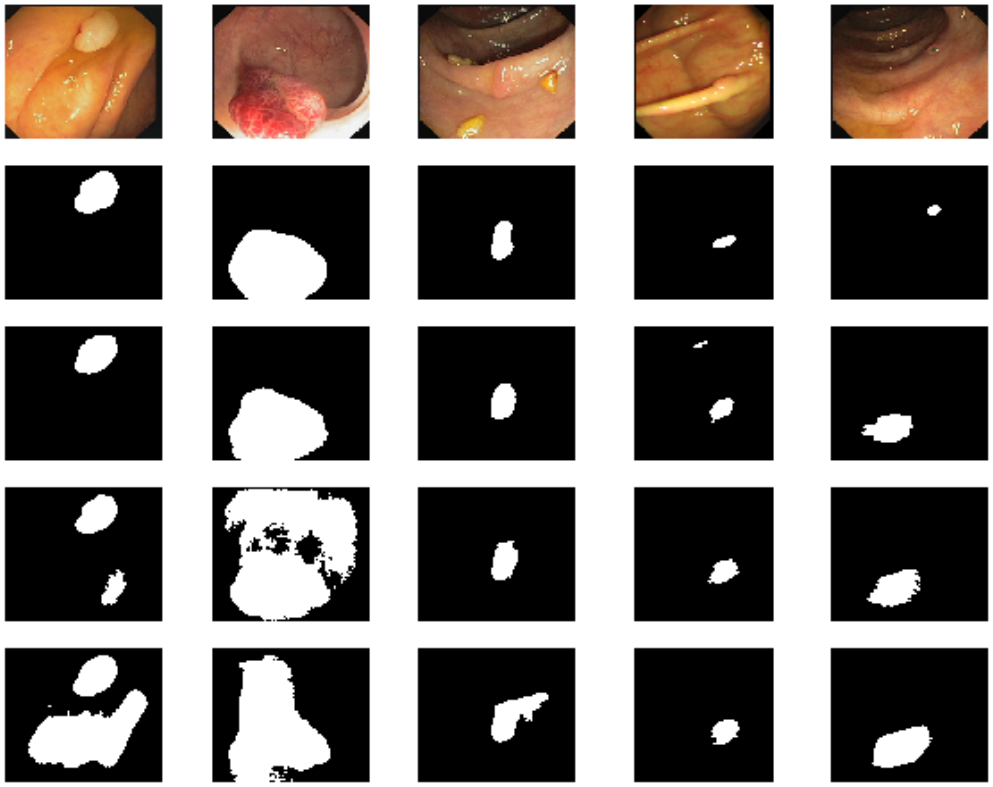


FIGURE 5.6: Qualitative results on the Endoscene dataset. Pixels labeled as white correspond to the polyp class and pixels labeled in black correspond to the background class. Each row represents, from top to bottom, original image, ground truth, EFCN-8 prediction, ESegNet prediction and EU-Net prediction. The first two columns present examples where the EFCN-8 successfully segments the polyp and both ESegNet and EU-Net produce poor segmentation. Column three presents an example where EFCN-8 and ESegNet both successfully segment the polyp while EU-Net fails. Column four presents an example where both ESegNet and EU-Net achieve a good segmentation, but this time, EFCN-8 produce a somewhat flawed prediction. The fifth and final column displays an example where all models fail to find the actual polyp. At the same time, all models segment a part of the column belonging to the background class as a polyp.

longer before stopping. EU-Net spent most time per epoch, but stopped training earlier. The difference in training time per epoch is a result of ESegNet utilizing the already computed pooling indices for the upsampling while EU-Net must perform the transposed convolutions. But the difference in convergence is more curious. We observed similar behaviour during the design phase of the networks and, judging by Figure 5.2 and 5.3, it does seem like EU-Net converges faster than ESegNet. One possible explanation could be that the flexibility of the learned upsampling kernels supports faster convergence. Nevertheless, they both perform worse than the EFCN-8,

which seems to suggest that the superior capacity of the EFCN-8 combined with recent techniques such as Batch Normalization is important for performance.

Regarding Figure 5.2, which presents the cost for all models, it is worth noting the difference before and after data augmentation commences. Up until 100 epochs, the EFCN-8 has already reached a cost of approximately 0.05 and begun plateauing, but after data augmentation begins it jumps up to 0.20 and never gets below 0.10. This might suggest that augmentation inhibits the model, but by inspecting Figure 5.3, which keeps increasing after 100 epochs, it is clear that the data augmentation encourages models better suited for unseen data. Similar behaviour is displayed by ESegNet and EU-Net, which also obtain an increased cost but enhanced validation score.

Turning to Figure 5.4, by observing which models are selected we can see that there were models that would yield a higher performance on the test set. But in general, the selected models do give a truthful representation of the capabilities of the network. Do note that the global accuracy and IoU background graphs have a different range on the vertical axis. As discussed previously, a completely black prediction will still yield a high global accuracy and IoU score for the background class, therefore these graphs stay approximately unchanged during the training of the network.

For the qualitative results, we consider Figure 5.5 first. The first three columns presented examples where all models successfully segment the present polyp, which we included to demonstrate that all networks are capable of producing clear and precise predictions. Many of the images in the Endoscene dataset contain polyps with an appearance similar to the one in the three leftmost columns of Figure 5.5, that is, elliptical and with a distinct edge. In these cases, all models produced mostly precise predictions. However, when the appearance of the polyp was more irregular, the task became more challenging. One problem that routinely troubled the models was the presence of glare from the camera light. In the fourth column of Figure 5.5 we presented an example where all models fail to provide a correct segmentation. Notice that the shape is somewhat irregular and also several specs of glare below the actual polyp. This seems to confuse the networks, which classify only part of the polyp as a polyp and also classify large parts of the colon as a polyp. The fifth and final column of Figure 5.5 displays an example containing a polyp that spans a large part of the image. All models agree on parts of the image, but they unanimously miss the greater proportion of the polyp. This is most likely because few of the training images contained such highly contorted polyp, and obtaining improved predictions in such cases

would require more diverse polyps in the training set.

Figure 5.6 is intended to provide examples of more challenging images. In the first two columns, we see an example where EFCN-8 successfully segments the polyp while both ESegNet and EU-Net fail. Many of the test images either saw all models segment successfully, neither segments successfully or the EFCN-8 provided a successful segmentation while ESegNet and EU-Net failed, which supports the quantitative results from Table 5.1 where EFCN-8 achieved the highest performance. Column three of Figure 5.6 displays an example where the original image contains yellow growths that might, to an untrained eye, be polyp suspects. But, both EFCN-8 and ESegNet disregard the suspicious growths and identify the true polyp, and also EU-Net but with slightly lower precision. The fourth column presents an example where ESegNet and EU-Net achieve successful segmentation but EFCN-8 is somewhat imprecise, demonstrating that EFCN-8 is not infallible. Judging by this example it is not obvious why EFCN-8 is less precise in this particular case, but by inspecting the original image in column 4 it is possible to see that EFCN-8 is segmenting part of a ridge as a polyp, in particular, a part of the ridge where there is some degree of glare present. One hypothesis might be that the greater number of parameters enables more attention to detail but can also result in the model being overly sensitive. However, for the majority of the images in the test set, EFCN-8 produced the clearest and precise predictions. In the last column of Figure 5.6 we present a curious example where all models miss the actual polyp but agree on a wrong prediction. Again, by inspecting the original image we can gain some insight into the shortcomings of the three models. There is no doubt that the example in column 5 is a challenging specimen, considering that the polyp make up a tiny region of the image. Furthermore, to an untrained eye the region segment as polyp might seem like a somewhat likely polyp candidate.

When discussing the qualitative results in Figure 5.5 and 5.5 we can hypothesize and assume the reasoning behind the predictions, but it is difficult to make definite statements. For example, how certain is the model about its prediction? What features are influencing its prediction? Are the different models affected by the same features? To answer such questions requires further investigation using the techniques outlined toward the end of Chapter 3, and in the beginning of Chapter 4.

5.3 Estimating Uncertainty in DSSs Based on DNNs

We estimated the uncertainty of the predictions presented in the previous section using the Monte Carlo Dropout described in Section 3.5.1 of Chapter 3. This analysis was conducted to asses the potential of Monte Carlo Dropout as tool to create more trustworthy DSSs and as a method to provide more room for comparison between models. We revisit five examples from Section 5.2 to illustrate the result from the uncertainty estimation.

5.3.1 Results

Figure 5.7, 5.8 and 5.9 display the results of the uncertainty estimation of EFCN-8, ESegNet and EU-Net, respectively, for the examples shown in column 1 and 2 of Figure 5.5 and column 1, 3 and 5 of Figure 5.6. Each row represents, from top to bottom, original image, ground truth, prediction and uncertainty map. The uncertainty maps were computed by drawing 10 samples from the network with a dropout rate of 0.5 and estimating the standard deviation from those samples. Dark blue pixels are associated with low uncertainty while bright green pixels are associated with high uncertainty.

5.3.2 Discussion

Results of the uncertainty estimation for each model is presented in separate figures, such that each model is given a clear evaluation before we discuss the differences between the models. We start by examining Figure 5.7, which displays the results of the uncertainty estimation for EFCN-8's predictions. First, the first four columns present examples where EFCN-8 successfully segments the polyp and the uncertainty maps associated with these predictions have a similar appearance. Most regions of the image contain pixels with low uncertainty, but regions around the actual polyp stand out. Intuitively, this should not be surprising seeing that even physicians will have a hard time pinpoint the exact point where the colon ends, and the polyp starts. If someone were to make a diagnosis based on EFCN-8's prediction but felt skeptical basing a decision on just a single image, including such uncertainty maps in the analysis clearly

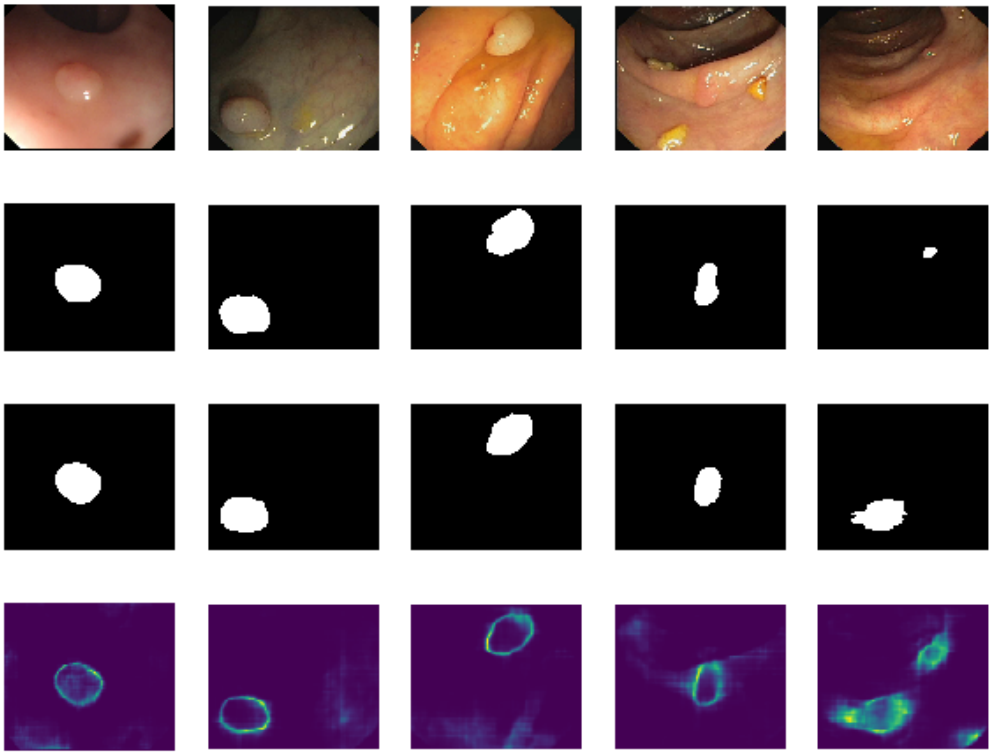


FIGURE 5.7: Figure displays EFCN-8's predictions and the uncertainty map associated with the predictions. Each row represents, from top to bottom, original image, ground truth, prediction and uncertainty map. For the uncertainty maps, dark blue pixels are associated with low uncertainty and bright green pixels are associated with high uncertainty.

gives a more thorough analysis and provide some assurances to the person performing the analysis. In particular, column 4 in Figure 5.7 makes an interesting case, where there are peculiar growths that could spur uncertainty in someone assessing the images. However, by consulting the uncertainty map, it is apparent that the model is only concerned with the region in the center of the image. Another interesting example is column 5 in Figure 5.7, where the model segments part of the column as a polyp and completely misses the polyp that is present in the image. Again, by only considering the prediction one might be reluctant to make a decision, so we turn to the uncertainty map for further details. This example clearly stands out compared to the examples shown in column 1 to 4. One thing to note is that the regions of uncertainty are larger and the model is not just uncertain about the border of the polyp, but also the polyp itself. Also, the model is not just uncertain about regions where it has predicted the presence of a polyp but also about a completely separate region that, by inspecting the ground truth, we know is the region where the actual polyp can be found. Obviously, during a real analysis, the ground truth would not be available, but the uncertainty

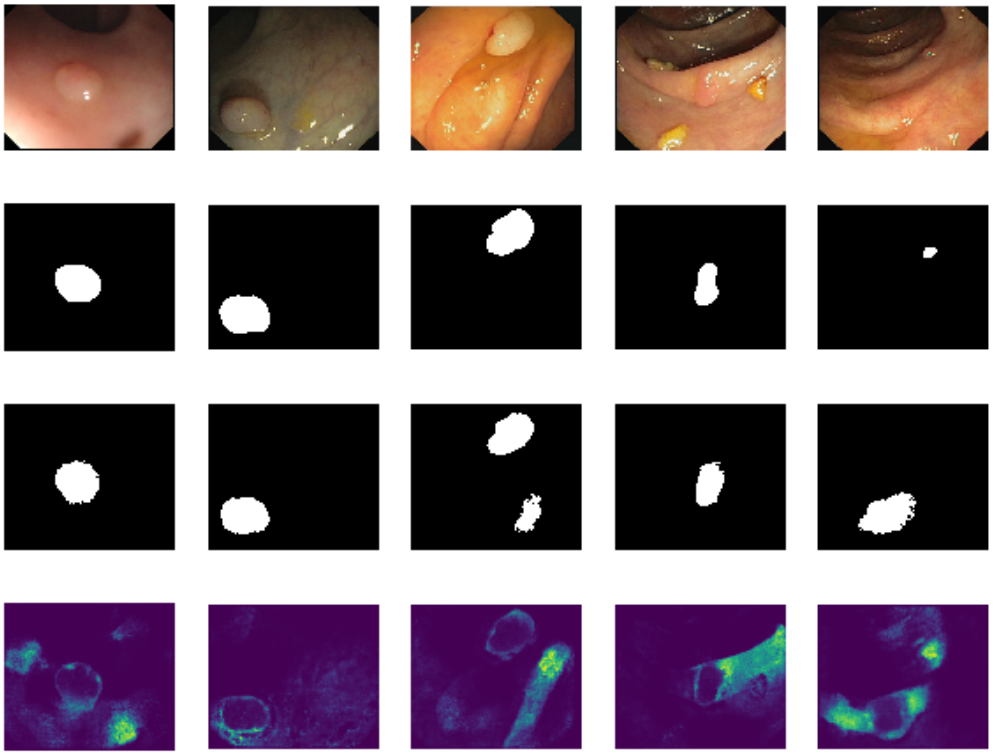


FIGURE 5.8: Figure displays ESegNet's predictions and the uncertainty map associated with the predictions. Each row represents, from top to bottom, original image, ground truth, prediction and uncertainty map. For the uncertainty maps, dark blue pixels are associated with low uncertainty and bright green pixels are associated with high uncertainty.

map would give a clear indication to the analyst that this is an image that requires further examination and is more ambivalent than the examples shown in column 1 to 4.

Next, we consider Figure 5.8, which presents the result of the uncertainty estimation for ESegNet's predictions. In column 1 and 2, ESegNet successfully segments the polyp, but the uncertainty maps are different between the two predictions. For column 2, the uncertainty map shows a degree of uncertainty about the border of the polyp. Column 1 on the other hand, shows the same behaviour for the region containing the actual polyp, but there are other regions in the image also associated with high uncertainty. In column 3 of Figure 5.8, ESegNet finds the polyp but also segments part of the colon as a polyp. From the corresponding uncertainty map we can see that the uncertainty associated with the correctly classified polyp is similar to that of column 2, but there is also a high uncertainty associated with a ridge going along the colon. Column 4 shows similar behaviour as the result from column 3, but the uncertainty is

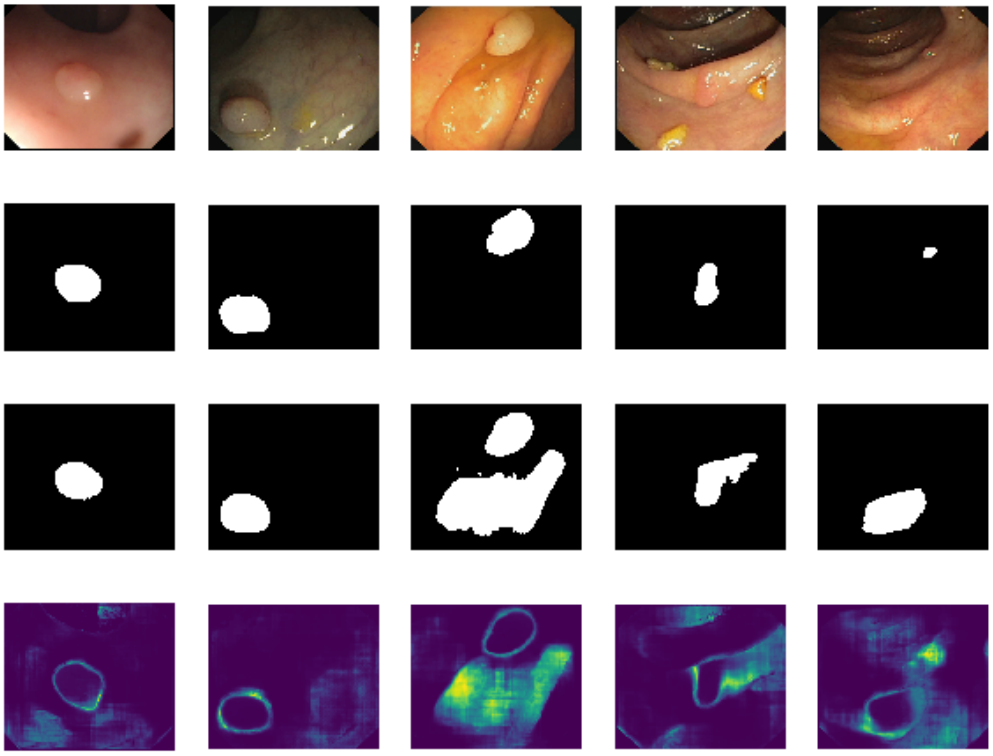


FIGURE 5.9: Figure displays EU-Net's predictions and the uncertainty map associated with the predictions. Each row represents, from top to bottom, original image, ground truth, prediction and uncertainty map. For the uncertainty maps, dark blue pixels are associated with low uncertainty and bright green pixels are associated with high uncertainty.

higher, so the model avoids classifying parts of the colon as a polyp. In the last column of Figure 5.8 the model miss the polyp completely and segment part of the colon as polyp. The uncertainty map shows two regions with elevated uncertainty, one where the actual polyp is and one where the model has predicted a polyp. As the uncertainty in the latter region is lower than the region corresponding to the true polyp, the model makes a mistake and segments the wrong region.

Finally, we consider Figure 5.9, which presents the result of the uncertainty estimation for EU-Net's predictions. For the first two column where EU-Net successfully segments the polyp, the uncertainty maps show relatively low uncertainty except for the boundaries of the actual polyp. In column 3, EU-Net segments the actual polyp but also segments a large region of the colon itself. But the uncertainty map shows us that these two regions are considered quite differently by the model, where the region towards the top of the region containing the actual polyp exhibits a similar appearance as the results from column 1 and 2 and the large region towards the bottom of the

image shows a high uncertainty associated with the entire segmentation, prompting suspicion regarding the segmentation towards the bottom of the image. Column 4 shows the example where EU-Net segments the polyp but also attaches part of the colon to the prediction. From the uncertainty map, we can see that the model is concerned about the entire ridge extending to the right from the polyp but slightly less concerned about a small region which is why it is included in the prediction. For the final column of Figure 5.9, where the network completely misses the polyp, we see that large parts of the image are associated with high uncertainty. Although, the region containing the prediction of the model does look similar to the results from column 1 and 2, indicating that the model is somewhat certain about its wrong prediction.

Comparing the results of each model, we can see that column 1 and 2 look similar between all three figures but with ESegNet showing a slight concern about the first column. This seems to support the results presented in Section 5.2, namely that polyps with an elliptical shape and distinct boundaries are well understood by the model. In column three we show an example where there are multiple ridges along the colon and the polyp exhibits an elongated shape. While EFCN-8 locates the polyp with a high certainty, both ESegNet and EU-Net are uncertain about the ridges contained in the image, with EU-Net displaying a higher degree of uncertainty than ESegNet. This seems to suggest that the increased complexity of the EFCN-8 has enabled a better understanding of the difference between edges associated with polyp and edges associated with the colon itself. We see similar behaviour is column 4, again with EFCN-8 obtaining the highest certainty about its prediction while ESegNet and EU-Net have larger regions of uncertainty. For the fifth column, we saw in the previous examples that all models made the same mistake. By consulting the uncertainty maps, we can now see that all models are equally uncertain about the small region where the actual polyp is located, suggesting that it is a region that deserves further analysis. Also, by considering the uncertainty map corresponding to the region where all models falsely predicted a polyp we notice another difference between the three models. While ESegNet and EU-Net show similar uncertainty as in their successful predictions displayed in column 1 and 2, that is, uncertainty about the exact border but not about the polyp itself, EFCN-8 shows that it is uncertain about the entire prediction. This suggests that EFCN-8's prediction for the original image in column 5 should not be considered as trustworthy as the ones in the remaining 4 columns.

In Section 5.2 we argued that EFCN-8 obtained higher quantitative results and gave

more precise predictions, but many questions were still unanswered. If EFCN-8, ESegNet, and EU-Net gave the same prediction, can we trust them equally? If they gave competing predictions, who should we trust? Using the uncertainty maps obtained from this experiment we have gained insight into how the models compare and that predictions, even if they look similar, can have different uncertainty. Making a definitive statement about the difference between EFCN-8 versus ESegNet and EU-Net can be difficult, but the two initial sections of this chapter might lead to the belief that the superior complexity of the EFCN-8 is proving advantageous. Of course, there might be other factors, such as the upsampling procedure. However, EU-Net and EFCN-8 both employ transposed convolutions for the upsampling. If the upsampling procedure was causing the improved performance and reliability, we would expect a difference between ESegNet and EU-Net, which have a similar number of parameters but different upsampling procedures. Another explanation might be the inclusion of transfer learning for EFCN-8. But we experimented with training ESegNet with and without weights trained on the ImageNet dataset without seeing significant improvements. Nevertheless, stating a definite cause for the difference between the models would be ill-advised and further investigation is encouraged.

To conclude the discussion of these results we would like to mention two interesting aspects that could be investigated further, but we considered outside the scope of this thesis. Firstly, how does the dropout rate affect the uncertainty maps? For example, if the dropout rate is increased and more units are dropped, we might expect that more samples would be required before the mean standard deviations begin to plateau. On the other hand, too low dropout rate would not give enough variation in the network and might result in an underestimate of the uncertainty. For our evaluation, we keep the same dropout rate during inference as we used during training, similar to [75], but determining the correct dropout rate to use during inference is a question that would provide a greater understanding of Monte Carlo Dropout. Secondly, how many samples are necessary to obtain stable uncertainty maps? Too few samples might give a poor uncertainty estimate, while too many samples introduces an unnecessary computational burden. We consider determining the optimal number of samples an interesting direction for future research of Monte Carlo Dropout.

5.4 Determining Importance of Input Features

Our continued analysis was aimed at investigating what features affect the predictions made by the network, utilizing the Guided Backpropagation approach described in Section 3.5.2 of Chapter 3. We look at the same examples displayed in the previous sections of this chapter to obtain a deeper understanding of our models. Determining what input features are affecting the decisions of the models provide greater room for model comparisons and also gives insight into what characteristics in the original image makes a particular example simple or difficult to segment.

5.4.1 Results

Figure 5.10, 5.11 and 5.12 displays the results of the interpretability investigation for EFCN-8, ESegNet and EU-Net, respectively. Each row represents, from top to bottom, original image, ground truth, prediction, uncertainty map and interpretability map. We chose a new color map to represent the interpretability maps such that the difference between the uncertainty maps and interpretability maps became clear. For the interpretability maps, blue pixels are associated with gradients of small magnitude, that is, features with low influence on the prediction of the model while bright teal pixels are associated with features with high influence on the prediction of the model.

5.4.2 Discussion

Results of the interpretability investigation for each model is presented in separate figures, such that each model is given a clear evaluation before we discuss the differences between the models. Once again, we start by examining EFCN-8 and look at Figure 5.10. One initial observation we can make by just glancing over the bottom row of Figure 5.10 is how few pixels are strongly affecting the prediction, which implies that only a few key features are instrumental to the model's prediction. If we consider column 1 of Figure 5.10, it is evident that top edge of the polyp is influencing the model's prediction. Also, the model seems to be unaffected by the bottom edge, which seems to suggest that EFCN-8 is basing its prediction on particular properties. Furthermore, the original image has a large, distinct edge toward the top of the image that is completely disregarded by the model, which enhances the belief that the model

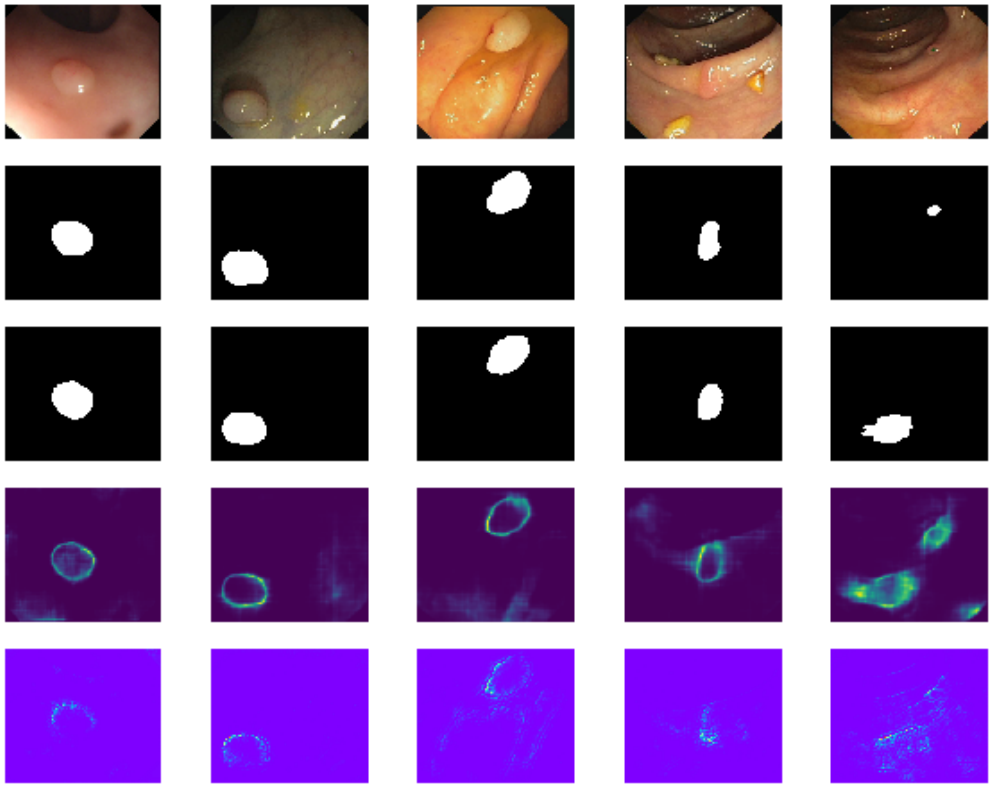


FIGURE 5.10: Figure displays EFCN-8's predictions, the uncertainty map associated with the predictions and the input features the network deems important. Each row represents, from top to bottom, original image, ground truth, prediction, uncertainty map and interpretability map. For the interpretability maps, blue pixels are associated with low influence features and bright teal pixels are associated with high influence features.

is considering a combination of particular features and their surrounding context. Column 2, 3 and 4 also demonstrates that EFCN-8 is affected by a certain kind of edges, where it considers the top edge of the polyp in column 2, the left edge of the polyp in column 3 and the bottom edge of the polyp in column 4. But if we turn to the last column of Figure 5.10 we see a different example. There are more pixels influencing the prediction and the particular form of the colon in this image is misleading the model to a wrong prediction. It is also interesting to see that models are considering pixels associated with the actual polyp in the original image but deems them not influential enough to warrant a prediction. Determining exactly why these pixels are not considered influential enough is difficult, but one hypothesis might be that the scale, brightness or color is discouraging a prediction, or the combined contribution of all those factors is considered insignificant my the model.

Figure 5.11 displays the result of the interpretability investigation for ESegNet. Once

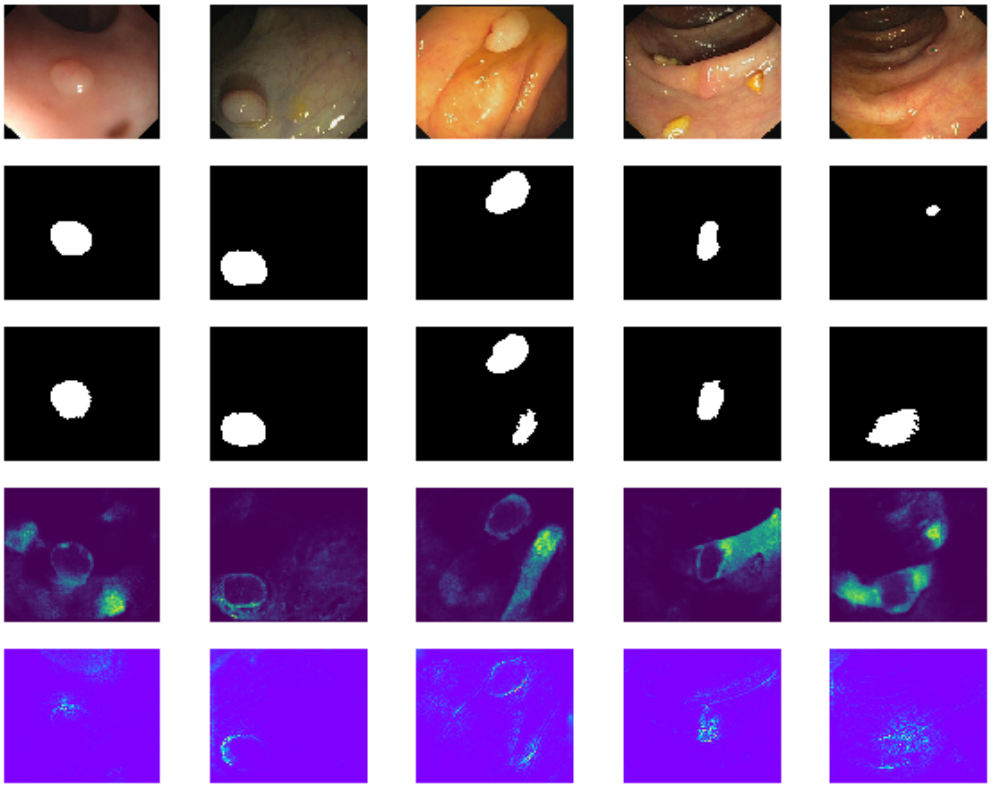


FIGURE 5.11: Figure displays ESegNet's predictions, the uncertainty map associated with the predictions and the input features the network deems important. Each row represents, from top to bottom, original image, ground truth, prediction, uncertainty map and interpretability map. For the interpretability maps, blue pixels are associated with low influence features and bright teal pixels are associated with high influence features.

more, we see that there are few pixels in the input image that strongly influence the prediction of the model. In column 1, only a small portion of the top edge of the polyp is contributing to the prediction, whilst in column 2 it seems that the model is influenced by the left edge of the polyp. Column 3 presents an example where ESegNet falsely segments part of the colon as a polyp, and from the interpretability map, we can observe that the model is influenced by a ridge in the colon. For the correctly segmented polyp in column three, ESegNet is motivated by both top and bottom edge of the true polyp. From the first three column it might seem as tough edge information has the highest influence on the prediction, but column four paints a different picture. Although the model is affected by the edges of the polyp, features associated with the center of the polyp are lit up. One interpretation could be that the model incorporates the slight color difference that, combined with the edge information, prompts a prediction. In the last column of Figure 5.11 we revisit the example where the model misses the actual polyp and falsely classifies part of the colon. The interpretability map shows

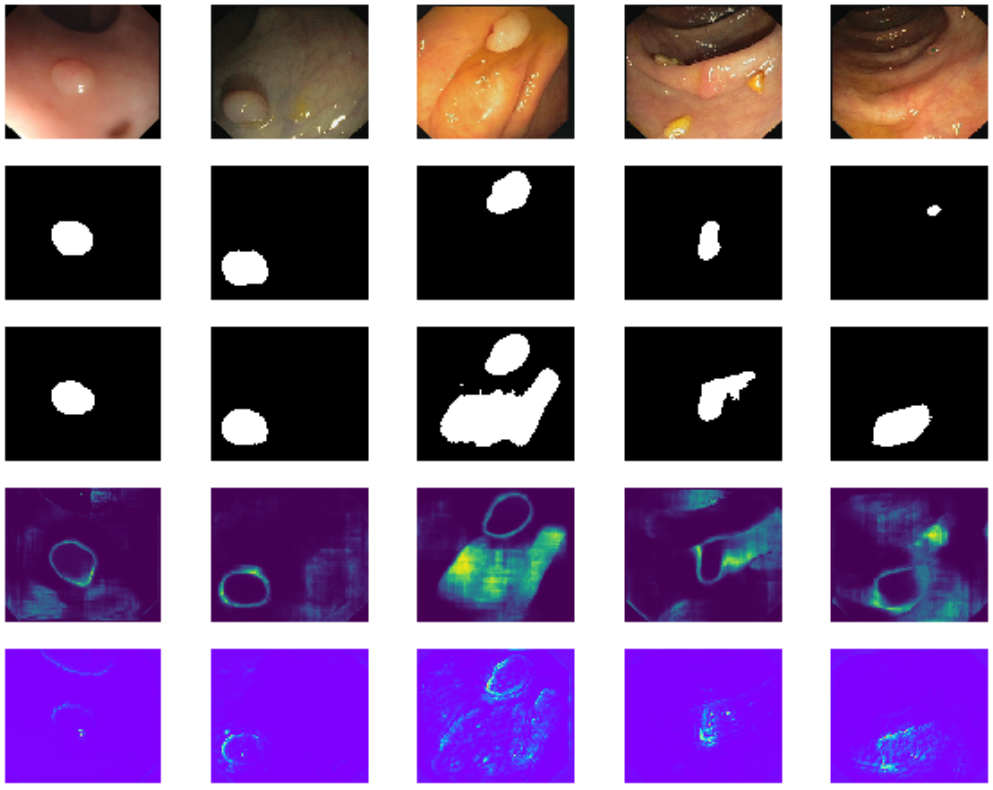


FIGURE 5.12: Figure displays EU-Net's predictions, the uncertainty map associated with the predictions and the input features the network deems important. Each row represents, from top to bottom, original image, ground truth, prediction, uncertainty map and interpretability map. For the interpretability maps, blue pixels are associated with low influence features and bright teal pixels are associated with high influence features.

that the network is mostly affected by pixels around the falsely classified polyp and does not consider the features corresponding to the actual polyp. But the uncertainty map still shows an uncertain region associated with the actual polyp, even though no features in this region is highlighted. This might lead to the belief that the model is recognizing features that are different from the colon, but it has yet to associate them with polyps, probably from the lack of such features in the training set. So these features are not highlighted in the interpretability map, because the network does not deem them important for the polyp class, but they do appear in the uncertainty map since the network is noticing something irregular. Based on the prediction, uncertainty map and interpretability it seems palpable that ESegNet has yet to identify polyps of this scale, shape, and color that could be improved by collecting more images of this character in future data gathering.

Lastly, we examine the interpretability maps constructed for EU-Net displayed in Figure 5.12. For the two successful segmentations in column 1 and 2, they both consider edge information, but the example in column 2 is much more distinct. In column 1 there is only a tiny amount of features that influence the prediction. Seeing that EU-Net successfully segments the entire polyp, it seems that the context around these highlighted features is important to complete the segmentation. In the third column, where EU-Net falsely classifies large parts of the colon as a polyp, it is fairly clear from the interpretability map that the model is confused by the many ridges that go along the colon in the original image. In the example presented in column 4, features corresponding to the edges of the polyp and the polyp itself are highlighted, indicating that a combination of features is considered. We can also see that there are a small number of highlighted features toward the top right of the polyp, and it seems like these features are causing the slight over-segmentation at the top right region of the polyp. The final column shows how EU-Net's false prediction is influenced by features around the wrongful prediction, while features associated with the polyp itself is not considered at all.

Comparing the interpretability maps of all models provide many interesting observations. First of all, for the successful examples shown in the first two columns of Figure 5.10, 5.11 and 5.12 all models consider the distinct edges of the polyp, but EFCN-8 is basing its decision on more features than ESegNet and EU-Net. This is particularly obvious in column 1, where EFCN-8 considers the entire top region of the polyp and ESegNet/EU-Net only considers a tiny part of the edge of the polyp. For column 3 we have the opposite case, where ESegNet and EU-Net are, wrongfully, considering plenty of features while EFCN-8 is able to single out the small number of features that are needed to locate the polyp. Column 4 displays somewhat similar results between all models, but it is interesting to see that they all extract a combination of features to produce a prediction. In the last column, we see similar behaviour for the falsely classified polyp across all networks. But EFCN-8 is the only model that is affected by features associated with the actual polyp in the image. Albeit it is not affected enough to produce a prediction, it does suggest that EFCN-8 has, to a certain degree, been able to acquire a description of polyps that encompass features associated with small, distant polyps.

There is no doubt hat the inclusion of interpretability maps broadens the room for analysis of each model and for model comparisons, and combined with the uncertainty maps we command a greater understanding of these models than we did after "just"

constructing DSSs based on FCNs. Taken by themselves, the interpretability maps can demonstrate what features are well understood by the network and by observing that features evade the models we can identify which type of images are lacking in the training set, providing a clear objective when gathering new data. In combination with uncertainty maps, the interpretability maps can display if the uncertainty is a result of inherent uncertainty in the original image, such as exact location of polyp edges, or a result of unfamiliar features that the network has yet to associate with the polyp class.

5.5 Estimating Uncertainty in Input Feature Importance

In order to obtain an even deeper understanding and enable more room for comparison we estimated the uncertainty of the input features by utilizing the Monte Carlo Gradients method proposed in Section 4.1 of Chapter 4. Once more, we continue our examination on the same examples as presented previously in this chapter.

5.5.1 Results

Figure 5.13, 5.14 and 5.15 displays the results of the uncertainty estimation of the input features for EFCN-8, ESegNet and EU-Net, respectively. Each row represents, from top to bottom, original image, ground truth, prediction, uncertainty map and gradient uncertainty map. The gradient uncertainty maps were computed by drawing 10 samples from the network with a dropout rate of 0.5 and estimating the standard deviation across these 10 samples.

5.5.2 Discussion

Once again, we present the result for each model separately for independent analysis before the results are compared between the models and we start be examining EFCN-8, presented in Figure 5.13. At first glance, the gradient uncertainty maps might seem less distinct than the uncertainty maps, but closer inspection provides some interesting observations. In the first column of Figure 5.13 there are several regions that are highlighted, but notice that features associated with the actual polyp is not. This

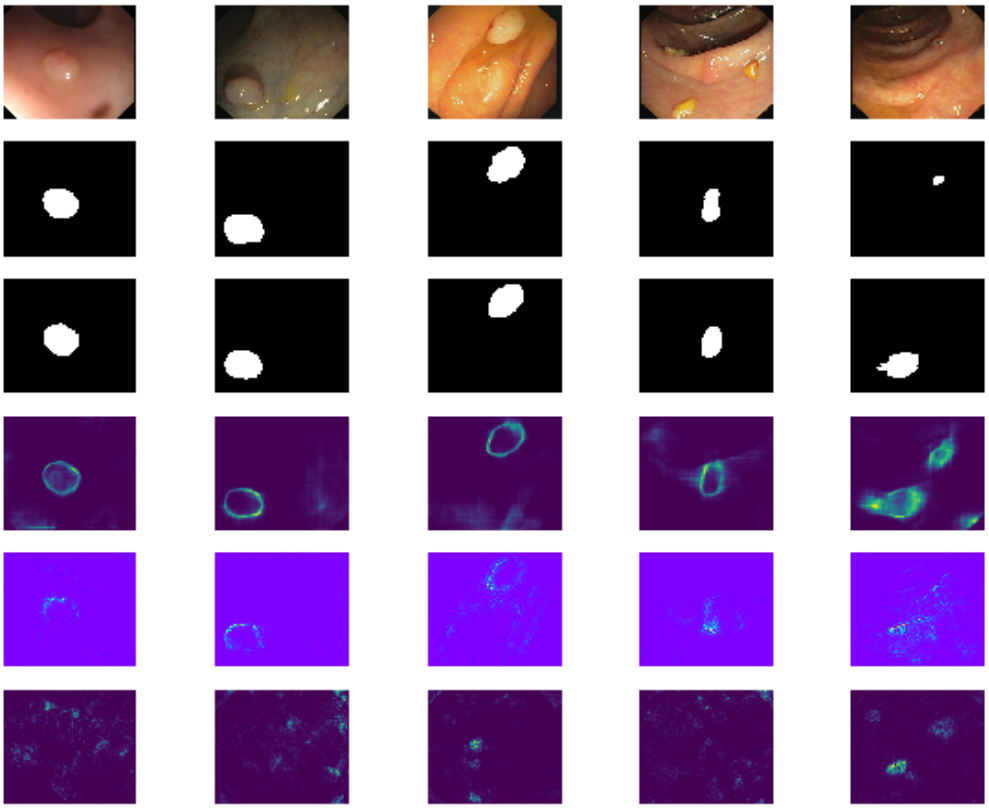


FIGURE 5.13: Figure displays EFCN-8's predictions, the uncertainty map associated with the predictions, the input features the network deems important and the uncertainty map associated with the input features. Each row represents, from top to bottom, original image, ground truth, prediction, uncertainty map, interpretability map and gradient uncertainty map. For the gradient uncertainty maps, dark blue pixels are associated with low uncertainty and bright green pixels are associated with high uncertainty.

indicates that EFCN-8 is consistently considering the same feature each forward pass even when units are dropped. Also, the regions that are highlighted in the gradient uncertainty map are associated with distinct regions of the colon, such as the ridge toward the top of the image or the change of illumination toward the left and right of the image, which seems to suggest that EFCN-8 is examining other features than those shown in the interpretability map, but does not consider them important enough to induce any prediction. We observe similar behaviour in column 2, 3 and 4, where the model is certain about the features shown in the interpretability maps while being uncertain about other features contained in the image. But the last column in Figure 5.13 shows a different story. During our analysis of the interpretability map from the last column, we saw that the model was considering features associated with both the colon and the polyp, but it was difficult to assess why the features related to the

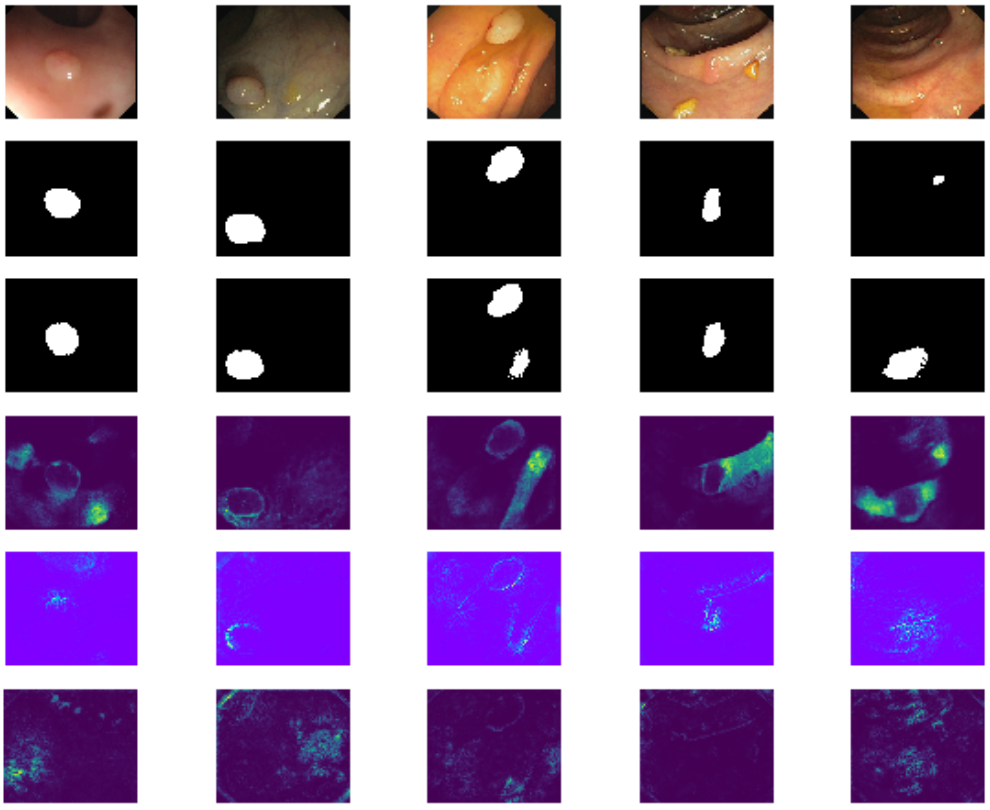


FIGURE 5.14: Figure displays ESegNet's predictions, the uncertainty map associated with the predictions, the input features the network deems important and the uncertainty map associated with the input features. Each row represents, from top to bottom, original image, ground truth, prediction, uncertainty map, interpretability map and gradient uncertainty map. For the gradient uncertainty maps, dark blue pixels are associated with low uncertainty and bright green pixels are associated with high uncertainty.

colon instigated a prediction and not the features associated with the polyp. But the gradient uncertainty maps shows that the features associated with the polyp has a certain degree of uncertainty associated with them, thus preventing a prediction. For the region wrongfully segmented as polyp we see a high degree of uncertainty towards the left of the region, but notice that the right part of region shows low uncertainty which might point to the features that encourage the false prediction.

Figure 5.14 displays the results of the gradient uncertainty estimation for ESegNet and in the last row of column one we can see two highlighted regions that stand out, one located at the bottom left of the image and one at the top of the image. By comparing the original image and the gradient uncertainty map we can see that the region at the bottom left of the image has a shift in illumination that the model reacts to. But seeing that they are not present in the interpretability map it indicates

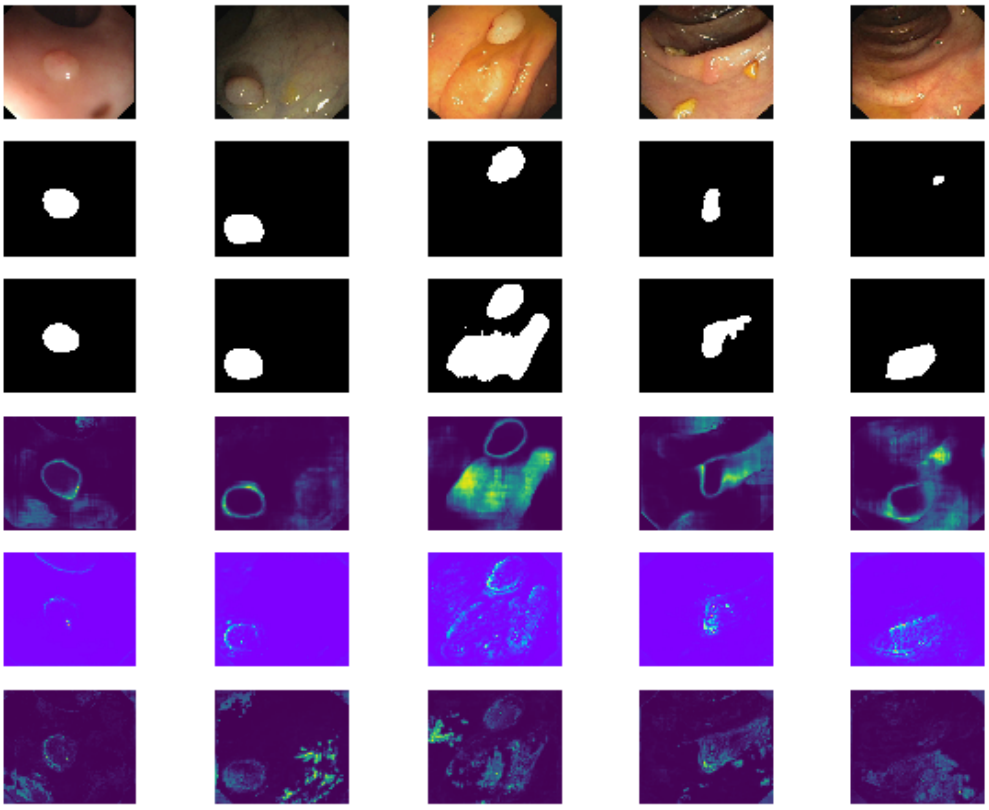


FIGURE 5.15: Figure displays EU-Net's predictions, the uncertainty map associated with the predictions, the input features the network deems important and the uncertainty map associated with the input features. Each row represents, from top to bottom, original image, ground truth, prediction, uncertainty map, interpretability map and gradient uncertainty map. For the gradient uncertainty maps, dark blue pixels are associated with low uncertainty and bright green pixels are associated with high uncertainty.

that the uncertainty associated with these features are too high to foster a prediction. The region at the top of the image corresponds to a ridge in the colon. But notice how the left half of the ridge shows up in the gradient uncertainty map while the right half of the ridge shows up in the interpretability map. It is difficult to say why one part of the ridge has more uncertainty associated with it compared to the other, but it exemplifies that judging the importance of input features solely based on the interpretability maps could be ill advised and consulting the gradient uncertainty map gives valuable information. In column 2 and 3 we see similar results as the first column, with high uncertainty associated with the ridge in the top left corner and change of illumination in the bottom right corner. Column four presents an example that seems to indicate very low uncertainty associated with all of the input features. While that might be the case, one should also be cautious when presented with such example and

this might be an example of a faulty gradient uncertainty map. In the last column of 5.14 we revisit the false polyp prediction. The gradient uncertainty map indicates high levels of uncertainty in several regions of the image and reinforces the view that ESegNet struggles with this particular example.

Figure 5.15 displays the results of the gradient uncertainty estimation for EU-Net. Turning our attention to the first column we notice two regions that stand out in the gradient uncertainty map. Similar to ESegNet, EU-Net notice features in the bottom left corner where illumination changes. But EU-Net also indicates uncertainty associated with features connected to the edges of the actual polyp, suggesting that it has yet to acquire a full understanding of features corresponding to polyps. In column 2 and 3 we also observe uncertainty connected with polyp features, but also bright green flecks of uncertainty that seem to correspond to features associated with glare in the original image. Column 4 displays uncertainty with the entire ridge that expands outward from the polyp in the center of the image, which is likely to be caused by the edges and glare present along the ridge. In the last column, we observe that the region toward the bottom of the image surrounding the falsely predicted polyp is related to some degree of uncertainty. Yet, the region is not present in neither the interpretability map or the gradient uncertainty map, indicating that EU-Net is unaware of any noteworthy features in that region of the image.

When comparing the gradient uncertainty maps of all models one aspect that stands out is the difference between EFCN-8 and ESegNet versus EU-Net. In particular, both EFCN-8 and ESegNet has little or no uncertainty about the features corresponding to an actual polyp, indicating that even when we sample different weights the model is still focusing on the same features in each forward pass. EU-Net, on the other hand, seems uncertain about several gradients, which might indicate that there is a need for further adjustment of EU-Net's weights. From Figure 5.2 we know that EU-Net's training was ended by the early stopping procedure earlier than EFCN-8 and ESegNet. During the design phase, we trained all network several times to observe their behaviour, and EU-Net consistently converged earlier than the two others. But judging by the gradient uncertainty maps it seems as though it could have benefited from further training to obtain a better understanding of the input features and suggest that we could have tuned the patience hyperparameter more specifically for each network. Of course, it is difficult to know for certain what exactly causes the difference between the models, but including gradient uncertainty does allow for more thorough analysis of models and greater room to observe differences between models.

5.6 Towards Understanding FCNs Through Information Theory

To investigate the ITL framework for FCNs described in Section 4.2 of Chapter 4 we utilize ESegNet to perform our analysis. We consider the input, output and the three central encoders and decoders of ESegNet as random variables, denoted as X, Y, E_3, E_4, E_5, D_1, D_2 and D_3. Our reasoning for employing ESegNet will be clarified in the discussion of this section. We consider 100 samples from the test set of the Endoscene dataset to estimate the mutual information between the aforementioned random variables.

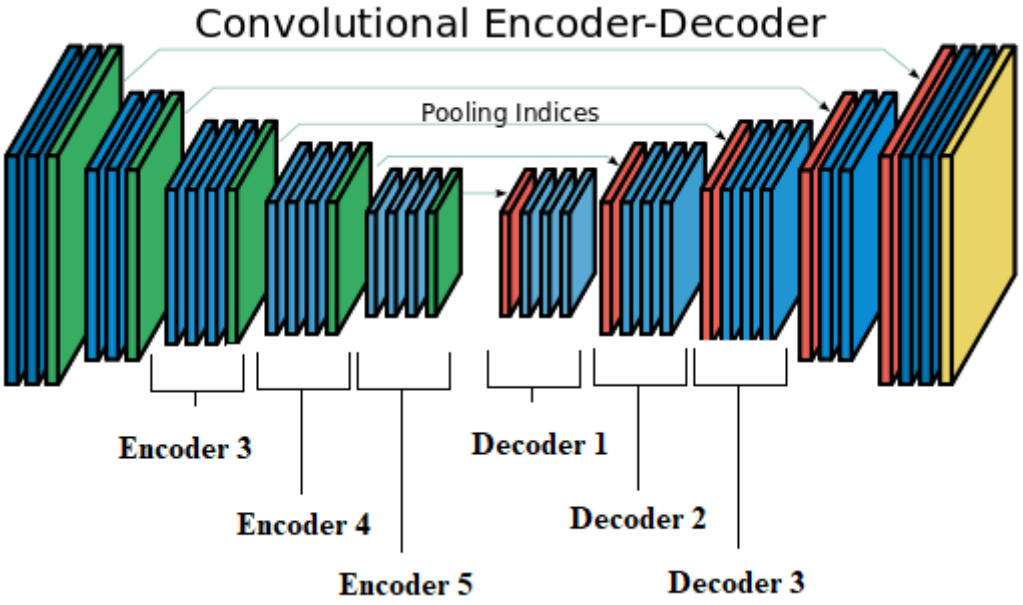


FIGURE 5.16: Figure displays the ESegNet architecture from [97] with the three central encoder and decoders labeled.

5.6.1 Results

In Table 5.2 we present the results of our mutual information investigation, where each cell of the table represent the mutual information between the two random variables in the corresponding row and column. We select $\alpha = 2$ following the example of [83] and determine the kernel width for each random variable using the heuristic method of [95], where the different kernel widths are presented in Table 5.3.

	X	Y	E_3	E_4	E_5	D_1	D_2	D_3
X	4.130	3.815	3.311	3.223	3.230	2.431	2.334	2.706
Y	3.815	3.921	2.837	2.934	3.336	2.730	2.652	2.948
E_3	3.311	2.837	2.964	2.712	2.304	1.518	1.417	1.762
E_4	3.223	2.934	2.712	2.585	2.431	1.755	1.674	1.971
E_5	3.230	3.336	2.304	2.431	2.917	2.479	2.433	2.643
D_1	2.431	2.730	1.518	1.75	2.479	2.462	2.449	2.474
D_2	2.334	2.652	1.417	1.674	2.433	2.449	2.443	2.454
D_3	2.706	2.948	1.762	1.971	2.643	2.474	2.454	2.554

TABLE 5.2: Results of the mutual information analysis for the three central encoders and decoders of ESegNet with the input and output. Each cell displays the mutual information between the variables denoted in the row and column of the cell in question, where mutual information has been calculated using Equation 4.14.

	X	Y	E_3	E_4	E_5	D_1	D_2	D_3
σ	63.808	34.741	16.956	23.561	11.630	14.827	18.858	18.334

TABLE 5.3: Kernel widths used to estimate the entropy the input, output and three central encoders and decoder of ESegNet.

5.6.2 Discussion

Before we proceed with our discussion of the results we would like to stress that our investigation of FCNs through an ITL framework is in its infancy and should be considered preliminary work towards an ITL framework for FCNs. Seeing as the initial works on an ITL framework for DNNs have been developed in the last couple of months [81, 84] we were somewhat constrained by time. Yet, we considered such an approach a natural inclusion in our work towards understanding DNNs as it provides more quantitative results compared to the more visual results presented up until this point, thus broadening our analysis even further.

We intend to abstain from making bold claims based on the results in Table 5.2, but we would like to direct the reader's attention to some interesting aspects of the analysis. From row one of Table 5.2 we can see that the mutual information between X and the three central encoders, E_3, E_4 and E_5, is quite similar, which begs the question; If the three central encoders explain the same information about X, do we need all three? But if we turn to the second row of Table 5.2, we encounter a different story. Notice that the mutual information between Y and the three central encoders is rising significantly from E_3 to E_5, which can indicate that the encoders are molding the features into a representation that is suitable for predictions.

Regarding the DPI discussed in Chapter 4, we can see that $I(X, D_2) \leq I(X, D_3)$, which seems to contradict the DPI, most likely a result of the skip-connections of the model. Also, note that the mutual information between Y and the three central encoders keeps rising as we move towards the decoder network, but that the mutual information drops in the first decoder. This might imply that it is difficult to retain information during the upsampling procedure.

Lastly, we would like to point out the curious behaviour of the second decoder, D_2. By inspecting the three last columns of Table 5.2 it is visible that the mutual information drops for all variables in the second decoder before rising again in the third decoder. Could this be signaling that the second decoder is in need of more training?

Our choice of ESegNet as the architecture for performing our analysis was mostly a practical decision. Seeing as our implementation of ESegNet included dropout in the three central encoders and decoders we had an idea of utilizing dropout to sample from the three central encoders and decoders that could enable some form of uncertainty estimation of the mutual information. But as time progressed we came to the conclusion that this was considered outside the scope of the thesis. Of course, optimally we should have analyzed all three architectures using the ITL framework, but time considerations limited us to only considering ESegNet.

Determining the width of the free parameter σ is an important part of kernel methods. If the kernel width is too small we might not capture the variance of data, but if the kernel width is too large we might be unable to distinguish important features of the data from noisy or unimportant features of the data. We experimented with Silverman's rule of thumb with $h = 5$, following the example of [81], which suggested a kernel width equal to 4.777. But we experienced that using such a kernel width resulted in the mutual information between input, output and the encoders and decoders were essential equal. This seems odd, as one would expect some information to be lost during the forward pass, for instance after the pooling layers. Therefore, we utilized the heuristic presented in [95] where we consider the mean of the five nearest neighbours for each sample, which resulted in the values displayed in Table 5.3.

Chapter 6

Discussion and Conclusion

In order to tie together all the results and discussions presented in the previous chapter, we would like to give a more overarching discussion regarding the potential of DSSs based on FCNs and the effect of including techniques that aim at providing a better understanding of such models. This chapter will also discuss different aspects of the thesis that could offer interesting ideas for future research, both in the field of semantic segmentation of colorectal polyps and in the field of deep learning. Additionally, we provide some concluding remarks to round off this thesis.

6.1 Conclusion

In this thesis, we have developed and evaluated several FCNs for the task of semantic segmentation of colorectal polyps. Also, to address the lack of interpretability, we introduced, developed and evaluated a number of different methods that seek to tackle this issue. Our results display that FCNs are capable of high performance, where our best model achieved state-of-the-art performance on the Endoscene dataset. However, there is still room for improvement, and DSSs based on FCNs can benefit from further development.

Our results also demonstrate that model understanding can be significantly improved by utilizing recent advances in DNN interpretability techniques. Utilizing Monte Carlo Dropout and Guided Backpropagation, we were able to estimate the uncertainty in each prediction and investigate what features each model deems important, both of

which have, to the best of our knowledge, yet to be investigated in the context of semantic segmentation of colorectal polyps prior to this work.

In Chapter 4 we introduced two novel methods aimed to increase the interpretability of DNNs, which we successfully demonstrated on the task of semantic segmentation of colorectal polyps. Our first method, which we refer to as Monte Carlo Gradients, enabled estimation of uncertainty in the input features that were deemed important by the Guided Backpropagation procedure. Such uncertainty estimation has, to the best of our knowledge, not been performed in the context of semantic segmentation of colorectal polyps nor in the deep learning field as a whole. Our second novelty was to develop a framework for analyzing FCNs using ITL concepts, also a procedure that have yet to be performed in the context of semantic segmentation of colorectal polyps nor the deep learning field as a whole prior to this work. We would also like to add that although we demonstrated these methods on polyp segmentation, they are applicable to any tasks where DNNs employed.

Lastly, we believe a DSS based on a FCN combined with uncertainty, interpretability and visualization techniques can become capable of providing precise and trustworthy prediction that can aid physicians. But in order for that to happen, the amount of available data must be significantly increased, both to provide more training data that produce better models but also to obtain larger test sets that foster trust.

6.2 Discussion

6.2.1 Potential of FCNs as DSSs

Perhaps the most important property a DSS must possess is consistently high precision and few false negatives. In the worst case scenario, a detected or missed polyp might spell the difference between being able to successfully treat a patient or not. As mentioned in Chapter 1, physicians miss approximately 8-37% depending on the size and type of polyp [7]. Surely, if DSSs are to be valuable they would need to be precise enough that physicians consider them a helpful tool. From Table 5.1 in the previous chapter it should be evident that DSSs based on FCNs have substantially increased precision compared to previous models based on non-deep methods. But, if we consider the IoU score for the polyp class, which provide the most accurate measure

to a model's polyp localization capabilities, it is not as high as we might desire. Also, when we encounter examples such as those shown in the last column of Figure 5.5 and Figure 5.6 in the previous chapter it is clear that there is still room for improvement.

Despite the positive results , we argue that there are several options for improvements. As we have focused more on model interpretability, there is certainly room for improvement concerning hyperparameter adjustments such as the dropout rate, batch size, and patience parameter. However, it is unlikely that it would yield a dramatic improvement in performance. Another modification that might improve the models is utilizing more sophisticated activation functions. For instance, PReLU can sometimes improve performance [53], but usually only by a couple of percentages. Another approach that has shown to increase performance in FCNs is to include a post-processing step using Conditional Random Fields (CRFs) [103], which might be an interesting addition for future research. Additionally, we could have employed even more recent architectures of FCNs such as the Fully Convolutional DenseNets (FCDNs) [72], which have shown impressing results on several segmentation tasks [72, 75]. Another promising architecture utilizes dilated convolutions in their network [104]. Furthermore, since the images are extracted from video sequences taken during colonoscopy, there might be temporal information that would be valuable to obtain. Combing a CNN with a Recurrent Neural Network (RNN), a particular kind of network designed to handle temporal information, such as in [105] is surely an interesting path to consider.

For all that, what would be most likely to improve all models is the inclusion of more training data. The Endoscene dataset consists of 912 images which are a reasonable amount but compared to large datasets like ImageNet it is still not as large as it ought to be. A large dataset would also allow for a larger test set that would give a better indication of how precise a model truly is. However, creating large-scale datasets is in itself a difficult and time-consuming procedure, especially when medical data is concerned. Sharing images in the medical domain can be difficult, seeing as they contain private information about patients that must be protected. Also, creating the annotated image requires a considerable time investment from trained physicians with an already pressured time-schedule. But if a truly large scale dataset with images from thousands of patients included was created, it would most likely provide a significant improvement for DSSs based on FCNs.

As a part of an ongoing collaboration between the UiT Machine Learning Group and the Department of Gastrointestinal Surgery at the University Hospital of North Norway

we are currently working on acquiring more colonoscopy related data for that hospital. Testing our models on that data obtain from a completely different source could yield valuable information with regards to different equipment and new patients. Closer cooperation with medical practitioners will also allow for more input from physicians with regards to how the techniques and methods we have discussed throughout this thesis can be helpful for them. These are aspects, which we have considered outside the scope of this thesis, we aim to investigate and incorporate in our journal manuscript mentioned in Section 1.7 of Chapter 1.

6.2.2 Understanding DNNs

High precision is not the only component a DSS must inhabit. As we have argued throughout this thesis, being able to interpret the predictions of a model is important for a number of different reasons. For network designers, it enables model analysis and model comparison. For physicians, it provides the necessary information to trust the system. Also, a DSSs can potentially reveal concealed patterns in the data, which might provide new perspectives and additional info for the diagnosis. Interpretation of DNNs can often be difficult and they are often referred to as "black boxes" [76]. Throughout Chapter 5 we have developed and evaluated a number of techniques that seek to increase our understanding of FCNs, but what insights can be gained from our analysis?

Most of the techniques we have evaluated provide visual results that try to explain particular aspects of the networks. The uncertainty maps proposed in [74] and extended in this thesis to a medical context by us, provide a tool to determine the uncertainty in a prediction. When highly uncertain cases are encountered we can present the original image to a medical expert for further assessment. Furthermore, it makes room for comparison between seemingly similar predictions, like the examples displayed in the first column of Figure 5.7, 5.8 and 5.9. Using Guided Backpropagation, proposed in [78] and developed for medical analysis in this thesis, we can inspect what features in the input are affecting the prediction of the model, thus allowing physicians to inspect if the model is considering features in the original image that actually correspond to a polyp. Also, designers can use the interpretability maps to see what kind of examples a model is struggling with and adjust accordingly. In Chapter 4 we proposed a new method called Monte Carlo Gradients, inspired by [74] and [78], which provide information regarding the uncertainty of the important features in the original image.

From our results in Chapter 5, we argue that it is evident such techniques provide a richer understanding of both models and the predictions they produce. But we would also like to point out that there is a number of limitations to such techniques and that there are still aspects that are unclear. In particular, visual results are appealing as they can be manually inspected but they are open to interpretation, which means that different people might interpret the results in different ways. Nevertheless, just the fact that such techniques allow interpretation and discussion is a step forward for DNNs and should undoubtedly be considered assets in the field of medical image analysis as well as the deep learning field as a whole.

In contrast, using ITL to analyze DNNs provide quantitative results that might not be suitable for visual inspection, but provide concrete numbers which are useful for a quantitative assessment of the performance. Once again we would like to stress that our work is the very early stages of development and one must be careful not to jump to conclusions. Despite our work being preliminary in several aspects, we believe that an ITL framework opens up some interesting pathways. For instance, one could incorporate mutual information between the layers directly into the cost function as a regularization technique. Another idea might be to inspect a trained network for redundant layers that can be removed for computational benefits [106]. Lastly, ITL could be used to examine the information of layers as they become wider or the information of a network as it grows deeper, which might open a window towards a more theoretical framework for DNN architecture analysis. Again, ITL for DNNs is still in its embryonic phase, but offer many interesting paths for future research.

Appendix A

Appendix Chapter 2

This appendix provide a more thorough explanation of cost functions and includes some details regarding SVMs

A.1 Cost Function

Deciding which cost function to use is dependent on the specific task, and a suitably chosen cost function can often lead to better results. A common choice is the aforementioned sum of squared errors from Equation 3.3, but to interpret the output easier, we often take the mean of Equation 3.3 resulting in the Mean Squared Error (MSE), which is

$$\mathcal{C}_{MSE} = \frac{1}{2N} \sum_{i=1}^{N} \sum_{m=1}^{k_l} (y_m(i) - \hat{y}_m(i))^2 \tag{A.1}$$

where N is the number of samples, k_l is the number of output neurons, $y_m(i)$ is the desired output and $\hat{y}_m(i)$ is the predicted output. Even though the MSE often gives satisfactory results for most tasks, it does have some drawbacks. Since all errors are squared and summed, large errors can have a disproportional effect on the learning process, essentially making the network vulnerable to outliers.

For the task of classification, where the desired outputs are binary, a widely used cost function shown to improve results is the *cross-entropy* cost function, defined by

$$\mathcal{C}_{CE} = -\sum_{i=1}^{N}\sum_{k=1}^{k_L}(y_k(i)\ln(\hat{y}_k(i)) + (1-y_k(i))\ln(1-\hat{y}_k(i)). \tag{A.2}$$

The minimal value of $\mathcal{C}_{CE}$ occurs when a sample is classified correctly, that is, $y_k(i) = \hat{y}_k(i)$. There are various interpretations of the cross-entropy cost function, but a common understanding is that it measures surprise [107]. If the predicted output is close to the desired output, we are not very "surprised" and the cost is low. If the predicted output is far from the desired output, we are very "surprised".

The cross-entropy cost function has several desirable properties. It depends on the relative errors and not the absolute error like $\mathcal{C}_{MSE}$, thus it gives the same weight to small and large values [108]. Also, it diverges if the outputs tend toward the wrong class which produces stronger gradients that speed up learning [108].

For classification of unbalanced datasets, that is datasets where one or several classes occur in a significantly higher number than other classes, it can be beneficial to modify the cost function to account for the imbalance. One such modification which has shown to improve results is Median Frequency Balancing (MFB) [99, 102], defined by

$$\mathcal{C}_{MFB} = -\sum_{i=1}^{N}\sum_{k=1}^{k_L}(y_k(i)\ln(\hat{y}_k(i)) + (1-y_k(i))\ln(1-\hat{y}_k(i))W_k \tag{A.3}$$

where

$$W_k = \frac{\text{median}\Big(f_k | k \in K\Big)}{f_k}, \tag{A.4}$$

f_k is the frequency of samples in class k and K is the set of all classes. If the discrepancy between classes is large enough the network might ignore the underrepresented class altogether and still obtain good overall performance. MFB weights the cost by the ratio of the median class frequency and the actual class frequency, such that the significance of errors corresponding to the underrepresented class is increased.

A.2 SVM details

Further details regarding the KKT conditions presented during the SVM description.

A.2.1 KKT conditions

In Chapter 2 we stated that the minimzer of Equation 2.4 and 2.5 must satisfy the Karush-Kuhn-Tucker(KKT) conditions. If $\boldsymbol{\theta}_*$ is a point that satisfies the regularity condition, then there exits a vector $\boldsymbol{\lambda}$ of Lagrange multipliers so that the following are valid:

1. $\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_*, \boldsymbol{\lambda}) = \mathbf{0}$
2. $\lambda_i \geq 0, \qquad i = 1, 2, ..., m$
3. $\lambda_i f_i(\boldsymbol{\theta}_*) \qquad i = 1, 2, ..., m$

The first condition states that the minimum must be a stationary point of the Lagrangian, with respect to $\boldsymbol{\theta}$. The second is states that the Lagrange multipliers are nonnegative. The third condition is known as the complementary slackness condition that state that at least one of the terms in the products is zero.

Appendix B

Appendix Chapter 3

This appendix describes the gradient descent algorithm and its many variants.

B.1 Optimization techniques

All neural networks share the goal of optimizing some cost function with respect to some parameters. Although there has been some experimentation with using Newton's method [109] for optimizing neural networks, the most widely used algorithm is gradient descent. This is because gradient descent only requires computing the gradients of a network which can be very efficient compared to methods that require higher order derivatives to be computed.

Gradient Descent Algorithms

As mentioned in Section 3.1.2, the most common optimization algorithm for training neural networks is gradient decent where parameters are updated according to Equations 3.4 and 3.5. In the general case gradient decent is given by

$$\boldsymbol{\theta}(\text{new}) = \boldsymbol{\theta}(\text{old}) - \mu \nabla_{\boldsymbol{\theta}} \mathcal{C}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) \tag{B.1}$$

where $\boldsymbol{\theta}$ are the parameters we want to update, μ is the learning rate and $\nabla\mathcal{C}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y})$ is the derivative of the cost function w.r.t $\boldsymbol{\theta}$ given some training data pairs $\{\mathbf{x}, \mathbf{y}\}$. Gradient decent is based on the simple idea of adjusting the parameters in the opposite direction of the function we want to minimize and the size of the adjustment in determined by the learning rate. However, standard gradient decent requires computing the gradient of the entire training dataset which can be computationally demanding. Therefore, we usually consider a batch of samples, such that Equation B.1 becomes

$$\boldsymbol{\theta}(\text{new}) = \boldsymbol{\theta}(\text{old}) - \mu \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{C}(\boldsymbol{\theta}; \mathbf{x}_i, \mathbf{y}_i), \tag{B.2}$$

where N is the number of samples in the batch. We refer to this algorithm as batch-gradient descent or Stochastic Gradient Descent (SGD). SGD is computationally less demanding but provide a more coarse estimate of the gradient. However, this stochasticity can have a regularizing affect, as it might prohibit the model from finding a set of parameters fitted to the training set.

A common issue for gradient descent algorithms are regions where the cost plateaus before descending further, which lead to gradients close to zero and thus no parameter updates. A typical solution is adding momentum [110] which accelerates the algorithm in the relevant direction. Momentum is included by adding a fraction γ of the gradients of the previous time step, expressed as:

$$v_t = \gamma v_{t-1} + \mu \nabla_{\theta} \mathcal{C}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) \tag{B.3}$$

$$\boldsymbol{\theta}(\text{new}) = \boldsymbol{\theta}(\text{old}) - v_t. \tag{B.4}$$

Momentum is often illustrated as a ball rolling down a hill which can traverse flat region as a result of the momentum it gathers while rolling down the hill. However, a ball rolling blindly down a hill might overshoot a desired minimum, so to give the ball a sense of direction one could employ a variation of momentum known as Nesterov Momentum [111]. Nesterov Momentum considers $\boldsymbol{\theta}(\text{old}) - \gamma v_{t-1}$, thus approximating the next position of the parameters. We can implement this procedure by

$$v_t = \gamma v_{t-1} + \mu \nabla_\theta \mathcal{C}(\boldsymbol{\theta} - \gamma v_{t-1}; \mathbf{x}, \mathbf{y}) \tag{B.5}$$

$$\boldsymbol{\theta}(\text{new}) = \boldsymbol{\theta}(\text{old}) - v_t. \tag{B.6}$$

There are a number of recent variations of gradient descent which seek to improve the optimization procedure, such as Adagrad [112], AdaDelta [113], and RMSprop [101]. In this thesis we will utilize a recently proposed algorithm known as the Adaptive Moment Estimation (ADAM) algorithm [100]. ADAM computes an adaptive learning rate for each parameter by storing an exponentially decaying average of past gradients and past squared gradients, defined as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta \mathcal{C}(\boldsymbol{\theta} - \gamma v_{t-1}; \mathbf{x}, \mathbf{y}) \tag{B.7}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \nabla_\theta \mathcal{C}^2(\boldsymbol{\theta} - \gamma v_{t-1}; \mathbf{x}, \mathbf{y}) \tag{B.8}$$

where m_t is an estimate of the mean of the gradients, v_t is an estimate of the variance of the gradients, β_1 is the decay rate of the estimated mean of the gradients, and β_2 is the decay rate of the estimated variance of the gradients. The authors of ADAM noticed that since m_t and v_t are initialized as vectors of zeros they are biased towards zero. Therefore, they computed bias corrected estimates

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{B.9}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{B.10}$$

which they used update the parameters, in the following way:

$$\boldsymbol{\theta}(\text{new}) = \boldsymbol{\theta}(\text{old}) - \frac{\mu}{\sqrt{\hat{v} + \epsilon}} \hat{m}_t. \tag{B.11}$$

Because ADAM adjusts $\hat{m}_t$ and $\hat{v}_t$ automatically during the training we do not need to tune these hyperparameters manually, which can be a time-consuming a difficult

process.

Appendix C

Appendix Chapter 4

This appendix provides a detailed description of the three networks proposed in this thesis.

C.1 Network Details

There are many small details one needs to consider when constructing DNNs. In order to provide the greatest degree of clarity, we provide a detailed description of each architecture we have used in this thesis.

C.1.1 FCN-8

Table C.1 displays details regarding our implementation of the FCN-8 from [9].

C.1.2 U-Net

Table C.2 displays details regarding our implementation of U-Net from [71].

C.1.3 SegNet

Table C.3 displays details regarding our implementation of the SegNet from [97].

Layer	(C, H, W)	Kernel Size	Stride	Padding
Conv1	(3, 224, 224)	3×3	1	1
Conv2	(64, 224, 224)	3×3	1	1
Pool1	(64, 224, 224)	2×2	2	0
Conv3	(64, 112, 112)	3×3	1	1
Conv4	(128, 112, 112)	3×3	1	1
Pool2	(128, 112, 112)	2×2	2	0
Conv5	(128, 56, 56)	3×3	1	1
Conv6	(256, 56, 56)	3×3	1	1
Conv7	(256, 56, 56)	3×3	1	1
Pool3	(256, 56, 56)	2×2	2	0
Conv8	(256, 28, 28)	3×3	1	1
Conv9	(512, 28, 28)	3×3	1	1
Conv10	(512, 28, 28)	3×3	1	1
Pool4	(512, 28, 28)	2×2	2	0
Conv11	(512, 14, 14)	3×3	1	1
Conv12	(512, 14, 14)	3×3	1	1
Conv13	(512, 14, 14)	3×3	1	1
Pool5	(512, 14, 14)	2×2	2	0
Conv14(d)	(512, 7, 7)	7×7	1	3
Conv15(d)	(4096, 7, 7)	1×1	1	0
Conv16	(4096, 7, 7)	1×1	1	0
UpConv1	(c, 7, 7)	4×4	2	1
SkipConv1	(512, 14, 14)	1×1	1	0
SkipConv1 + UpConv1	(c, 14, 14)	-	-	-
UpConv2	(c, 14, 14)	4×4	2	1
SkipConv2	(256, 28, 28)	1×1	1	0
SkipConv2 + UpConv2	(c, 28, 28)	-	-	-
UpConv3	(c, 28, 28)	16×16	8	4
Softmax	(c, 224, 224)	-	-	-

TABLE C.1: Architecture details for our FCN-8 implementation. First column refers to the operation carried out in that layer. Note that all convolutional layers include batch normalization and a ReLU, except Conv16 and the SkipConv layers. Convolutional layers with (d) has dropout applied with p=0.5. Second column refers to the number of channels, height and width of the image passed into the layer. The c corresponds to the number of classes in the dataset.

Layer	**(C, H, W)**	**Kernel Size**	**Stride**	**Padding**
Conv1	(3, 224, 224)	3×3	1	1
Conv2	(64, 224, 224)	3×3	1	1
Pool1	(64, 224, 224)	2×2	2	0
Conv3	(64, 112, 112)	3×3	1	1
Conv4	(128, 112, 112)	3×3	1	1
Pool2	(128, 112, 112)	2×2	2	0
Conv5	(128, 56, 56)	3×3	1	1
Conv6	(256, 56, 56)	3×3	1	1
Pool3	(256, 56, 56)	2×2	2	0
Conv7	(256, 28, 28)	3×3	1	1
Conv8	(512, 28, 28)	3×3	1	1
Pool4	(512, 28, 28)	2×2	2	0
Conv9(d)	(512, 14, 14)	3×3	1	1
Conv10(d)	(512, 14, 14)	3×3	1	1
UpConv1	(512, 14, 14)	4×4	2	1
Conv11	(512+512, 28, 28)	3×3	1	1
Conv12	(512, 28, 28)	3×3	1	1
UpConv2	(256, 28, 28)	4×4	2	1
Conv13	(256+256, 56, 56)	3×3	1	1
Conv14	(256, 56, 56)	3×3	1	1
UpConv3	(128, 56, 56)	4×4	2	1
Conv15	(128+128, 112, 112)	3×3	1	1
Conv16	(128, 112, 112)	3×3	1	1
UpConv4	(64, 112, 112)	4×4	2	1
Conv17	(64+64, 224, 224)	3×3	1	1
Conv18	(64, 224, 224)	3×3	1	1
Conv19	(64, 224, 224)	1×1	1	0
Softmax	(c, 224, 224)	-	-	-

TABLE C.2: Architecture details for our U-Net implementation. First column refers to the operation carried out in that layer. Note that all convolutional layers include batch normalization and a ReLU, except Conv19. Convolutional layers with (d) has dropout applied with p=0.5.The c in the last row corresponds to the number of classes in the dataset.

Layer	(C, H, W)	Kernel Size	Stride	Padding
Conv1	(3, 224, 224)	3×3	1	1
Conv2	(64, 224, 224)	3×3	1	1
Pool1	(64, 224, 224)	2×2	2	0
Conv3	(64, 112, 112)	3×3	1	1
Conv4	(128, 112, 112)	3×3	1	1
Pool2	(128, 112, 112)	2×2	2	0
Conv5(d)	(128, 56, 56)	3×3	1	1
Conv6(d)	(256, 56, 56)	3×3	1	1
Conv7(d)	(256, 56, 56)	3×3	1	1
Pool3	(256, 56, 56)	2×2	2	0
Conv8(d)	(256, 28, 28)	3×3	1	1
Conv9(d)	(512, 28, 28)	3×3	1	1
Conv10(d)	(512, 28, 28)	3×3	1	1
Pool4	(512, 28, 28)	2×2	2	0
Conv11(d)	(512, 14, 14)	3×3	1	1
Conv12(d)	(512, 14, 14)	3×3	1	1
Conv13(d)	(512, 14, 14)	3×3	1	1
Pool5	(512, 7, 7)	2×2	2	0
UnPool1	(512, 7, 7)	2×2	2	0
Conv14(d)	(512, 14, 14)	3×3	1	1
Conv15(d)	(512, 14, 14)	3×3	1	1
Conv16(d)	(512, 14, 14)	3×3	1	1
UnPool2	(512, 14, 14)	2×2	2	0
Conv17(d)	(512, 28, 28)	3×3	1	1
Conv18(d)	(512, 28, 28)	3×3	1	1
Conv19(d)	(512, 28, 28)	3×3	1	1
UnPool3	(256, 28, 28)	2×2	2	0
Conv20(d)	(256, 56, 56)	3×3	1	1
Conv21(d)	(256, 56, 56)	3×3	1	1
Conv22(d)	(256, 56, 56)	3×3	1	1
UnPool4	(128, 56, 56)	2×2	2	0
Conv23	(128, 112, 112)	3×3	1	1
Conv24	(128, 112, 112)	3×3	1	1
UnPool5	(64, 112, 112)	2×2	2	0
Conv25	(64, 224, 224)	3×3	1	1
Conv26	(64, 224, 224)	3×3	1	1
Softmax	(c, 224, 224)	-	-	-

TABLE C.3: Architecture details for our SegNet implementation. First column refers to the operation(s) carried out in that layer. Note that all convolutional layers include batch normalization and a ReLU, except Conv26. Convolutional layers with (d) has dropout applied with p=0.5. Second column refers to the number of channels, height and width of the feature maps passed into the layer.

The c in the last row corresponds to the number of classes in the dataset.

Appendix D

Appendix Chapter 5

This appendix describes the data utilized in this thesis and further details regarding the exact split of the dataset into training, validation and test set.

D.1 Experimental Setup

D.1.1 Data

The Endoscene dataset was introduced in [11] and is actually a combination of two previous polyp segmentation dataset. To obtain a fair comparison with the previous work done using FCN-8 on semantic segmentation of colorectal polyp we follow [11] when dividing the dataset into training, validation and test set. Table D.1 displays the details regarding the dataset and the split.

Database	Training	Validation	Test	Resolution
CVC-Colon [5]	[1-76], [98-148], [221-273].	[77-97], [209-220], [274-300].	[149-208].	500×574
CVC-Clinic [114]	[26-50], [104-126], [178-227], [253-317], [384-503], [529-612].	[51-103], [228-252], [318-342], [364-383].	[1-25], [127-177], [343-363], [504-528].	384×288

TABLE D.1: Summary of the split used to construct the Endoscene dataset from CVC-Colon and CVC-Clinic.

Bibliography

[1] Rebecca L. Siegel, Kimberly D. Miller, and Ahmedin Jemal. Cancer statistics, 2017. *CA: A Cancer Journal for Clinicians*, 67(1):7–30, 2017. ISSN 1542-4863. doi: 10.3322/caac.21387. URL http://dx.doi.org/10.3322/caac.21387.

[2] Wanqing Chen, Rongshou Zheng, Peter D. Baade, Siwei Zhang, Hongmei Zeng, Freddie Bray, Ahmedin Jemal, Xue Qin Yu, and Jie He. Cancer statistics in china, 2015. *CA: A Cancer Journal for Clinicians*, 66(2):115–132, 2016. ISSN 1542-4863. doi: 10.3322/caac.21338. URL http://dx.doi.org/10.3322/caac.21338.

[3] Larsen IK red. Cancer in norway 2015 - cancer incidence, mortality, survival and prevalence in norway. oslo: Cancer registry of norway; 2016., 2016. URL https://www.kreftregisteret.no/globalassets/cancer-in-norway/2015/cin-2015.pdf.

[4] Rebecca L. Siegel, Kimberly D. Miller, Stacey A. Fedewa, Dennis J. Ahnen, Reinier G. S. Meester, Afsaneh Barzi, and Ahmedin Jemal. Colorectal cancer statistics, 2017. *CA: A Cancer Journal for Clinicians*, 67(3):177–193, 2017. ISSN 1542-4863. doi: 10.3322/caac.21395. URL http://dx.doi.org/10.3322/caac.21395.

[5] F. Javier Sanchez Jorge Bernal and Fernando Vilario. Towards automatic polyp detection with a polyp appearance model. *Pattern Recognition*, 45(9):3166–3182, 2012. URL http://mv.cvc.uab.es/projects/colon-qa/cvccolondb.

[6] Helsedirektoratet. Nasjonalt screeningprogram mot tarmkreft, 2017. URL https://helsedirektoratet.no/Documents/Kreft/Rapport%20om%20et%20Nasjonalt%20screeningprogram%20mot%20tarmkreft%20300617.pdf.

[7] Stoker J Bossuyt PM van Deventer SJ Dekker E. van Rijn JC, Reitsma JB. Polyp miss rate determined by tandem colonoscopy: a systematic review., 2006. URL https://www.ncbi.nlm.nih.gov/pubmed/16454841.

[8] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL http://arxiv.org/abs/1506.02640.

[9] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014. URL http://arxiv.org/abs/1411.4038.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[11] David Vázquez, Jorge Bernal, F. Javier Sánchez, Gloria Fernández-Esparrach, Antonio M. López, Adriana Romero, Michal Drozdzal, and Aaron C. Courville. A benchmark for endoluminal scene segmentation of colonoscopy images. *CoRR*, abs/1612.00799, 2016. URL http://arxiv.org/abs/1612.00799.

[12] Ignacio Arganda-Carreras, Srinivas C. Turaga, Daniel R. Berger, Dan Cirean, Alessandro Giusti, Luca M. Gambardella, Jrgen Schmidhuber, Dmitry Laptev, Sarvesh Dwivedi, Joachim M. Buhmann, Ting Liu, Mojtaba Seyedhosseini, Tolga Tasdizen, Lee Kamentsky, Radim Burget, Vaclav Uher, Xiao Tan, Changming Sun, Tuan D. Pham, Erhan Bas, Mustafa G. Uzunbas, Albert Cardona, Johannes Schindelin, and H. Sebastian Seung. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9:142, 2015. ISSN 1662-5129. doi: 10.3389/fnana.2015.00142. URL https://www.frontiersin.org/article/10.3389/fnana.2015.00142.

[13] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, L. Lanczi, E. Gerstner, M. A. Weber, T. Arbel, B. B. Avants, N. Ayache, P. Buendia, D. L. Collins, N. Cordier, J. J. Corso, A. Criminisi, T. Das, H. Delingette, . Demiralp, C. R. Durst, M. Dojat, S. Doyle, J. Festa, F. Forbes, E. Geremia, B. Glocker, P. Golland,

X. Guo, A. Hamamci, K. M. Iftekharuddin, R. Jena, N. M. John, E. Konukoglu, D. Lashkari, J. A. Mariz, R. Meier, S. Pereira, D. Precup, S. J. Price, T. R. Raviv, S. M. S. Reza, M. Ryan, D. Sarikaya, L. Schwartz, H. C. Shin, J. Shotton, C. A. Silva, N. Sousa, N. K. Subbanna, G. Szekely, T. J. Taylor, O. M. Thomas, N. J. Tustison, G. Unal, F. Vasseur, M. Wintermark, D. H. Ye, L. Zhao, B. Zhao, D. Zikic, M. Prastawa, M. Reyes, and K. Van Leemput. The multimodal brain tumor image segmentation benchmark (brats). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024, Oct 2015. ISSN 0278-0062. doi: 10.1109/TMI.2014.2377694.

[14] Hari Babu Nandpuru, S. S. Salankar, and V. R. Bora. Mri brain cancer classification using support vector machine. In *Electrical, Electronics and Computer Science (SCEECS), 2014 IEEE Students' Conference on*, pages 1–6, March 2014. doi: 10.1109/SCEECS.2014.6804439.

[15] S. L. A. Lee, A. Z. Kouzani, and E. J. Hu. A random forest for lung nodule identification. In *TENCON 2008 - 2008 IEEE Region 10 Conference*, pages 1–5, Nov 2008. doi: 10.1109/TENCON.2008.4766750.

[16] W. Huang, N. Li, Z. Lin, G. B. Huang, W. Zong, J. Zhou, and Y. Duan. Liver tumor detection and segmentation using kernel-based extreme learning machine. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 3662–3665, July 2013. doi: 10.1109/EMBC.2013.6610337.

[17] Yuan Sui, Ying Wei, and Dazhe Zhao. Computer-aided lung nodule recognition by svm classifier based on combination of random undersampling and smote, 05 2015.

[18] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL https://doi.org/10.1023/A:1022627411411.

[19] Tin Kam Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 278–, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7128-9. URL http://dl.acm.org/citation.cfm?id=844379.844681.

[20] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992. ISSN 00031305. URL http://www.jstor.org/stable/2685209.

[21] Jrme Louradour and Khalid Daoudi. State-of-the-art sequence kernels for svm speaker verification, 11 2008.

[22] Xiangying Wang and Yixin Zhong. Statistical learning theory and state of the art in svm. In *The Second IEEE International Conference on Cognitive Informatics, 2003. Proceedings.*, pages 55–59, Aug 2003. doi: 10.1109/COGINF.2003.1225953.

[23] Sergios Theodoridis and Konstantinos Koutroumbas. Chapter 3 - linear classifiers. In Sergios Theodoridis, , and Konstantinos Koutroumbas, editors, *Pattern Recognition (Fourth Edition)*, pages 91 – 150. Academic Press, Boston, fourth edition edition, 2009. ISBN 978-1-59749-272-0. doi: https://doi.org/10.1016/B978-1-59749-272-0.50005-0. URL https://www.sciencedirect.com/science/article/pii/B9781597492720500050.

[24] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines, April 1998. URL https://www.microsoft.com/en-us/research/publication/sequential-minimal-optimization-a-fast-algorithm-for-training-support-vector-machines/,.

[25] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 209:415–446, 1909. ISSN 02643952. URL http://www.jstor.org/stable/91043.

[26] Luís A. Alexandre, João Casteleiro, and Nuno Nobreinst. Polyp detection in endoscopic video using svms. In Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Knowledge Discovery in Databases: PKDD 2007*, pages 358–365, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-74976-9.

[27] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000029664.99615.94. URL https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[28] Y. Yuan and M. Q. H. Meng. Polyp classification based on bag of features and saliency in wireless capsule endoscopy. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3930–3935, May 2014. doi: 10.1109/ICRA.2014.6907429.

[29] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2372-2. doi: 10.1109/CVPR.2005.177. URL http://dx.doi.org/10.1109/CVPR.2005.177.

[30] H Agrahari, Yuji Iwahori, Manas Bhuyan, S Ghorai, H Kohli, Robert Woodham, and Kunio Kasugai. Automatic polyp detection using dsc edge detector and hog features, 01 2014.

[31] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, Aug 2000. ISSN 0162-8828. doi: 10.1109/34.868688.

[32] Xiaodong Zhang, Fucang Jia, Suhuai Luo, Guiying Liu, and Qingmao Hu. A marker-based watershed method for x-ray image segmentation. *Computer Methods and Programs in Biomedicine*, 113(3):894 – 903, 2014. ISSN 0169-2607. doi: https://doi.org/10.1016/j.cmpb.2013.12.025. URL http://www.sciencedirect.com/science/article/pii/S016926071300415X.

[33] Filipe Condessa and José Bioucas-Dias. Segmentation and detection of colorectal polyps using local polynomial approximation. In Aurélio Campilho and Mohamed Kamel, editors, *Image Analysis and Recognition*, pages 188–197, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-31298-4.

[34] Jorge Bernal, Joan Manel Núñez, F. Javier Sánchez, and Fernando Vilariño. Polyp segmentation method in colonoscopy videos by means of msa-dova energy maps calculation, 2014. URL https://doi.org/10.1007/978-3-319-13909-8_6.

[35] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL http://arxiv.org/abs/1409.4842.

[36] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL http://arxiv.org/abs/1504.08083.

[37] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL http://arxiv.org/abs/1506.01497.

[38] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016. doi: 10.1109/CVPR.2016.91.

[39] Geert J. S. Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A. W. M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A survey on deep learning in medical image analysis. *CoRR*, abs/1702.05747, 2017. URL http://arxiv.org/abs/1702.05747.

[40] E. Ribeiro, A. Uhl, and M. Hfner. Colonic polyp classification with convolutional neural networks. In *2016 IEEE 29th International Symposium on Computer-Based Medical Systems (CBMS)*, pages 253–258, June 2016. doi: 10.1109/CBMS.2016.39.

[41] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[42] Qinghui Liu. Deep learning applied to automatic polyp detection in colonoscopy images : master thesis in system engineering with embedded systems, may 2017.

[43] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL http://www.sciencedirect.com/science/article/pii/089360809190009T.

[44] Paul Werbos. Beyond regression: New tools for predicting and analysis in the behavioral sciences, 11 1974.

[45] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research, 1988. URL http://dl.acm.org/citation.cfm?id=65669.104451.

[46] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jrgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[47] Rosenblatt. F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[48] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. doi: 10.1007/BF02478259. URL https://doi.org/10.1007/BF02478259.

[49] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks, 11–13 Apr 2011. URL http://proceedings.mlr.press/v15/glorot11a.html.

[50] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL http://dl.acm.org/citation.cfm?id=3104322.3104425.

[51] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, Sept 2009. doi: 10.1109/ICCV.2009.5459469.

[52] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models, 2013.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

[54] Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, pages 381–387, Cambridge, MA, USA, 2000. MIT Press. URL http://dl.acm.org/citation.cfm?id=3008751.3008807.

[55] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines, 2012. URL https://doi.org/10.1007/978-3-642-35289-8_32.

[56] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 78–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015435. URL http://doi.acm.org/10.1145/1015330.1015435.

[57] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[58] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. URL http://proceedings.mlr.press/v9/glorot10a.html.

[59] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

[60] David Bradley. *Learning In Modular Systems*. PhD thesis, Robotics Institute , Carnegie Mellon University, Pittsburgh, PA, May 2010.

[61] Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995. doi: 10.1162/neco.1995.7.2.219. URL https://doi.org/10.1162/neco.1995.7.2.219.

[62] Lutz Prechelt. Early stopping — but when?, 2012. URL https://doi.org/10.1007/978-3-642-35289-8_5.

[63] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[64] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 07 2012.

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

[66] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://arxiv.org/abs/1311.2901.

[67] Sebastien C. Wong, Adam Gatt, Victor Stamatescu, and Mark D. McDonnell. Understanding data augmentation for classification: when to warp? *CoRR*, abs/1609.08764, 2016. URL http://arxiv.org/abs/1609.08764.

[68] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

[69] Irwin Sobel. An isotropic 3x3 image gradient operator, 02 2014.

[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[71] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

[72] Simon Jégou, Michal Drozdzal, David Vázquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. *CoRR*, abs/1611.09326, 2016. URL http://arxiv.org/abs/1611.09326.

[73] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results. http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html, 2011.

[74] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference*

on Machine Learning, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/gal16.html.

[75] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *CoRR*, abs/1703.04977, 2017. URL http://arxiv.org/abs/1703.04977.

[76] G. Alain and Y. Bengio. Understanding intermediate layers using linear classifier probes. *ArXiv e-prints*, October 2016.

[77] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013. URL http://arxiv.org/abs/1312.6034.

[78] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014. URL http://arxiv.org/abs/1412.6806.

[79] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015. URL http://arxiv.org/abs/1503.02406.

[80] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017. URL http://arxiv.org/abs/1703.00810.

[81] S. Yu and J. C. Principe. Understanding Autoencoders with Information Theoretic Concepts. *ArXiv e-prints*, March 2018.

[82] Pejman Khadivi, Ravi Tandon, and Naren Ramakrishnan. Flow of information in feed-forward deep neural networks. *CoRR*, abs/1603.06220, 2016. URL http://arxiv.org/abs/1603.06220.

[83] Eder Santana, Matthew Emigh, and José C. Príncipe. Information theoretic-learning auto-encoder. *CoRR*, abs/1603.06653, 2016. URL http://arxiv.org/abs/1603.06653.

[84] S. Yu, R. Jenssen, and J. C. Principe. Understanding Convolutional Neural Network Training with Information Theory. *ArXiv e-prints*, April 2018.

[85] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb01338.x.

[86] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley and Sons Inc, second edition, 2006.

[87] Andrew Michael Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan Daniel Tracey, and David Daniel Cox. On the information bottleneck theory of deep learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ry_WPG-A-.

[88] Jose C. Principe. *Information Theoretic Learning: Renyi's Entropy and Kernel Perspectives*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 1441915699, 9781441915696.

[89] Alfrd Rnyi. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 547–561, Berkeley, Calif., 1961. University of California Press. URL https://projecteuclid.org/euclid.bsmsp/1200512181.

[90] Emanuel Parzen. On estimation of a probability density function and mode. *Ann. Math. Statist.*, 33(3):1065–1076, 09 1962. doi: 10.1214/aoms/1177704472. URL https://doi.org/10.1214/aoms/1177704472.

[91] Sergios Theodoridis and Konstantinos Koutroumbas. Chapter 2 - classifiers based on bayes decision theory. In Sergios Theodoridis, , and Konstantinos Koutroumbas, editors, *Pattern Recognition (Fourth Edition)*, pages 13 – 89. Academic Press, Boston, fourth edition edition, 2009. ISBN 978-1-59749-272-0. doi: https://doi.org/10.1016/B978-1-59749-272-0.50004-9. URL https://www.sciencedirect.com/science/article/pii/B9781597492720500049.

[92] Luis Gonzalo Sánchez Giraldo, Murali Rao, and José C. Príncipe. Measures of entropy from data using infinitely divisible kernels. *CoRR*, abs/1211.2459, 2012. URL http://arxiv.org/abs/1211.2459.

[93] Roger A. Horn. On infinitely divisible matrices, kernels, and functions. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 8(3):219–230, Sep 1967. ISSN 1432-2064. doi: 10.1007/BF00531524. URL https://doi.org/10.1007/BF00531524.

[94] Bernard Silverman. Density estimation for statistics and data analysis, 01 1986.

[95] Jonas Myhre and Robert Jenssen. Mixture weight influence on kernel entropy component analysis and semi-supervised learning using the lasso, 09 2012.

[96] Avi Ben-Cohen, Idit Diamant, Eyal Klang, Michal Amitai, and Hayit Greenspan. Fully convolutional network for liver segmentation and lesions detection. In Gustavo Carneiro, Diana Mateus, Loïc Peter, Andrew Bradley, João Manuel R. S. Tavares, Vasileios Belagiannis, João Paulo Papa, Jacinto C. Nascimento, Marco Loog, Zhi Lu, Jaime S. Cardoso, and Julien Cornebise, editors, *Deep Learning and Data Labeling for Medical Applications*, pages 77–85, Cham, 2016. Springer International Publishing.

[97] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015. URL http://arxiv.org/abs/1511.00561.

[98] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

[99] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *CoRR*, abs/1411.4734, 2014. URL http://arxiv.org/abs/1411.4734.

[100] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[101] T Tieleman and G Hinton. Rmsprop adaptive learning. in: Coursera: Neural networks for machine learning, 2012.

[102] Michael Kampffmeyer, Arnt-Borre Salberg, and Robert Jenssen. Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2016.

[103] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2014. URL http://arxiv.org/abs/1412.7062.

[104] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015. URL http://arxiv.org/abs/1511.07122.

[105] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014. URL http://arxiv.org/abs/1411.4389.

[106] M. Ciccone, M. Gallieri, J. Masci, C. Osendorfer, and F. Gomez. NAIS-Net: Stable Deep Networks from Non-Autonomous Differential Equations. *ArXiv e-prints*, April 2018.

[107] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. http://neuralnetworksanddeeplearning.com/.

[108] Sergios Theodoridis and Konstantinos Koutroumbas. Chapter 4 - nonlinear classifiers. In Sergios Theodoridis, , and Konstantinos Koutroumbas, editors, *Pattern Recognition (Fourth Edition)*, pages 151 – 260. Academic Press, Boston, fourth edition edition, 2009. ISBN 978-1-59749-272-0. doi: https://doi.org/10.1016/B978-1-59749-272-0.50006-2. URL https://www.sciencedirect.com/science/article/pii/B9781597492720500062.

[109] Yann Dauphin, Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, abs/1406.2572, 2014. URL http://arxiv.org/abs/1406.2572.

[110] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(98)00116-6. URL http://www.sciencedirect.com/science/article/pii/S0893608098001166.

[111] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2), 1983.

[112] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, jul 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2021068.

[113] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL http://arxiv.org/abs/1212.5701.

[114] Snchez F. J. Fernndez-Esparrach G. Gil D. Rodrguez C. Bernal, J. and F. Vilario. Wm-dova maps for accurate polyp highlighting in colonoscopy: Validation vs. saliency maps from physicians. *Computerized Medical Imaging and Graphics*, 43:99–110, 2015. URL https://polyp.grand-challenge.org/site/Polyp/CVCClinicDB/.