A self-learning teacher-student framework for gastrointestinal image classification

Henrik Løland Gjestang

Thesis submitted for the degree of
Master in Computational Science
(Imaging and Biomedical Computing)
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2020

A self-learning teacher-student framework for gastrointestinal image classification

Henrik Løland Gjestang

© 2020 Henrik Løland Gjestang

A self-learning teacher-student framework for gastrointestinal image classification

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

Printed: Reprosentralen, University of Oslo

Abstract

Medical data is growing at an estimated 2.5 exabytes per year [1]. However, medical data is often sparse and unavailable for the research community, and qualified medical personnel rarely have time for the tedious labeling work required to prepare the data. New screening methods of the gastrointestinal (GI) tract, like video capsule endoscopy (VCE), can help to reduce patients discomfort and help to increase screening capabilities. One of the main reasons why VCE is not more commonly used by medical experts is the amount of data it produces. A high level of extra work is required by the physicians who, depending on the patient, have to look at more than 50,000 frames per examination. To make VCE more accepted and useful data analysis methods such as machine learning can be very useful.

Even if a lot of frames are collected per patient they are most of the time showing normal tissue without any relevant finding. This introduces another problem, namely that it is difficult to train a machine learning based method using this data. Existing models often struggle with the challenge of not having enough data that contains anomalies. This often leads to overfitted and not generalisable models. Our work explores ways to help existing models to overcome this problem by utilising a popular sub-category of machine learning called semi-supervised learning. Semi-supervised learning uses a combination of labeled and unlabeled data which allows us to take advantage of large amounts of unlabeled data.

In this thesis, we introduce our proposed semi-supervised teacher-student framework. This framework is built specifically to take advantage of vast amount of unlabeled data and consists of three main steps: (1) train a teacher model with labeled data, (2) use the teacher model to infer pseudo labels with unlabeled data, and (3) train a new and larger student model with a combination of labeled images and inferred pseudo labels. These three steps are repeated several times by treating the student as a teacher to relabel the unlabeled data and consequently training a new student.

We demonstrate that our framework can be of use for classifying both, VCE and endoscopic colonoscopy images or videos. We demonstrate that our teacher-student model can significantly increase the performance compared to traditional supervised-learning-based models. We believe that our framework has potential to be a useful addition to existing medical multimedia systems for automatic disease detection, because new data can be continuously added to improve the models performance while in production.

Acknowledgements

I would like to thank my supervisors Pål Halvorsen and Michael Riegler for all the help and motivation which has kept me going throughout my thesis, and for the opportunity of working on this research topic. I also wish to express my gratitude towards the two PhD student; Steven Hicks and Vajira Thambawita. Thank you for all your help, advice and support, of which I was blessed with during late nights, weekends, holidays and a global pandemic. You are the wisest men I ever knew, yet, no question felt too stupid to ask.

I would also like to thank my parents, Heidi and Brede, and my sister Agnete, for all the support and encouragement I have received both before, and during the thesis. Without you, none of this would have been possible.

Contents

List of Figures xi

List of Tables xiii

1 Introduction 3
1.1 Background and Motivation 3
1.2 Problem statement 4
1.3 Scope and limitations 5
1.4 Research methods 6
1.4.1 Theory 6
1.4.2 Abstraction 6
1.4.3 Design 7
1.5 Main Contributions 7
1.6 Thesis Outline 9

2 Background 11
2.1 Medical scenario 11
2.1.1 The digestive system 12
2.1.2 Colorectal Cancer and Screening 13
2.1.3 Traditional Endoscopy 14
2.1.4 Wireless Capsule Endoscopy 16
2.1.5 Remote diagnostic 21
2.2 Datasets 21
2.2.1 ImageNet 21
2.2.2 Available endoscopy datasets 22
2.2.3 Kvasir-V2 22
2.2.4 Hyper-Kvasir 24
2.2.5 Augere Medical AS 27
2.2.6 Class imbalance in dataset 28
2.3 Deep learning 30
2.3.1 Machine learning types 30
2.3.2 Convolutional Neural Network 33
2.3.3 Gradient descent optimization algorithms 36
2.3.4 ResNet 39

2.3.5 EfficientNet 39
2.3.6 Self learning with noisy student 42
2.4 Model evaluation 43
2.4.1 Dataset splitting 44
2.4.2 Performance metrics 45
2.4.3 Cross validation 47
2.5 TensorFlow Framework 47
2.5.1 tf.keras 47
2.5.2 tf.data 48
2.6 Related work 48
2.6.1 Object tracking 49
2.6.2 Segmentation 50
2.6.3 Mapping 51
2.7 Summary 52

3 Methodology 55
3.1 Data collection 55
3.1.1 Privacy, Legal and Ethics Issues 56
3.1.2 Kvasir-Capsule 56
3.2 Data pipeline 60
3.2.1 Splitting and resize images 61
3.2.2 Loading images into the pipeline 62
3.2.3 Optimize performance 63
3.2.4 Shuffle the dataset 65
3.2.5 Repeat 66
3.2.6 Data augmentation 67
3.2.7 Batching 69
3.2.8 Handling dataset class imbalance 70
3.3 Training and parameter tuning 74
3.3.1 Batch size 74
3.3.2 Weight initialization 75
3.3.3 Learning rate 76
3.4 Teacher-student architecture 79
3.4.1 Python implementation of system 80
3.4.2 Generating new pseudo labels 85
3.4.3 Feature drifting 88
3.4.4 Evaluation methods and metrics 91
3.5 Summary 92

4 Experiments and results 93
4.1 Experiment management 93
4.1.1 Keeping track of experiments 94
4.1.2 Evaluation method and metrics 94
4.1.3 Hardware details 96

4.1.4 Network architecture 97
4.1.5 Network parameters 98
4.1.6 Labeled and unlabeled dataset 101
4.2 Optimizing the teacher model 101
4.2.1 Benefits of pre-trained weights 101
4.2.2 Re-sampling versus weighting classes 103
4.2.3 Effect of varying image resolution 105
4.2.4 Neural network dimensions 108
4.2.5 Models ability to learn class features 110
4.3 Teacher-student model on Hyper-Kvasir 114
4.3.1 Evaluation metrics 114
4.3.2 Model complexity: iterative training 114
4.3.3 Noising the student 116
4.4 Teacher-student model on Kvasir-Capsule 117
4.4.1 Importance of good dataset split 118
4.4.2 Unlabeled dataset 121
4.4.3 Evaluate Kvasir-Capsule results 122
4.4.4 Noisy student experiments 122
4.5 Summary 128

5 Conclusions and Further Work 131
5.1 Summary 131
5.2 Contributions 132
5.3 Further work 133

Bibliography 137

List of Figures

1.1 Image of a video capsule 3

2.1 An overview of the terms used to describe the digestive system 13
2.2 Image from Kvasir-V2 dataset of a polyp in the colon 15
2.3 Image of a fiber optic endoscope with explanation of different parts of the tool 15
2.4 Distinction of the nomenclature relating to capsules, wirelessness and video 19
2.5 Images taken with VCE from stomach and small intestine 19
2.6 Video capsule endoscopy equipment manufactured by Olympus 20
2.7 Number of labeled samples for each of the 23 classes in Hyper-Kvasir dataset 25
2.8 Example of a segmented image from Hyper-Kvasir dataset 27
2.9 Augere Medical tagging tool screen capture 28
2.10 Two popular resampling methods 29
2.11 Workflow of supervised machine learning. 31
2.12 Reinforcement learning: Agent and environment. 32
2.13 Different model scaling methods 41
2.14 A visualization of the splits 44
2.15 Illustration of how object in two frames is tracked with a bounding box . 50
2.16 An example of Deep EndoVO accuracy 52

3.1 Image examples of the various labeled classes for images 57
3.2 Distribution of labeled samples per class in the Kvasir-Capsule dataset . . 58
3.3 Pipeline for training model 60
3.4 The input pipeline 61
3.5 Naive pipeline. 63
3.6 Pipeline with prefetching. 64
3.7 Naive pipeline with mapping. 65
3.8 Pipeline with parallel mapping. 65
3.9 Pipeline with cache. 66
3.10 The effect of data augmentation on sample image. 67
3.11 The effect of reduced data augmentation on sample image. 69
3.12 EfficientNetB0 with custom top 75
3.13 Learning rate with inverse time decay 77
3.14 Effect of inverse time decay on training and validation loss. 78
3.15 Illustration of the Noisy student method 79

3.16 Number of images of interest found in unlabeled dataset 85
3.17 Class distribution development after two additions of pseudo labels. . . . 87
3.17 Generated pseudo label by teacher model trained on labeled data 90
3.18 Pseudo labels of the hemorrhoids class, generated by teacher 1 and teacher 2 91

4.1 Scatter plot of precision and recall 96
4.2 Confusion Matrix of a EfficientNet model trained on Hyper-Kvasir dataset 97
4.3 Similarities between impacted-stool and polyps classes 99
4.4 Example where the model accuracy for training data greatly outperforms the accuracy for testing data 100
4.5 Loss computed on training and validation data using carefully initialized weight bias compared against zero initial bias 103
4.6 EfficientNetB0 trained with different layer weights initialization 104
4.7 Confusion matrix from model trained on resampled data 106
4.8 Loss and accuracy calculated on training and validation data respectively for a EfficientNetB4 model 106
4.9 ROC curve with results for baseline, weighted and resampled models trained on a binary version of Hyper-Kvasir 107
4.10 Accuracy and loss calculated on Hyper-Kvasir labeled training data for three models 108
4.11 Model accuracy and loss during training for different image resolutions . . 109
4.12 Different resolutions for input pipeline images. 109
4.13 Four EfficientNet models with different network dimensions 111
4.14 Data prediction for Averaged ResNet-152 + DenseNet-161 113
4.15 Inferred pseudo labels of Hyper-Kvasir unlabeled dataset comparison . . . 113
4.16 F_1-score measured per class in teacher-student model after 6 iterations . . 115
4.17 Precision, recall, and F_1-score for teacher-student model 115
4.18 Average weighted F_1-score for different EfficientNets 117
4.19 Noisy student compared against un-noised student 118
4.20 Trailing frames taken from Angiectasia class of Kvasir-Capsule dataset . . 119
4.21 Accuracy and loss from training and validation dataset after 18 epoch of training on Kvasir-Capsule dataset 119
4.22 Accuracy and loss from training and validation dataset after 15 epoch of training on Kvasir-Capsule 121
4.23 Accuracy and loss computed on the split_0 and split_1 during training of 3 iterations of teacher and student models for 25 epochs 121
4.24 Accuracy and loss computed on the validation data during training of 3 iterations of teacher and student models for 25 epochs 124
4.25 Averaged accuracy and F_1-score for both splits after 3 iterations of switching out the teacher with the student 124
4.26 Weighted average score for recall, precision and F_1-score for teacher-student models run for 3 iterations 125
4.27 Averaged accuracy and F_1-score for both splits after 3 iterations of switching out the teacher with the student 126

4.28 Confusion matrix created by testing the last student model on split_1 of Kvasir-Capsule dataset 127
4.29 Preview of pseudo labels from Kvasir-Capsule unlabeled dataset 129

List of Tables

2.1 List of the most common types of endoscopy. 17
2.2 Existing colonoscopy image and video datasets 23
2.3 An overview of VCE datasets from the GI tract 24
2.4 Kvasir-V2 classes 24
2.5 Hyper-Kvasir class descriptions 26
2.6 EfficientNet performance results on ImageNet 42
2.7 Confusion matrix layout for a binary classifier 45
2.8 Segmentation results on the ISBI cell tracking challenge in 2015 51

3.1 Distribution of samples per class before and after dataset re-sampling . . 72
3.2 Hyper-Kvasir class weights 74
3.3 Max batch size for each EfficientNet model 76

4.1 Classification report generated by system 95
4.2 Validation report for a sample of generated pseudo label. 98
4.3 System specifications 99
4.4 List of weights use during experiment 105
4.5 Metrics for three models, with three different methods of handling class imbalance 105
4.6 Noised student metrics from training on Hyper-Kvasir 117
4.7 Two fold dataset split of Kvasir-Capsule dataset 120
4.8 Noised student metrics from training on Kvasir-Capsule 125
4.9 Noised student metrics from training on Kvasir-Capsule 126
4.10 Validation report for a sample of generated pseudo label. 128

Chapter 1

Introduction

1.1 Background and Motivation

Colorectal cancer (CRC) is the third most common cause of cancer mortality for both men and women [2], and it is a condition where early detection is of clear value for the ultimate survival of the patient. As statistics show that 15% of male and female above 50 years are at risk, the procedure is recommended on a regular basis (every 3-5 years) for the population over 50, and from an earlier age for high-risk groups.

Figure 1.1: Image of a video capsule[1].

Colonoscopy is a demanding procedure requiring an significant amount of time by specialized physicians, in addition to the discomfort and risks inherent in the procedure. Traditional methods based on colonoscopy are not cost-effective for population-based screening purposes, so only about $2-3\%$ of the target population is reached at present. The cost of a population screening program is prohibitively expensive. Colonoscopy is the most expensive cancer screening process in the US, with annual costs of $10 billion dollars ($1100 per person) [3]. In Norway we have similar costs of around $1000 per person, with a time consumption of about 1 doctor-hour and 2 nurse-hours per

[1] Image Credit: Medtronic

examination.

By researching an automatic system for a camera pill, the aim is to greatly increase the number of patients that can be examined, i.e., making the public health care system more scalable and cost effective, while at the same time reducing the need for intrusive procedures like ”bottom-up” examinations like colonoscopy.

In this thesis, we aim to design and develop a system for analyzing medical images from a camera pill, as seen in Figure 1.1. The pill is swallowed and records video of the entire digestive system. The goal is to be able to detect different irregularities in the patients digestive system, like a colon polyp, erosions, blood, etc. by using machine learning and other relevant tools. However, medical data is often sparse and unavailable to the research community, and qualified medical personnel rarely have time for the tedious labelling work. In this respect, we have gathered our own dataset with video capsule data provided by Department of Medicine at Bærum Hospital, Vestre Viken Hospital Trust in Norway. Still, some pathological findings in the data are very rare and are only represented by tens of samples. Because of this lack of labeled data, we look to a branch of Artificial Intelligence (AI) called semi-supervised learning, which can take advantage of large amounts of unlabeled data to further improve upon traditional supervised models. The semi-supervised learning algorithm is to train a model on labeled data, then use the model to predict image labels, called pseudo labels, from a corpus of unlabeled images, then finally train a new and improved model on the combination of labeled images and pseudo labels. We call this self-learning framework for a teacher-student framework because we first train a model on the labeled data (the teacher), and then use the teacher to train a student which eventually becomes better than the teacher.

1.2 Problem statement

Based on the background and motivation presented in the previous section, we decided to look into how unlabeled medical data can help to improve computer-aided diagnostic in the GI tract. The research question we want to answer in this project is the following:

> *Can a semi-supervised teacher-student framework improve on traditional supervised models by incorporating inferred pseudo labels into the labeled training data in the field of gastrointestinal tract endoscopy?*

Because of the nature of highly sparse and skewed labeled medical datasets we are especially interested in if this method can create models which are better at classifying minority classes or otherwise improve the class imbalance problem. From our research question, we define the objectives targeted by this thesis as follows:

Objective 1 Collect data for a video capsule endoscopy dataset of both labeled and unlabeled images from the gastrointestinal tract, with a skewed balance of class samples to represent real world scenario. This dataset should be used for testing our framework.

Objective 2 Provide a implementation of a teacher-student framework for multiclass image classification based on the novel EfficientNet architecture, with a suite of evaluation tools to help with further analysis.

Objective 3 Use various model hyper-parameters and framework setting to get a better understanding of the effect which is caused by combining pseudo labels with original training images, and map performance gains by using various network dimensions.

To meet these three objectives we must undergo literature searches, research and development. Because of this, we also decided to define two requirements which we should consider when developing the dataset and teacher-student framework. The two requirements are as follows:

Requirement 1 All data and code used to complete this project should be open source, and easily accessible.

Requirement 2 The technology and tools used to create our teacher-student framework should be mature and widely tested in the field of deep learning. This will ensure all programming libraries used are well documented and easily available for others such that our results are reproducible.

1.3 Scope and limitations

Based on the described problem statement, the scope of this thesis is to focus on the completion of our three main objectives, which act as the initial steps of answering the research question introduced in Section 1.2. The first objective is to create a dataset of labeled and unlabeled images from the gastrointestinal tract taken with video capsule endoscopy. The second objective is to develop a semi-supervised teacher-student framework and corresponding framework for validation the results. We will be using the largest, to the best of our knowledge, colonoscopy dataset Hyper-Kvasir to do preliminary testing and validation and then test on our capsule endoscopy dataset and compare the results. The last main objective is to find suitable parameters to use for our model training and framework settings.

Considering the scope of this thesis, we will limit ourselves to use the novel family of EfficientNets. The EfficientNet architecture uses compound scaling to uniformly scale network width, depth and resolution to create an optimal network that capture all fine-grained features of an image. By using different compound scaling coefficients we can easily compare different network dimensions for our teacher-student framework. Had we had more time and resources we would have tested with many different architectures to find the optimal one.

An additional limiting factor is the low temporal information achieved by the video capsule endoscopy data which we received from Vestre Viken Hospital Trust. This limitation is due to battery and frame-rate trade-offs made by the device manufacturer and for this reason we opted to create a system which only handles still frames and not video.

Another limiting factor is the computer memory on the machines we have used to run our experiments. Due to time constraints and other limited resources we prioritized not to go deeper with code optimization, but rather to take other actions like image resolution reduction and model downscaling.

1.4 Research methods

For this thesis, we have decided to use the Association for Computing Machinerys (ACMs) methodology for our research. In the spring of 1986 ACM president Adele Goldberg and ACM Education Board Chairman Robert Aiken appointed a task force with the prime objective of describing the core fundamentals of computer science and computer engineering into a detailed report [4].

The report describes three major paradigms which represent different areas of competence in the field of computer science and computer engineering. Some will argue that the different paradigms are implicitly based on an assumption that one of the three processes is the most fundamental, but as we will see, the three paradigms are so intricately intertwined that it is irrational to say that one is the most fundamental. Below, we give a brief description of each paradigm and discuss how our work fits into each of them.

1.4.1 Theory

The first paradigm, Theory, is deeply rooted in mathematics and is concerned with the ability to describe and prove relationships among objects. The paradigm consist of the following four steps; (i) characterize objects of study (definition), (ii) hypothesize possible relations among them (theorem), (iii) determine whether the relationships are true (proof), and (iv) interpret the results found.

In this thesis we support this paradigm by touching upon the theory behind machine learning and convolutional neural networks. We identify the problems regarding multiclass classification on small datasets with skewed class balance and propose theoretical solutions which later are tested in practice.

1.4.2 Abstraction

The second paradigm is abstraction. Abstraction is a form of modeling and is rooted in the experimental scientific method. This paradigm is concerned with the ability to use the relationships found in the theory paradigm to make predictions that then can be compared to the real world. It has four steps, which are described as follows; (i) form a hypothesis, (ii) construct a model and make a prediction, (iii) design an experiment and collect data, and (iv) analyze the results.

Our work support the abstraction paradigm by analyzing the relationship between the convolutional neural networks and its predicted results. Using this information we identify problems regarding class imbalance, overfitting and lack of generalisability. By forming a hypothesis, we then reran the experiments with additional data input noise, and interpreted how the change in dataset affected the models performance.

1.4.3 Design

The third paradigm, design, is rooted in engineering and consist, like the others, of four main steps listed below. The design paradigm is concerned with the ability to implement specific instances of those relationships and use them to perform useful actions. The following four steps will help an engineer to construct a device or system to complete a given task; (i) state requirements, (ii) state specifications, (iii) design and implement the system, and (iv) test the system.

This paradigm is supported by the completion of our teacher-student framework. This framework was extensively used throughout the thesis to conduct a plethora of experiments.

1.5 Main Contributions

Over the course of this thesis, we have researched and developed a semi-supervised teacher-student framework for classification of pathological findings and anatomical landmarks in the gastrointestinal tract. This framework focus on using a large corpus of unlabeled images exported from video capsule endoscopy taken during patient examinations at Bærum hospital, by iteratively inferring pseudo labels and combining them with the labeled training images to increase model performance. As defined in our problem statement in Section 1.2, we set two requirements which our framework should meet to be considered finished (within the context of this thesis). We will reiterate the requirements and describe how our system meets them:

Requirement 1 *All data and code used to complete this project should be open source, and easily available.*

This requirement is supported by our dataset publication "Kvasir-Capsule, a Video Capsule Endoscopy Dataset" [5], and our provided GitHub repository[2]. Our initial experiments and research are done on another open source dataset, Hyper-Kvasir[3] [6].

Requirement 2 *The technology and tools used to create our teacher-student framework should be mature and widely tested in the field of deep learning. This will ensure all programming libraries used are well documented and easily available for others such that our results are reproducible.*

[2] `https://github.com/henriklg/master-thesis`

[3] `https://datasets.simula.no/hyper-kvasir/`

This requirement is supported by our proposed teacher-student framework introduced in Section 3.4. We have chosen to create own implementations of methods which was possible in a timely manner. This reduce library dependencies and framework complexity. Other libraries we have used are well used in the domain of deep learning, and are well documented.

With these requirements fulfilled, we look at how our teacher-student framework solves the three research objectives which define the work that should have been done over the course of this thesis.

Objective 1 *Collect data for a video capsule endoscopy dataset of both labeled and unlabeled images from the gastrointestinal tract, with a skewed balance of class samples to represent real world scenario. This dataset should be used for testing our framework.*

This objective is supported by our creation of a gastrointestinal dataset, Kvasir-Capsule [5], containing a total of 44,260 manually labeled images with bounding boxes around the respective finding, split into 13 classes for pathological findings, anatomical landmarks and quality of mucosal view. The annotation was performed by three MSc student, supervised by an expert endoscopist with many years in the field. Whenever the MSc student encountered an issue, the endoscopist reviewed the case. We also include the 44 videos used for extraction of labeled images, as well as 72 videos which are not labeled and thus useful for unsupervised and semi-supervised machine learning system or to generate more labeled images by other qualified personnel in the future. All videos are taken by video capsule endoscopy during a number of examinations at Bærum Hospital in Norway, between the year 2016 and 2018. The Kvasir-Capsule dataset is available from the Open Science Framework (OSF) accessible via the link `https://osf.io/dv2ag/`.

Objective 2 *Provide a implementation of a teacher-student framework for multiclass image classification based on Google's novel EfficientNet architecture with a suite of evaluation tools.*

This objective is supported by our proposed teacher-student framework presented in Section 3.4, which use a teacher model, based on Google's EfficientNet architecture, to create pseudo labels from unlabeled endoscopy images, which are then combined with original labeled training data. Next, we create a larger student model with more stochastic noise, and inject the input pipeline with noise transformations, like image translation, rotation and variances in brightness, saturation etc, to create a larger and more noisy student model. This noised student model learn more features from the combined dataset of labeled images and inferred pseudo labels, than the smaller teacher model. This process is then iterated a couple of times to further increase model performance.

Objective 3 *Use various model hyper-parameters and framework setting to get a better understanding of the effect which is caused by combining pseudo labels with original training images, and map performance gains by using various network dimensions.*

This last objective is supported by our research made in Chapter 4, where we present a detailed analysis and ablations of various design choices, such as architecture, hyperparameters, class imbalance equalizing methods, image input resolution and more. When performing such experiments we measure cross-entropy loss, model accuracy, recall, precision and F_1-score during training and create easy-to-diagnose plots and reports for every model. The data is split in suitable folds and used for training and validation to ensure good validity. Based on this performed analysis, we derive a configuration of our teacher-student framework which improve on the baseline performance of our initial EfficientNet models by 3.2% for Hyper-Kvasir and 4.7% for Kvasir-Capsule.

Through the work produced in this thesis, and by answering the stated research objectives, we have learned the value of using pre-trained network weights to greatly reducing training time, importance of sampling a imbalanced dataset to help the model generalize better during training, how changes in image resolution speed up training at the cost of model performance, and the various effects of changing network dimensions. With this knowledge we then designed and developed a self-learning teacher-student framework. This semi-supervised teacher-student framework, trained on sparsely and skewed labeled video capsule endoscopy images and traditional endoscopy images, has shown the ability to improve on traditional supervised models in our conducted experiments with varying results. With more tuning of the framework settings and more data for both training and validation, this self-learning paradigm of machine learning can have profound effects on the future of computer-assisted diagnose in the medical domain.

1.6 Thesis Outline

This thesis is split into five chapters. Chapters one and two are mostly to introduce the reader to the topic and to fill in the necessary knowledge to understand the rest of the thesis. In the last chapter, we conclude on our findings and discuss our findings and propose further work. The papers that have been referenced in the thesis is added in the bibliography at the very end. The chapters in this thesis are summarized below:

- In Chapter 2, we discuss the literature that focus on the topic of the digestive system, patient screening, endoscopy, endoscopy datasets, and deep learning used for automated lesion detection in computer systems.
- In Chapter 3, we present the details of design, implementation of our semi-supervised image classification system and the processing and collection of data.
- In Chapter 4, we present the experiments we have conducted with the different image datasets, and the results
- In Chapter 5, we provide a comprehensive overview of the results found and discuss what that contributes to the field and propose some further work.

Chapter 2

Background

In recent years, there have been many proposed methods to use automated object tracking, segmentation, and deep learning to produce a better, and cheaper health care system [7]–[10]. Many of these methods are considered state of the art systems within the fields of deep learning. One requirement for such a system to work in reality is a good flow of data. Ideally, all the data should be labeled by a doctor before it is used for training deep neural networks, but this is rarely the case. We propose a method that takes advantage of this unlabeled data which is more readily available.

In this chapter, we will present the necessary background and related works to understand how such a semi-supervised model can be built. This will be covered over two main parts; one where we go through the related works and background to understand the medical aspect of this topic and the other will cover the technical use of deep learning in mission-critical fields such as the medical domain.

We begin with the digestive system and how it operates to aid the human body with digestion of food. Next we will cover disease detection by using various types of endoscopes. We will look at how the current state of lesion detection and how it could be improved by using deep learning.

In the next part we will focus on deep learning and its various architectures and building blocks. To fully understand this we need to have a look at its inner workings and outputs. We begin with looking at a basic three layer fully-connected network and build from there up to Convolutional Neural Networks (CNN) and some of the most advanced architectures recently proposed. This will give a good understanding of how and why we use deep learning to classify medical images.

2.1 Medical scenario

Detecting irregularities in the digestive system (Figure 2.1) is a difficult and time-consuming task, which require expert knowledge. To fully understand the necessity of an automated system for detection lesions in the gastrointestinal tract we will go through the medical aspect of our problem statement, beginning with the anatomical explanation of the digestive tract. Then we will get to know the details of lesions in the small intestine, and the equipment currently in use to observe them.

2.1.1 The digestive system

The digestive system is made up of the gastrointestinal tract (GI tract), and the liver, pancreas and gallbladder. The GI tract is a series of hollow organs joined in a long and twisting tube beginning at your mouth and end with the anus, covering a distance of about nine meters. This is possible because the small intestine is very twisty. The GI tract is controlled by the brain through nerves and hormones. Organs that make up the GI tract is the mouth, esophagus, stomach, small intestine, large intestine and rectum.

The main purpose of the digestive system is so that the cells in the body can extract the nutrients from the food we eat and dispose of the waste which the body can't process. Special cells helps absorb the nutrients and cross the intestinal lining into the bloodstream. The circulatory system carries simple sugars, vitamins, salts, amino acids and glycerol to the liver which processes, stores, and deliver them back into the circulatory system which transports the nutrients to wherever in the rest of the body it is needed. The body uses amino acids, fatty acids and sugars to build substances needed for growth, energy and cell repair for example.

Clinicians commonly divide the gastrointestinal tract in upper and lower regions called upper gastrointestinal tract and lower gastrointestinal tract. The upper gastrointestinal tract consist of mouth, esophagus, stomach and duodenum while the lower gastrointestinal tract consist of most of the small intestine, large intestine and rectum. Each organ in the GI tract helps to move the food and liquid forward throughout the body while its being broken into smaller parts. Next we will explain the function for each organ in the GI tract in the order of which food is processed.

1. **Mouth**; this is where food enters the GI tract and where the digestive process begin. After being split apart by chewing the food is swallowed and enters the esophagus.

2. **Esophagus**; after the swallow the brain signals the esophagus to begin the peristalsis, which is the process of contraction and relaxation of muscles that propagates the food (now called *bolus*, a ball of saliva and food) down towards the stomach. At the bottom of the esophagus you'll find a sphincter which opens to let the food into the stomach and normally keep the fluids in the stomach from traveling back up the esophagus.

3. **Stomach**; upon entering the stomach the stomach muscles begin to mix the bolus with gastric acid which begins the digestion of proteins. The stomach is lined with gastric folds, which helps the stomach to expand to hold about one liter of food. After an hour or two the pyloric valve opens and the contents (called *chyme*, a liquid of partially digested food and acids) are emptied into the small intestine.

4. **Small intestine**; the small intestine mix chyme from the stomach with digestive juices from the pancreas, liver and intestine and push the mixture forward for further digestion. The small intestine is divided into three sections; duodenum, jejunum and ileum. The walls of the small intestine, covered with intestinal villi

(to increase the absorption area), absorb 95% of the nutrients, and carries it to the bloodstream. Whats left, the waste product, move into the large intestine by the peristalsis forces.

5. **Large intestine**; undigested parts of food, fluids and old cells from the GI tract lining enters the large intestine. The large intestine absorbs water, salts, sugars and vitamins back into the blood in the colon and changes the waste from liquid into stool.

6. **Rectum**; the rectum stores stool until it is pushed out of anus during a bower movement.

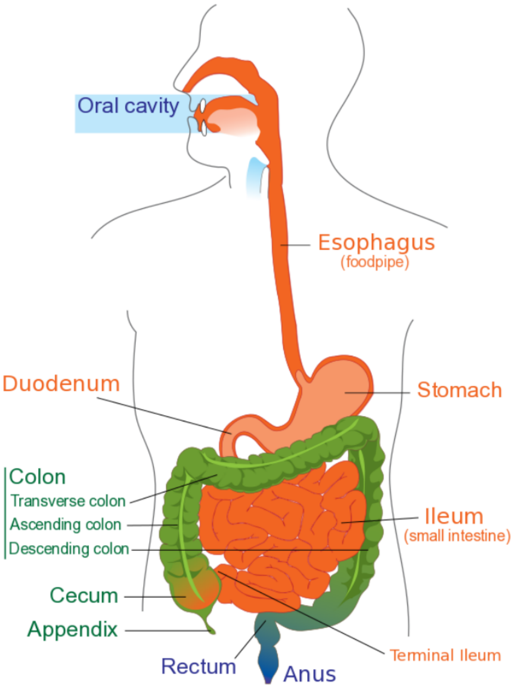


Figure 2.1: An overview of the terms used to describe the digestive system[1].

2.1.2 Colorectal Cancer and Screening

The GI tract may be home to a multitude of diseases, including infections, inflammations and cancers. Given our problem statement and the severity of the disease, we will focus on colorectal cancer (CRC). See Section 3.1 for list of other diseases from the

[1] By Mariana Ruiz, edited by Joaquim Gaspar. Released into public domain by author. `https://en.wikipedia.org/wiki/File:Digestive_system_diagram_edit.svg`

datasets we have used during our experiments. One of the most substantially significant factors for lowering morbidity and mortality in GI tract diseases are early screening and treatment [2], [11]. In this section, we will therefore explain the importance of screening and the difficulties that the current methods inflict upon the medical sector.

A study from 2014 found that CRC were the leading cause of cancer death in the United States in the late 1940 and early 1960 [2], but CRC mortality has since been slowly decreasing due to historical changes in risk factors (E.g decreased smoking and red meat consumption) and better use of screening and early treatment. Today CRC is the third most common cause of cancer death in both men and women.

Another study used a micro-simulation called MISCAN-COLON [12] to simulate the 2,000 U.S population with regards to the CRC risk factor prevalence, screening use, and treatment use. They used the model to project age-standardized CRC mortality from the year 2000 to 2020 for three intervention scenarios and found that without any changes the risk factor would decrease by 17% by the year 2020. However, if the use of screening was improved to 70% of the population and the use of chemotherapy increased for all age groups, then the reduction of CRC mortality was estimated to be close to 50% by the year 2020. They found that the highest contributor to the reduced mortality rate was high level of screening (23%).

At the current state of screening the patient is relying on a doctor's ability to correctly spot early signs of cancer, most commonly polyps (See Figure 2.2), which are abnormal tissue growth often taking the shape of a mushroom. This is a problem as it has been proven that the person who perform the procedure can be more important than the most important health factors, like age and gender [13]. Most screening occurs through endoscopy examinations and is uncomfortable for the patient and require about one medical-doctor-hour and two nurse-hours and cost $1100 USD per person in the US [3]. This could be improved by the use of cheaper screening methods like Video Capsule Endoscopy (VCE), which collect patient data and transfer it back to the hospital, and Artificial Intelligence (A.I), which assist a doctor to diagnose the data.

2.1.3 Traditional Endoscopy

The most common way of screening patients is with a endoscope. When this tool is used by a professional some of the irregularities that can be spotted are; *Colon polyp*, *Colorectal Cancer*, *Ulcerative Colitis*, *Crohn's Disease*, *Familial adenomatous polypsis*, *Diverticulosis* and *Diverticula Bleeding.* See Section 2.1.2 for a more detailed list of diseases.

The basic technology behind the modern endoscope was developed in the early 1950s by English physicist Harold Hopkins and his student Narinder Kapany which let light travel through flexible pieces of glass, now known as optical fibers [14]. These fibers, as many as 50,000 optic fibers, can be packed very dense and allow for light to be transported over long distances with a high resolution. Later iterations of the endoscope allows for recording images through an added camera recorder connected at the end of the tool, water pipes, control cables and operation channels. See Figure 2.3 for a detailed look at the endoscope and its functions.

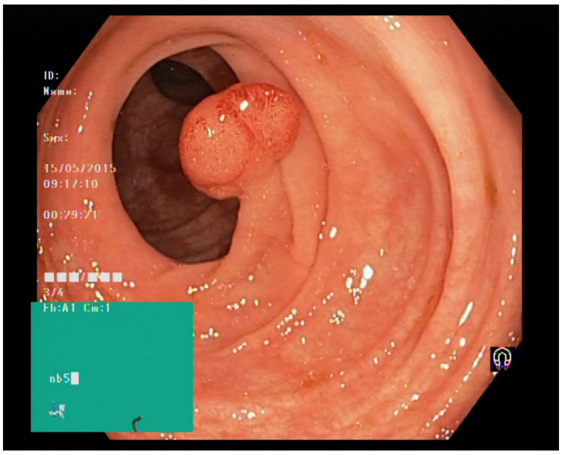


Figure 2.2: Image from Kvasir-V2 dataset of a *polyp* in the colon, taken with a fiber-optic endoscope (Section 2.2.3). The *polyp* is seen in the middle of the image as a reddish mushroom of excess mucosa tissue.

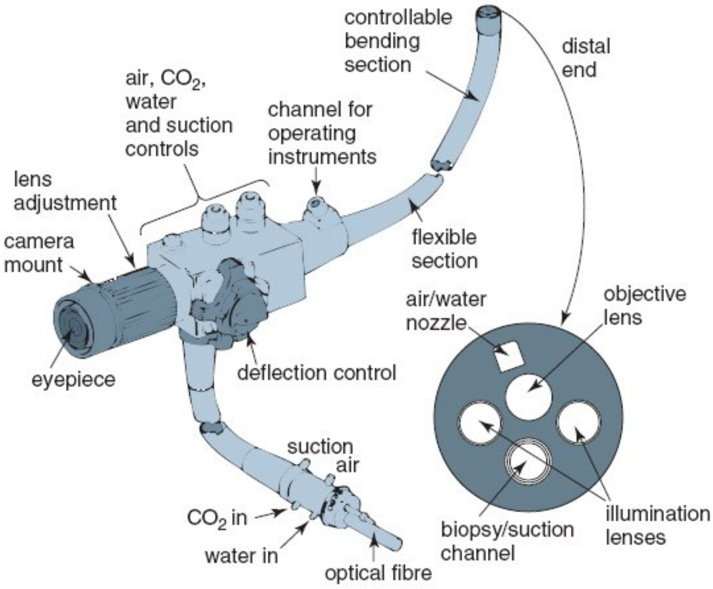


Figure 2.3: Image of a fiber optic endoscope with explanation of different parts of the tool. Bottom right show a cross section of the operating tube with dedicated channels for air/water, light, fibers and tools.[2].

The clever design of the tool allows it to be used for both ends of the GI tract, but also ears, nose and urinary tract. See Table 2.1 for a more detailed list of endoscope

[2] Image credit: Jacaranda Physics 1 2nd Edition © John Wiley & Sons, Inc.

types. There are also some special forms of endoscopy which combines an endoscope with other medical applications, like fluoroscopy and ultrasound, to take medical imaging of special tricky parts of the body.

When the endoscope is inserted into the mouth and throat it is called upper endoscopy and if it is inserted through the anus it is called lower endoscopy.

Upper endoscopy

An upper endoscopy is a procedure used to examine the upper gastrointestinal tract, that is the mouth, esophagus, stomach and duodenum (the beginning of the small intestine). A specialist, called a gastroenterologist, use endoscopy to diagnose and, sometimes, treat conditions that affect the upper part of the digestive system. Upper endoscopy is often performed while the patient is conscious. But sometimes the patient receives a local anesthetic in the form of a spray to the back of the throat, or the patient can be fully sedated. This procedure is sometimes performed in the hospital or emergency room to identify acute bleeding and problems with swallow and breathing.

Lower endoscopy

An lower endoscopy is a procedure used to examine the lower gastrointestinal tract, which is most of the small intestine, the large intestine and the rectum. The procedure may include rectum and entire colon, in which case it is a colonoscopy, or just the rectum and sigmoid colon, then it is called a sigmoidoscopy. Treatments that may be performed in the lower digestive system include biopsy (collecting tissue sample), polyp removal, cauterize a bleeding vessel and other medical procedures.

An endoscopy is usually a safe procedure, and the risk of serious complications is very low. Rare complications are; an infection in the part of the body the endoscope is used, or piercing or tearing in an organ, or bleeding, or reaction to the sedation used.

2.1.4 Wireless Capsule Endoscopy

Before the year 2000 the only option you had to visualize the food pipe, stomach, duodenum, colon and terminal ileum (see Figure 2.1 for details) was to use a fiber-optic endoscope. These cables have to carry fiber optic bundles, water pipes, operations channel and control cables. Although these cables can be quite flexible there is a limit for how far they can advance into the small bowel. This method cause pain and discomfort for the patient, and there was a clinical need for an improved method.

That is why in the year 2000 Iddan *et al.* developed a new type of video-telemetry capsule endoscope which the patients were able to swallow [15], [16]. This capsule can travel through the entire digestive system because it has no external wires, fiber-optic bundles or cables of any sorts. The capsule travels by peristalsis, a radially symmetrical contraction and relaxation of muscles that propagates in a wave down through the gastrointestinal tract. This process takes from 10 to 48 hours. For as long as the battery allows, usually in the range of 6 to 15 hours, the capsule transmits images on a regular interval to eight abdominal receivers and stores the data on a portable solid state

Procedure	**Name of tool**	**Area/organ viewed**	**Insertion point**
Anoscopy	Anoscope	Anus and/or rectum	Anus
Arthroscopy	Arthroscope	Joints	Incision at the joint
Bronchoscopy	Bronchoscope	Trachea, windpipe and the lungs	Mouth
Colonoscopy	Colonoscope	Colon and large intestine	Anus
Colposcopy	Colposcope	Vagina and cervix	Vagina
Cystoscopy	Cystoscope	Inside of bladder	Urethra
Esophagoscopy	Esophagoscope	Esophagus	Mouth
Gastroscopy	Gastroscope	Stomach, duodenum	Mouth
Hysteroscopy	Hysteroscope	Uterus	Vagina
Laparoscopy	Laparoscope	Stomach, liver or other abdominal organs	Incision in the abdomen
Laryngoscopy	Laryngoscope	Larynx	Mouth
Neuroendoscopy	Neuroendoscope	Areas of the brain	Incision in the skull
Proctoscopy	Proctoscope	Rectum and sigmoid colon	Anus
Sigmoidoscopy	Sigmoidoscope	Sigmoid of colon	Anus
Thoracoscopy	Thoracoscope	Pleura	Incision in the chest

Table 2.1: List of the most common types of endoscopy.

recorder, which is carried on the patients belt. Some vendors, of which CapsoVision is one, have opted for a design which uses local flash storage to save the collected images directly on the device and therefore eliminates the need for abdominal receivers and wireless transmission of data. Writing data directly to flash storage has some drawbacks: (1) it is not possible to observe the area being imaged before after the capsule has passed, and (2) the need for a special docking station that enables access to the flash storage.

Endoscopic capsules are divided by terms of their application and is used to diagnose: (1) the esophagus; (2) the small intestine; (3) the large intestine. Depending on application they differ in areas like operating time, imaging frequency and number of cameras. To diagnose the esophagus the capsule travels a short distance in a short time and it is common to use a capsule with cameras on opposite ends, and capture images in high frequency. This comes at a cost of operating time. For a clinician to diagnose the small and large intestine the most significant feature is operating time, and it is therefore common to use a single camera with lower imaging frequency to reduce the drain on battery.

Nomenclature used in the field of capsule endoscopy

We have conducted deep literature searches in the domain of video capsule endoscopy and found that there is different terms commonly in use. The terms are often confused and used interchangeably by different researcher, papers and institutes. Consequently in this thesis, they will all be referred to as VCE. We have chosen this because not all capsules are wireless, and we believe local capsule storage is key to reach our goal of cheap and efficient population screening, more on this in Section 2.1.5. From a variety of literature searches we have found these five different terms in use (summarized in Figure 2.4):

- **VCE**: Video Capsule Endoscopy - capsule endoscopy including an imaging device such as a CCD (the capsule does not have to be wireless);
- **WVE**: Wireless Video Endoscopy (not necessarily a capsule);
- **CE**: Capsule Endoscopy - endoscopic capsule (not necessarily wireless);
- **WCE**: Wireless Capsule Endoscopy (not necessarily containing an image sensor);
- **WVC**: Wireless Video Capsule.

Two example images taken by VCE are presented in Figure 2.5. By triangulating the signal strength and the location of the receivers taped on the body it is possible to roughly estimate the position of the capsule. This is however not very precise and can not tell us the rotation or direction of the capsule. Regardless, that information will not be available for us in this study as we only have access to the images themselves. By looking at some of the anatomical landmarks in the images we still might be able

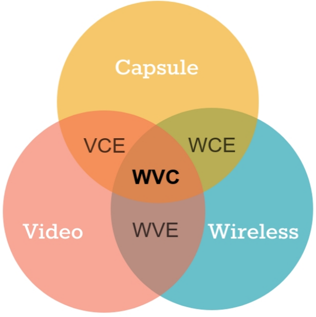
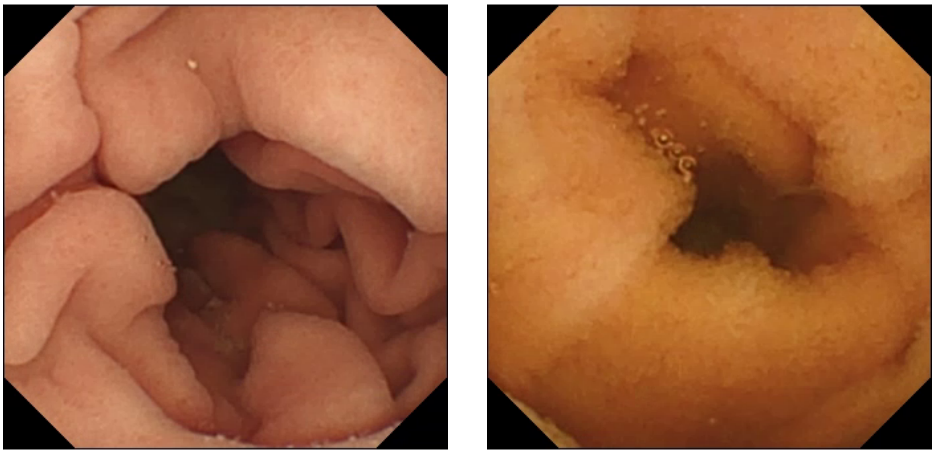


Figure 2.4: Distinction of the nomenclature relating to capsules, wirelessness and video.



(a) Stomach

(b) Small intestine

Figure 2.5: Images from Kvasir-Capsule (See Section 3.1.2) dataset taken with VCE. Figure 2.5a show a image taken from the patients stomach, and Figure 2.5b show a image taken from the small intestine.

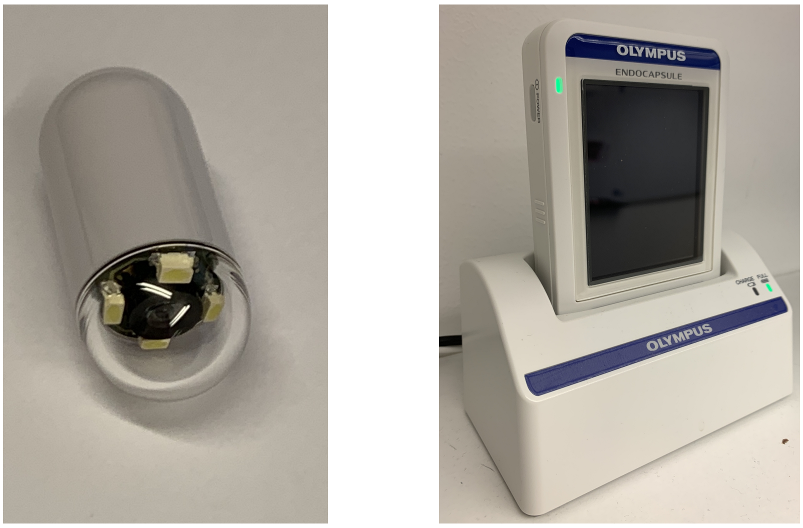
to predict when the capsule exits the stomach through the pylorus, as we are most interested in images taken from the small intestine.

There is ongoing research done in the field of map prediction (see Section 2.6.3) which could be of great interest for VCE technology as it would allow us to better predict the location of a disease inside the patients abdomen, as well as enable the clinician processing the video to see the orientation of the capsule.

VCE devices come in a variety of different versions. Depending on travel speed through the GI tract, the purpose of the device and the localization, it will capture between 1 and 30 images every second, produced with pixel resolution in the range of 256 × 256 to 512 × 512. They are specialized for different parts of the GI tract and are produced by different vendors. The most known manufactures are Given Imaging (Medtronics), Ankon Technologies, Chongqing Science, IntroMedic, CapsoVision and

Olympus.

The data used for this study is collected by the Olympus Endocapsule 10 System[3] using the Olympus EC-S10 endocapsule (Figure 2.6a) and the Olympus RE-10 endocapsule recorder (Figure 2.6b). This system has a 160° wide-angle lens, a light source, a minimum of 12 hours battery life (sometimes up to 20 hours), captures between 80,000 and 140,000 images and user friendly functionalities like Omni-selected Mode. Omni-selected Mode skips images that overlay with previous ones and therefore reduce review time for clinicians. To reduce drain on battery the light source will only emit light just as the camera is taking a picture. Its dimensions are 11 mm (diameter) × 26 mm (length) and it weight 3.3 gram.



(a) EC-S10 endocapsule

(b) RE-10 endocapsule recorder

Figure 2.6: Video capsule endoscopy equipment manufactured by Olympus. Figure 2.6a shows a image of the swallowable capsule, from where our VCE data is taken, and Figure 2.6b show the receiving unit which stores the data transmitted by the endocapsule.

A typical video collected by VCE examination lasts a few hours. A clinician must watch the entire video to make a diagnosis because in a typical clinical situation there is no indication of which part of the GI tract they need to search for damaged tissue, polyps, bleeding, etc. The capsule moves through the tract by two forces, gravity and bowel movements. In the small intestine there is two types of bowel movements: (1) peristaltic and (2) staple (segment). The first type is responsible for transit of food and is pretty linear movement, while the latter is responsible for mixing of food and is therefore much more chaotic in nature. These movements sometimes ceases temporarily as the muscles in the intestine relaxes. The result is an video which is highly diverse with moments of stillness, camera obscured by food debris and moments of chaotic movements. These effects can cause rapid changes in the imaging area. As such, the

[3] `https://www.olympus-europa.com/medical/en/Products-and-Solutions/Products/Product/ENDOCAPSULE-10-System.html`

clinician watching the video will often have to speed up the footage, slow it down, and sometimes watch it frame by frame. Consequently, there is ongoing research related to the implementation of image analysis and processing methods allowing automatic video analysis. Such a automatic analysis system could greatly shorten the time for diagnosis and reduce the cost related to clinician salary. In practice this means that the clinician watch a few minutes of video with the pathologies detected by the software. To understand how such a software could be created we need to take a look at deep learning, which is discussed in the next section.

2.1.5 Remote diagnostic

A topic of which have peaked in interest, as the technology which drives the incremental progress of the VCE engineering and the autonomous video classification tools advances, is the adoption of remote diagnostics. This would, in the near future, allow for patients to use their own devices, such as an iPhone or a tablet, to receive e-medical service from their local doctor or hospital. The patient order a VCE device online, swallows it, and the device will save images to its non-volatile memory which is then transferred to the doctor or hospital through the patients device via the internet. The doctor or hospital uses video analysis tools to further reduce burden on the physician and cost of the procedure itself. As no doctor-nurse-hours are spent during examinations, and the endoscopist can use computer-aided assistance for diagnosing the patient video, this has the potential to make the public health care system more scalable and cost effective.

2.2 Datasets

The datasets used in our experiments are Kvasir v2 [17] and Hyper-Kvasir [6]. This section will demonstrate the main differences between the two datasets and explain how they can be found and used for fact checking. Both datasets are collected using endoscopic equipment at Vestre Viken Health Trust (VVHT) in Norway. The VVHT consists of 4 hospitals and provides health care for 470,000 people. One of the hospitals is Bærum Hospital, which has a large gastroenterology department from where the data is collected.

We will also go through some other publicly and restrictively available datasets, and explain why there is a need for a novel wireless video endoscopy capsule dataset. We will introduce Augere Medical, and their tagging tool implementation which we have used to label our VCE videos. In the later part of this section we will discuss some of the difficulties of the aforementioned datasets.

In Section 3.1.2 in the next chapter, we will introduce how we gathered data and created a new VCE dataset, Kvasir-Capsule [5].

2.2.1 ImageNet

Deng, Dong, Socher, *et al.* introduced a database called ImageNet in 2009. This database was coined for the academic world of researchers and students, to be used for visual object recognition and classification software research. It contains more than 14 million hand-annotated images in more than 20,000 classes. Each category contain from a hundred samples to some thousand. Since 2010, the ImageNet project have held a yearly competition, the ImagNet Large Scale Visual Recognition Challenge (ILSVR), where teams compete to create a classifier which can correctly classify as many images as possible.

The reason we introduce the ImageNet database is because of its widespread usage in the academic world. State of the art models often compare their results on the open source ImageNet database, and release the weights created by the model. These weights contain the image features learned by a specific network, and can be transferred to a different network to stimulate better and faster learning.

2.2.2 Available endoscopy datasets

There is a great number of publicly available endoscopy datasets online, and some that are restricted. To further improve detection rates in automated gastrointestinal analyze tools there is a demand for large amounts of data for different use cases, and since medical data often is scarce, or restricted, we introduce Kvasir-Capsule dataset, currently in development. This dataset is among the few publicly available VCE datasets, see Table 2.3 for an overview. Traditional colonoscopy have been around for longer and have been under more research. Therefore colonoscopy datasets are easier to find publicly, see Table 2.2 for a list of these datasets. This can benefit the ongoing automated VCE analysis as deep learning models can be tested and pretrained on them.

2.2.3 Kvasir-V2

The Kvasir multiclass dataset [17] contains images from inside the gastrointestinal (GI) tract. The samples are classified into three important anatomical landmarks and three clinically significant findings. In addition it has two classes related to the removal procedure of polyps. The dataset is sorted and annotated is performed by medical doctors. The class names and findings for each class is given in Table 2.4. One of the most important aspects of the Kvasir dataset is that it makes it easy to reproduce and compare results in scientific computing.

The dataset consists of 4,000 images, annotated and verified by experienced endoscopists. In Table 2.4 we have listed all eight classes from anatomical landmarks, pathological findings, and endoscopic procedures. Each class have 500 samples, and are explained in more medical detail in Table 2.5 together with the classes from Hyper-Kvasir as all Kvasir v2 classes are a subset of Hyper-Kvasir dataset.

[4] https://www.endoatlas.net/ea/AtW01/106.aspx

[5] http://www.endoatlas.org/index.php

[6] http://www.gastrolab.net/index.htm

Dataset Name	Data Source	Findings	Size	Status	Description
CVC-ClinicDB [19]	Colonoscopy	Polyps	612 still images from 29 different sequences with polyp mask	Available	From 29 different sequences with polyp mask (ground truth)
ASU-Mayo Clinic Colonoscopy Video DB [20]	Colonoscopy	Polyps	20 videos for training and 18 for testing	Copyrighted	10 videos with polyp detection, 10 videos without polyps, GT available
CVC colon DB [21]	Colonoscopy	Polyps	300 frames with ROI	By explicit permission	15 short colonoscopy sequences (different studies)
ETIS-Larib Polyp DB [22]	Colonoscopy	Polyps	196 images	By request	196 images with GT
GI Lesions in Regular Colonoscopy Data Set [23]	Colonoscopy	GI lesions	76 instances	Available	15 serrated adenomas, 21 hyperplastic lesions, 40 adenomas
The Atlas of Gastrointestinal Endoscopy[4]	Endoscopy	GI lesions	2259 images	Available	Esophagus, Stomach, Duodenum and Ampulla, Capsule Endoscopy, Inflammatory Bowel Disease, Colon and Ileum and some Miscellaneous
WEO Clinical Endoscopy Atlas[5]	Endoscopy	GI lesions	152 images	By explicit permission	One image per lesion
GASTROLAB[6]	Endoscopy	GI lesions	Several hundreds of images and several tenths of videos	Discontinued	Partially damaged and unavailable dataset
Kvasir-V2 [17]	Various	GI lesions & landmarks	8,000 images, 8 classes, 1,000 images per class	Available, public, free for research and educational purposes	See Section 2.2.3 for the description
Hyper-Kvasir [6]	Endoscopy	GI lesions and landmarks	10,662 labeled images, 373 videos and 99,417 unlabeled images	Available, public, free for research and educational purposes	See Section 2.2.4
Nerthus [24]	Colonoscopy	GI cleansing	5,525 frames extracted from the 21 videos, 4 classes, from 500 to 2,700 frames per class	Available, public, free for research and educational purposes	Bowel preparation dataset
Medico [25]	Various	GI lesions, landmarks and findings	14,033 images, 16 classes, from 4 to 2,331 images per class	Available, public, free for research and educational purposes	Heavily unbalanced

Table 2.2: Existing colonoscopy image and video datasets taken from the GI tract. These datasets may not include all available datasets, but were the ones we could find after an extensive literature search.

Name	Findings	Size	Status	Description
KID [26]	Angiectasia, bleeding, inflammations, polyps	2,500+ images + 47 videos	Discontinued	Open academic
GIANA'17 [27]	Angiectasia	600 images	Available, by request	Includes ground truth segmentation masks
CAD-CAP [28]	Normal, Vascular Lesions and Inflammatory Lesions	25,000 images	Discontinued	Available by request
Kvasir-Capsule [5]	GI lesions, landmarks and findings	44,260 images with ROI, 13 classes and 2.6 million unlabeled images	Available, public, free for research and educational purposes	Ours, See Section 3.1.2 for detailed description.

Table 2.3: An overview of VCE datasets from the GI tract which were discovered during our extensive literature search. Kvasir-Capsule is ours dataset.

Type	Name
Anatomical landmark	Z-line Pylorus Cecum
Pathological finding	Esophagitis Polyps Ulcerative colitis
Polyp removal	Dyed and lifted polyp Dyed resection margins

Table 2.4: Kvasir-V2 classes grouped into anatomic landmarks, pathological findings, and endoscopic procedures.

2.2.4 Hyper-Kvasir

The Hyper-Kvasir dataset [6] is one of the largest medical datasets available, containing 110.079 images and 373 videos of anatomical landmarks and pathological findings, as well as normal GI tract images. Resulting in more than 1.1 million images and video frames all together. The dataset contain four parts, labeled images, unlabeled images, segmented images and lastly, videos. In total the dataset is 70 GB in size, but can be downloaded and stored in parts from Simula[7].

All the data is fully anonymized and approved by Privacy Data Protection Authority, and all experiments were performed in accordance with the relevant guidelines and regulations of the Regional Committee for Medical and Health Research Ethics - South East Norway, and the GDPR.

[7] `https://datasets.simula.no/hyper-kvasir/`

The Hyper-Kvasir dataset is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaption, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original authors and the source.

Labeled images

Hyper-Kvasir contains 10,662 labeled images. The images are split into 23 different classes, and are stored in a folder with the same name as its corresponding class. All of the images are stored in JPEG format [34], which means it has some image quality loss but quite insignificant compared to the reduction in file size. Like in situations most often encountered the classes has a different number of samples, this is a challenge in the medical field because some findings occur more often than others. In Table 2.5 we have summarized the classes and their description, organized after which part of the GI tract the images are taken.

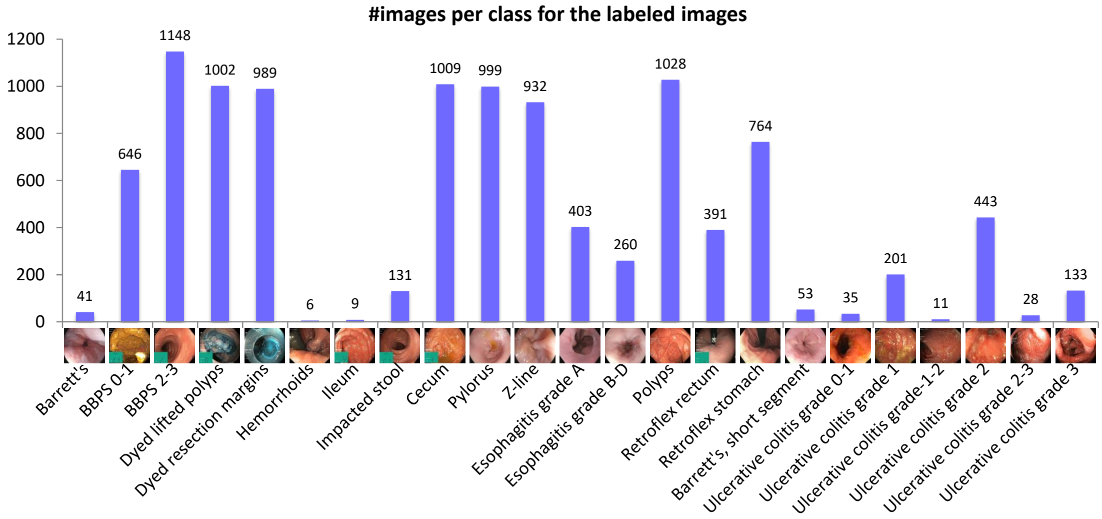


Figure 2.7: Number of labeled samples for each of the 23 classes in Hyper-Kvasir dataset. Under each bar is one example image of that class.

Unlabeled images

This part of the dataset contains 99,417 unlabeled images. When extracted they can be found in a separate subfolder. The images are accompanied with extracted global features and clusters assignments in Hyper-Kvasir GitHub repository[8].

[8] https://github.com/simula/hyper-kvasir

GI location and type	Class	Description
Landmarks in upper GI	normal-z-line	the anatomical junction between the squamous epithelium of the esophagus and columnar epithelium of the stomach.
	retroflex-stomach	the endoscope is retroflexed, looking back to visualize the cardia and fundus in the upper parts of the stomach.
	pylorus	the anatomical junction between the stomach and duodenal bulb.
Findings in upper GI	esophagitis-a	Reflux esophagitis is an inflammation mostly affecting the lower third of the esophagus, near the Z-line. Graded A to D by severity according to the Los Angeles (LA) classification [29]. (A) mucosal breaks shorter than 5mm.
	esophagitis-b-d	(B) mucosal breaks longer than 5mm that does not extend between the tops of two mucosal folds, (D) one (or more) mucosal break that is continuous between the tops of two or more mucosal folds and involves more than 75% of the circumference.
	barretts-short-segment	Barrett's esophagus represents a metaplastic transformation of the squamous epithelium of the esophagus into a gastric like columnar epithelium and are graded according to the Prague classification [30].
	barrets	short segment is characterized by a longitudinal extension of less than 3 cm.
Landmarks in lower GI	ileum	the distal 2/3 of the small bowel, recognized by visible intestinal villi.
	cecum	the proximal end of the large bowel and is characterized by the visualization of the appendiceal orifice and the ileo-cecal valve.
	retroflex-rectum	the endoscope is retroflexed in the rectum to visualize the dentate line and the circumference of the proximal orifice of the anal canal.
Bowel cleansing lower GI	bbps-0-1	the degree of bowel cleansing during a colonoscopy is described by the Boston Bowel Preparation Scale (BBPS) [31]. (0) no mucosa seen due to solid stool; (1) portions of the mucosa of the colon segment seen.
	bbps-2-3	(2) minor amount of residual staining, small fragments of stool and/or opaque liquid; (3) entire mucosa of colon segment seen well with no residual staining, small fragments of stool or opaque liquid.
	impacted-stool	Sometimes stool is impacted in these diverticula and may increase the risk of diverticulitis.
Findings in lower GI	ulcerative-colitis-grade-1	a chronic inflammatory bowel disease often debuting in the twenties. Classified according to the Mayo Score [32]. (1) Mild with erythema, decreased vascular pattern, mild friability.
	ulcerative-colitis-grade-2	(2) Moderate with erythema, absent vascular pattern, mild friability, erosions.
	ulcerative-colitis-grade-3	(3) Severe with spontaneous bleeding, ulceration.
	ulcerative-colitis-0-1	classes in-between the Mayo Score, where it is difficult to determine the exact class. (0) Inactive where the mucosa only has normal vascular patterns.
	ulcerative-colitis-1-2	findings in-between grade 1 and 2.
	ulcerative-colitis-2-3	findings in-between grade 2 and 3.
	polyps	most frequently neoplastic lesions of the large bowel. They have mainly three different shapes, protruding in the lumen, flat or excavated according to the Paris Classification [33].
	dyed-lifted-polyps	images of polyps lifted with submucosal injection using a solution containing indigo carmine.
	dyed-resection-margins	the resection margins and site appears blue due to the indigo carmine solution, after resection of dyed polyps with a snare.
	hemorrhoids	pathologically swollen veins in the anus or lower rectum.

Table 2.5: Hyper-Kvasir types, class names, and their corresponding medical description.

Segmented images

Hyper-Kvasir includes images with corresponding segmentation masks and bounding boxes for 1,000 images from the polyp class. The segmentation masks depicts the polyp tissue for the corresponding image pixel. The bounding box is defined as the outermost pixels of the found polyp, and for the corresponding images are stored in a JavaScript Object Notation (JSON) file. The Region of Interest (ROI) are represented by the white mask while the black does not contain polyp pixels. In Figure 2.8 we can see an example of the segmented Kvasir images, with both the original image and the segmentation mask for that same image.

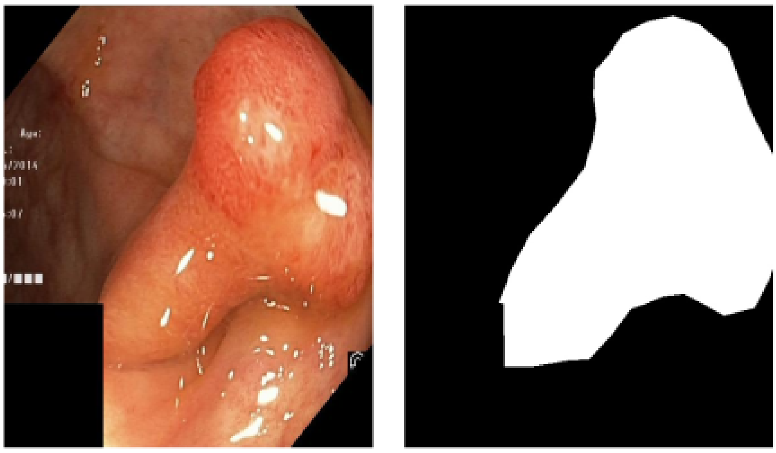


Figure 2.8: Example of a segmented image from Hyper-Kvasir dataset This figure in particular show a image from the *polyp* class, and its segmented mask is shown in the right image.

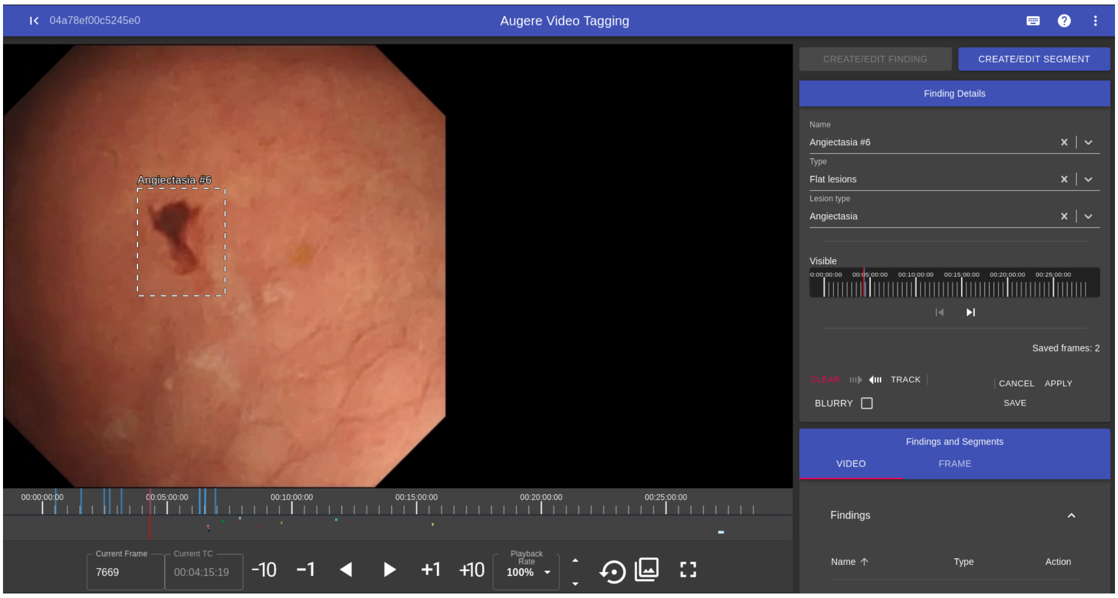
Videos

In total there are 373 videos provided in the dataset, corresponding to 11.62 hours of videos and about 1 million video frames that can be converted to images. The file format for the videos are Audio Video Interleave (AVI). The video portion of the dataset is 38.6GB in file size. In addition to the video folder there is a Comma-Separated Values (CSV) file provided, containing the videos IDs and findings. Video ID contains the corresponding video file name, and the finding contain the description of the pathological finding in the video.

2.2.5 Augere Medical AS

Augere Medical As is a team of technologists and physicians dedicated towards reducing variability between physicians based in visual observation, use deep learning to increase early detection rates and finally to reduce the cost of health care and less aggressive interventions and treatments.

As a shared partner with Simula we were allowed to utilize their in-house developed tagging tool for medical videos. This tool allows for real-time playback of VCE videos, as well as to go frame by frame. We annotate each finding for numerous videos with bounding boxes which specify pixel coordinates for the relevant finding. Then we tag the rest of the videos as unlabeled so they can be separated upon export from the tagging tool. In Figure 2.9 is a screen capture of the tagging tool in use.



04a78ef00c5245e0

Augere Video Tagging

Name: Angiectasia #6

Type: Flat lesions

Lesion type: Angiectasia

Saved frames: 2

BLURRY □

Current Frame: 7669

Current TC: 00:04:15:19

Playback Rate: 100%

Figure 2.9: Augere Medical tagging tool screen capture. The current frame show a angiectasia and its respective bounding box. Bottom of the image show various buttons for going frame-by-frame, backwards, skipping frames etc. The option bar at the right show settings for marking a frame as a finding or video segment. The tool also display a timeline-like bar for showing where in the videos there are markings and allow for ”scrubbing” through portions of the video.

2.2.6 Class imbalance in dataset

Class imbalance typically refers to a problem with classification problems where the classes in the models training data are not represented equally. In the medical imaging domain this is very common for two reasons: (1) there are a lot more healthy patients than there are sick ones, and (2) for a specific unhealthy patient, there are a lot more images of healthy mucosa than there are images of pathological findings. As an example you could have a VCE video which has been carefully analyzed by a clinical expert and each frame is tagged with either being in class 1; healthy, or class 2; unhealthy. Of this dataset 10,000 frames have no findings, and 50 of them have confirmed pathological findings. You could then use this data to train a model to have 99% accuracy which sounds great, but in reality the model only achieves these impressive results because it classifies all the data as class 1; healthy.

Imbalanced dataset pose a challenge for predictive algorithms as most learning algorithms are based on the assumption of an equal number of samples for each class. This results in models that have poor predictive performance, especially for minority class or classes. This is a great problem to overcome because in many medical datasets the minority class is the most important and therefore more sensitive for classification errors.

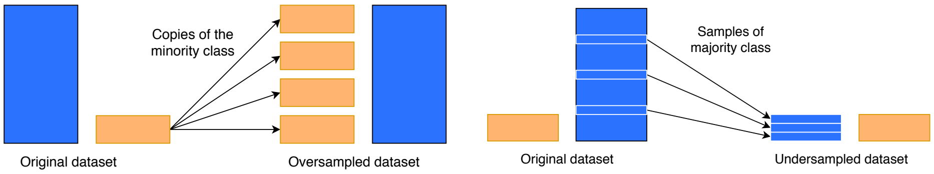
There are different methods to combat this class imbalance problem:

- Collect more data.
- Changing the performance metric.
- Resample the dataset.
- Introduce class penalties.

It is a laboring, time consuming and expensive task to collect more data for medical image classifications projects because a clinical expert is required to manually validate the data.

Dataset re-sampling

In Figure 2.10 are two naive but effective methods of handling class imbalance; oversample and undersample. Oversampling repeatedly sample data from the minority class until reaching the desired amount, while undersampling sample the dataset same amount of data from the majority class to match the minority class [35], [36]. Oversampling can cause overfitting when used with traditional Machine Learning (ML) methods like linear classifiers [36], but in the setting of deep learning, oversampling shows better compatibility than undersampling which is missing a lot of information from the unused data [37].



(a) Oversampling

(b) Undersampling

Figure 2.10: Two popular resampling methods. Oversampling (2.10a) make copies of the minority class until all classes have the same amount of samples, and undersampling (2.10b sample from the majority class until it reaches the same amount of samples as the minority class.

Weighting the classes

Class weights balances our data by altering the weight that each training example carries when computing the loss. Like dataset sampling, it is useful if the dataset consist of few, but otherwise important images for a minority class. We want the minority class to hold more weight because otherwise it won't effect the weights of the model during training, and the model will instead put more emphasis on the data which is easy to classify.

The main advantage of using class weights to compute the loss over re-sample the dataset is that instead of duplicating or removing images we only need to assign the class with a weight.

2.3 Deep learning

As apposed to using regular optic-fiber endoscopy, it can be difficult to know the location and orientation of the capsule when it is traveling through the digestive system. In a paper by Zou *et al.* it is shown that by using Deep Convolutional Networks (DCNN) it is possible to classify the digestive organs in wireless capsule endoscopy with about 95% classification accuracy on average [38]. The DCNN-based VCE digestive organ classification system is constructed of three stages of convolution, pooling and two fully-connected layers. This is illustrated in Figure 3 in the paper [38]. The main steps of this convolutional neural network are described in detail in Section 2.3.2.

2.3.1 Machine learning types

In deep learning it is common to differentiate between three types of machine learning models, supervised learning, unsupervised learning and reinforcement learning. In this section, we will go through them and explain how they function and which use cases suites them best. In addition we will introduce a combination of supervised and unsupervised learning, called semi-supervised learning.

Supervised learning

The first category of machine learning is supervised learning. If you imagine yourself work under supervision of a leader or boss, it would mean someone is present and judging whether you are doing the correct work. Similarly to this, when a learning algorithm is under supervision is has a fully labeled dataset to work on; continuously updating the algorithm whether the answer is correct or wrong after every test.

Fully labeled dataset means that for every sample in the dataset, it is known what the true answer is to the problem at hand. As an example; if the dataset is images to classify you can think of it as having the correct answer written on the back of the image, but the algorithm will pick up the image front side up, and not look at the correct answer until after making a prediction.

This method is best suited for classification problems and regression problems, where there is a set of available reference points or a ground truth with which to train the algorithm, but this is not always accessible, or too expensive to create.

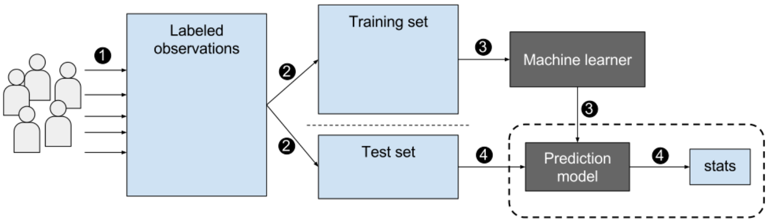


Figure 2.11: Workflow of supervised learning; 1. the dataset is labeled by observers; 2. the samples is split into training and test sets; 3. algorithm is learning on the training set; 4. checking the predictions on the 'unseen' test set to understand how the model performs.

Unsupervised learning

Large, cleanly labeled datasets are not always easy to come by. And sometimes the answers we are looking for are not discrete, but discontinuous and hard to define. This is where unsupervised learning comes in.

In unsupervised learning, the algorithm is handed non-labeled data without any instructions on what to do with it. It is the algorithms job to automatically find which features that best separates the data and find a structure within it. An example of a problem well suited for unsupervised learning is; using anomaly detection to discover unusual data points in a dataset, like fraudulent bank transactions.

It is common to further categorize unsupervised learning into four additional groups;

- **Clustering**: The deep learning model looks for data that are similar to each other and group them together.
- **Anomaly detection**: Used to flag outliers in a dataset. Samples that does not fit well in with the rest.
- **Association**: The model looks at how a certain feature of a data sample correlates with other features.
- **Auto-encoders**: Auto-encoders take input data, compress it into code and then try to recreate that same input data only using the compressed code.

Since the training data has not been reviewed by a human beforehand it is difficult to say with certainty how good the final model perform like it is with supervised learning. But for problem areas where there is little to none labeled data it is a valuable tool.

Semi-supervised learning

This is not its own category, but a combination of the two categories just mentioned. It is good for dealing with problems where you have some labeled data and a lot of unlabeled data.

Many real world problems fall into this problem as large, fully labeled datasets are difficult to obtain. To create one is both expensive and time consuming and often require domain experts like analysts or doctors. Whereas unlabeled data is cheap and easy to collect and store.

Our problem is in this realm and is therefor also a good example of a semi-supervised problem. We have a relatively small dataset of labeled medical images and almost an unlimited quantity of unlabeled images.

The goal of the semi-supervised machine learning technique is to make best predictions on unlabeled data. This is done by first using a trained supervised model to best predict unlabeled data and then feed that back into the supervised learning algorithm as training data. Then use the newly trained model to make predictions on the new unseen data. To get the best result this process can be repeated until accuracy converges.

Reinforcement learning

In Reinforcement Learning (RL) algorithms learn how to react to the environment on their own and is neither supervised nor unsupervised. Instead the algorithm rely on being able to monitor response of its action and measure against a defined "reward".

Reinforcement learning is a type of machine learning where AI agents are attempting to find the optimal way through an environment, to accomplish a set goal or to improve on a specific task. As the agent take an action in the environment it receives a reward, as seen in Figure 2.12. If the action improved on the last agent state it gets a positive reward and if the new state of the agent is worse than the previous it get a negative reward. The goal is to predict which next step to take to get the biggest final reward.

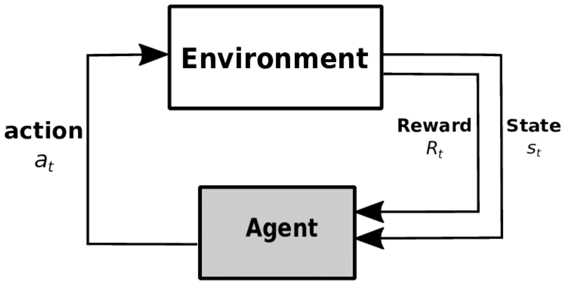


Figure 2.12: Reinforcement learning: Agent attempts to find the optimal way through an environment, and receives a positive reward if the action was led the agent to a better position and a negative reward if t he new position is worse than the initial.

To make these predictions the agent need to rely on what it has previously learned, and be able to explore uncharted territory. For example if the first option for the agent is to pick a left or right turn on a road which leads to two different cities, and it gets a positive reward for picking left, the agent will never explore the other city. Therefore the agent must try to maximize the cumulative reward and not only the immediate reward.

To achieve good cumulative reward the algorithm must iterate over the problem many times. For each iteration, and each round of feedback, the agents strategy incrementally improves. This works really good for problems that can be simulated, where iterating the problem only cost computer power. A good example of a good RL problem is video games and autonomous driving.

2.3.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) is a category of neural networks which has proven to be very efficient in image recognition and classification. CNN have a wide application in image and video recognition due to its two main characteristics; feature extraction and classification. The CNN method was first published by Lecun *et al.* in 1998 [39], yet his work went mostly unnoticed for fourteen years before it was proven to be of a great value. The publicly accepted implementation of the CNN was first seen during *ImageNet Computer Vision competition* where this method managed to achieve an accuracy of 84.7% when classifying 1,2 million images from 1,000 distinct classes with top-1 and top-5 error rates of $37,5\%$ and $17,0\%$ respectively, which far surpassed all other models at the time [40]. Today, the CNN is widely used and has by far surpassed human level of performance in classifying images [41]–[43].

The four most common building blocks in a Convolutional Neural Network: convolution layer, non linearity, pooling layer, and finally a fully connected layer. In the next sections, we will explain their respective function and purpose. These might be the most important building blocks of a CNN, but to build a network which competes with the best in class networks it is required to use more complex methods like dropout, batch normalization, skip layers and more. Due to the scope of our thesis we have not chosen to dedicate own sections for them.

Convolution layer

The first step in a convolutional neural network is to extract features from the input image. This is done to preserve the relationship between pixels by learning image features using filters, or *kernels*. As a result, the network learn filters that activate when it detects some specific patterns or features.

The convolution of f and g is written as $f * g$, and is defined as the integral of the product of the two functions after one (usually the filter) is reversed and shifted.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.1}$$

Non Linearity (ReLU)

Rectified Linear unit function, known as simply ReLU, is an activation function represented by equation (2.2). It sets all negative numbers to zero, by discarding them from the activation map entirely. In this way, ReLU increases the nonlinear properties of the decision function and thus of the overall network without affecting the receptive fields of the convolution layer.

$$ReLU(x) = max(0, x) \tag{2.2}$$

Pooling layer

Pooling layers are applied to reduce the number of parameters when the images are considerably large. Spatial pooling, or merely down sampling, reduces the dimensionality of each image but it keeps the important information. The most used down sampling is max pooling. It extracts the largest element from the rectified feature map and thus reduces computational complexity of the algorithm. In addition average pooling is also frequently used, this method computes the average value of the input map. The input-output model is denoted as:

$$y_i = f(pool(x_i)) \tag{2.3}$$

Fully-connected layer

In a FC-layer every neuron in one layer is connected to every neuron in the previous layer. It is here the high-level reasoning is done. The activation function in the neurons is a *sigmoid* or *tanh* function.

$$f(z) = \frac{1}{1 + exp(-z)} \quad \text{or} \quad f(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.4}$$

At the end of FC-layer we have an activation function such as softmax (equation 2.7) to calculate probability of the predicted classes.

Feed Forward

In the feed forward algorithm input image will be processed through all the layers in the neural network. The first layer will be a convolution layer, containing K filters F_i^1, $i = 1, ..., K$, of size $k \times k$ and a bias b^1. The image will be convoluted with each filter, and the bias is added.

$$\hat{z}_i^l = I * \hat{F}_i^l + b^l, \tag{2.5}$$

where $*$ (asterisk) is the convolution operator in equation 2.1. The final output of each convolutional layer l is a^l,

$$\hat{a}_i^l = f(z_i^l), \tag{2.6}$$

where f represents the ReLU activation function. After going through the convolution layer, the next layer could be a pooling layer, which will reduce the spatial dimensionality either by using the max value or the average value. Before getting our final output $\hat{y}$, we need to collect the outputs from all the filters, which will be an input to a fully connected layer. The fully connected layer use the softmax activation function to classify the input image, much like a neural network would. The softmax function is an accepted standard probability function for a multiclass classifier [44]. The total sum of the probabilities will always add up to 1 when using softmax.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, ..., K. \tag{2.7}$$

To calculate the error of the forward propagation it is common to use cross-entropy error function.

$$C(\hat{y}) = -\sum_{i=1}^{N} t_i log(y_i) \tag{2.8}$$

Backpropagation

Starting from the last layer L, we calculate the derivative of the loss function (function 2.8) with regards to the activation function in order to update the weights. Computing the gradient of the loss function yields

$$\frac{\partial C}{\partial y_i} = -\frac{t_i}{y_i} \tag{2.9}$$

We also require the gradient of the output of the final layer y_i with regards to the input z_k^L of the activation function (equation 2.7)

$$\frac{\partial y_i}{\partial z_k^L} = \begin{cases} y_i(1 - y_i), & i = k \\ -y_i y_k, & i \neq k \end{cases} \tag{2.10}$$

Now with regards to z_i^L

$$\begin{aligned} \frac{\partial C}{\partial z_i^L} =& \sum_{k}^{N} \frac{\partial C}{\partial y_k} \frac{\partial y_k}{\partial z_i^L} \\ =& \frac{\partial C}{\partial y_i} \frac{\partial y_i}{\partial z_i^L} - \sum_{k}^{N} \frac{\partial C}{\partial y_k} \frac{\partial y_k}{\partial z_i^L} \\ =& -t_i(1 - y_i) + \sum_{k \neq i} t_k y_i \\ =& y_i - t_i \end{aligned} \tag{2.11}$$

And finally with regards to the weights

$$\frac{\partial C}{\partial w_i j^L} = (y_i - t_i) a_j^{L-1} \tag{2.12}$$

where $\hat{a}_j^{L-1}$ is the vectorized output from the previous layer. From here, we will propagate the error throughout the layers. The error with regards to the input a_i^L to the fully connected layer is:

$$\delta^{L-1} = \frac{\partial C}{\partial a_i^L} = \sum_i^N (y_i - t_i) w_{ji}^L \tag{2.13}$$

Thus the error is propagated backwards through each layer. If max pooling was used in a pooling layer, the error will only be propagated to the input that had the highest value in the forward pass. The other values will be set to zero. If average pooling was used, the error is averaged in the backwards pass. In equation 2.13 a^l is the output of a convolutional layer l. Since a convolutional layer is always preceded and followed by a activation layer, the input to layer l is $a^{l-1} = \sigma(z^l)$. Now consider the error with regards to z^l.

$$\begin{aligned}
\delta_{ij}^l &= \frac{\partial C}{\partial z_{ij}^l} \\
&= \sum_i^{\prime} \sum_j^{\prime} \frac{\partial C}{\partial z_{i'j'}^{l+1}} \frac{\partial z_{i'j'}}{\partial z_{ij}^l} \\
&= \sum_{i'} \sum_{j'} \delta_{i'j'}^{l+1} \frac{\partial(\hat{W}\sigma(z^l) + b^{l+1})}{\partial z_{ij}^l} \\
&= \delta^{l+1} * ROT180(w^{l+1})\sigma'(z^l)
\end{aligned} \tag{2.14}$$

Having found the error, the gradient of the cost function with regards to the weights is

$$\frac{\partial C}{\partial w_{ij}^l} = \delta_{ij}^l * \sigma ROT180(z_{ij}^{l-1}) \tag{2.15}$$

2.3.3 Gradient descent optimization algorithms

Gradient descent is one of the most used algorithms to perform backpropegation optimization, especially in the case of neural networks. In this section, we are going to look at the different variants of gradient descent, as well as introducing the most common variants.

Gradient descent is the process of minimizing the objective function $J(\theta)$ parameterized by a models parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ with regards to the parameters. The learning rate determines the length of the step towards the minimum. In layman terms we follow the slope of the surface (which is created by the aforementioned objective function) downhill until the only way to go is up.

There are three variants of gradient descent. The difference between the three is how much data is being used when computing the gradient of the objective function. When we use a lot of data to compute the gradient we get a good accuracy but at the cost of computational complexity which again leads to longer time to perform an update.

- **Batch Gradient Descent**: all training samples are used to create one batch.
- **Stochastic Gradient Descent**: batch size is one element of the training data.
- **Mini-Batch Gradient Descent**: batch size is more than one sample and less than the size of the entire dataset

Batch gradient descent computes the gradient of the cost with regards to the parameters θ for the entire dataset with all its samples:

$$\theta = \theta - \eta \cdot \nabla J(\theta) \tag{2.16}$$

where η is the learning rate. Because we compute the gradients for the entire dataset this is slow and consume a lot of memory. This makes it problematic for image classification tasks.

Stochastic gradient descent (SGD) is performs a parameter update for each sample in the dataset, not the entire dataset. It is computed by:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^i; y^i) \tag{2.17}$$

where x^i and y^i are one training sample. This sample-wise update of gradients leads to a lot of redundant computations, but unlike batch gradient descent it does one update at a time and is therefore much faster.

Mini-Batch gradient descent is a combination of batch gradient descent and stochastic gradient descent. It performs one update for every mini-batch of training samples:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.18}$$

where n is the number of samples in every mini-batch. The benefits of using mini-batches is twofold; (1) it reduces variance of parameter updates as gradient are averaged over multiple samples, which can lead to more stable convergence; and (2) very computationally efficient compared to batch gradient descent and stochastic gradient descent.

However, mini-batch gradient descent comes with some challenges. It can be difficult to chose a suitable learning rate. Low learning rate leads to slow convergence and too high learning rate can cause the loss to skip over the minimum. To combat this one can use learning rate schedules which adjust the learning rate during training according to a pre-defined set of rules. But these schedules must also be tuned manually to achieve good performance and fail to adapt to the next dataset. Another problem is that even when setting a good learning rate scheduler the learning rate applies equally to all parameter updates. In the case of a highly imbalanced dataset it is better to tweak how much and how often the parameters is updated.

Momentum based learning algorithms

Momentum [45] fixes one of the main problems with normal SGD, which is that is will often get stuck in in saddle points [46] and converge slowly in ravines. Momentum accelerate SGD in the correct directions and dampens oscillations. These oscillations is common in ravine-like areas where instead of moving parallel to the walls in the ravine it will oscillate perpendicular while only taking small steps in the correct directions. The way momentum does this is by adding a fraction γ of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \tag{2.19}$$

$$\theta = \theta - v_t \tag{2.20}$$

The fraction γ is usually set to a value in the range of 0.9. Using momentum almost always leads to a better and faster convergence than in the case of using standard SGD.

Another version of SGD with momentum which has gained a lot of popularity is called Nesterov momentum, or Nesterov Accelerated Gradient (NAG) [47]. NAG add a notion of in which direction is best for the next update. In the case that we are converging fast on a local minimum normal momentum might have enough momentum to overshoot the minimum, and there is a need for a way to slow down when before the hill slopes up again. NAG achieves this by computing $\theta - \gamma v_{t-1}$ which gives an approximation of the next position of where the parameters are going to be. This allows NAG to look ahead by calculating the gradient with regards to future position of parameters:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \tag{2.21}$$

Adaptive gradient based learning algorithms

Adagrad (ADAptive GRADient algorithm) [48] improves upon stochastic gradient descent with momentum with an adaptive learning rate tailored to the parameters. For features in the mini-batch which occurs frequently it will perform smaller updates (lower learning rate) to the parameters and for features which occurs rarely it will perform larger updates. This makes Adagrad algorithm good for handling imbalanced datasets and sparse data. The way Adagrad calculates the learning rates for every parameter at every time step is by:

$$\theta_{t+1} = \theta_i - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{2.22}$$

where G_t is a diagonal matrix witch holds the sum of the squares of the past gradients with regard to all parameters θ along its diagonal, ϵ is a small value to avoid division by zero and g_t is the gradient at time step t, $g_t = \nabla_\theta J(\theta_t)$.

Momentum & adaptive gradient based learning algorithms

One flaw with Adagrad is that by storing the sum of the squares of the past gradients, which are all positive, the denominator grows ever large. Eventually the learning rate will shrink towards zero and the network won't be able to learn any more additional knowledge.

Adaptive Moment Estimation (Adam) [49] store an exponentially decaying average of past squared gradients which reduces the aggressive decreasing learning rate of Adagrad. This is also done in optimization algorithms like Adadelta and RMSprop. But Adam also store an exponentially decaying average of past gradients m_t which is similar to momentum. The way Adam update its parameters is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{2.23}$$

where $\hat{v}$ and $\hat{m}$ is computed by:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \quad \hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.24}$$

m_t and v_t are estimates of the first order moment and the second order moment of the gradients. Upon initialization the authors, Kingma and Ba, discovered that biased towards zero, so to counteract the bias they instead compute $\hat{v}$ and $\hat{m}$ with a factor, β, which the authors propose default values of 0.9 for β_1 and 0.999 for β_2.

2.3.4 ResNet

Residual Network (ResNet) [50] is in a way an upgraded form of previously mentioned Convolutional Neural Networks. This architecture enable the model to have hundreds of layers. The original CNN would succumb to to the "vanishing gradient" problem, meaning that during the backpropegation the weights that minimizes the loss function would multiplied so many times that the gradient becomes smaller and smaller. In this case adding more layers will no longer lead to better performance and in some cases even degrade model performance.

What makes this architecture so efficient for high number of layers is "identity shortcut connections". ResNet stacks up these connections which is initially don't do anything. During training these layers are skipped, and the model uses the activation functions from previous layers. This compresses the model down to only a few layers at the beginning of training, this enables faster learning. When the model trains again all the layers are expanded again and the "residual" parts of the network explore more and more of the feature space of the source image.

ResNet outperforms shallower networks and are easy to implement in TensorFlow, Keras etc. And is therefor very popular in computer vision tasks. In the years after its authors published the model many new and prominent versions of the architectures have emerged.

2.3.5 EfficientNet

Convolutional neural networks (Section 2.3.2) are often developed according to some specification, and later re-purposed and scaled up for other projects. However, training deep neural networks becomes difficult as depth increases [51], [52]. In deep convolutional neural networks the many non-linear transformations in conventional feed-forward network architectures lead to poor propagation of activations and gradients, often called 'vanishing gradient problem'. This performance degradation problem was attributed to the fact that very deep models are hard to optimize, and the optimizer is not able to converge to a correct minimum of the loss function. To overcome this ResNet [53] introduced shortcut connections between convolutional blocks that allow the gradients to flow more easily through the network. This development of residual networks allows for network to be hundreds of layers deep. These networks have reached state of the art performance many times in the past, but they require tedious manual tuning, and multiple weeks of training.

EfficientNets [54] focuses on improving the accuracy of the state of the art models even further, but also on increasing the efficiency of the model by tweaking the scaling. There are three different dimensions which can be scaled in a CNN; depth (d), width (w), and resolution (r). Depth parameter is the amount of layers there are in the network. Width is how wide the network is. One measure of width is how many channels are in the images - usually three, one channel each for red, green and blue, or one for gray-scale images. Resolution is the number of pixels for height and width of the source image.

- Depth scaling; the most common way of scaling a neural network is to add or remove layers in the model. Number of layers in a model usually range from just a couple to a hundred. The reason larger models perform well is that they capture more complex features and generalize better than smaller models. But due to vanishing gradients in very deep neural networks it is a limit for how many layers a model can hold. Even when avoiding the problem of vanishing gradients adding more layers won't always help, and ResNet with 1,000 layers has similar performance as ResNet with 100 layers.

- Width scaling; very deep residual networks has a problem of diminishing feature reuse, and therefore the networks are slow to train [55]. Widening the network is done by increasing filter sizes in the convolutional layer or to add more kernels. But with very shallow and wide networks accuracy saturates quickly.

- Resolution scaling; the third option is change the pixel resolution of the input image. Reducing the spatial resolution of an image will decrease the performance score of a CNN [56]. But the accuracy gain diminished very quickly for medium to high resolution images.

Scaling in computer vision tasks is usually fixed, and set so that a given model performs optimally on a given tasks. Tan *et al.* proposed a novel technique which use compound scaling to uniformly scale network width, depth and resolution to create an optimal network that capture all fine-grained features of an image. In Figure 2.13 is a visualization created by Tan and Le, the authors of the paper, which depicts how compound scaling adjusts the network dimensions. This automatic scaling is achieved by the inclusion of a compound coefficient ϕ that uniformly scales network width, depth,

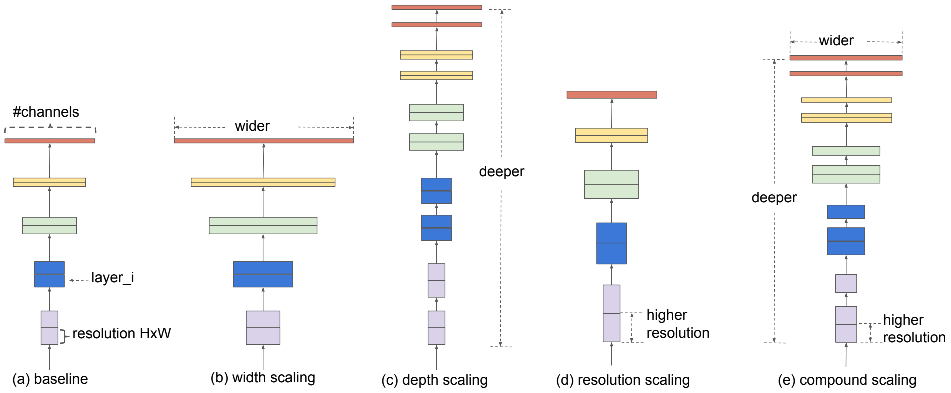


Figure 2.13: Different model scaling methods. (a) baseline network example; (b,c,d) conventional scaling which change one dimension of network width, depth, or resolution. (e) compound scaling method which uniformly scales all three dimensions with a fixed ratio.

and resolution:

$$\begin{aligned} \text{depth: } d &= \alpha^{\phi} \\ \text{width: } w &= \beta^{\phi} \\ \text{resolution: } r &= \gamma^{\phi} \end{aligned} \tag{2.25}$$

Where $\alpha{\cdot}\beta^2{\cdot}\gamma^2 \approx 2$, and $\alpha \geq 1, \beta \geq 1, \gamma \geq 1$. α, β, γ are constants that can be determined by a small grid search and ϕ is a user-specified coefficient that controls how many resources are available for model scaling. The cost of floating point operations (FLOPS) of a regular convolution operation is proportional to d, w^2, r^2, which means that when the network depth is doubled, the FLOPS are doubled as well. When width or resolution are doubled the FLOPS increase by four times. Because of this exponential growth in computational cost of increasing the network dimensions the authors of EfficientNet set a constraint such that for any ϕ, the total FLOPS don't exceed 2^{ϕ}.

The network architecture is not changed when scaling up or down dimensions, so the effectiveness of the model relies heavily on the baseline network. The authors performed a neural architecture search using the AutoML MNAS framework [57], which is used to find well suited architectures for mobile devices, and must therefore be efficient and perform well. The resulting architecture use mobile inverted bottleneck convolution (MBConv), similar to MobileNetV2 [58] and MnasNet [59]. Given the base network the authors fixed the compound coefficient $\phi = 1$ and did a grid search to find the optimal values for α, β, γ. For the baseline network B0, the optimal values found are $\alpha = 1.2$, $\beta = 1.1$, and $\gamma = 1.15$. The next step was to fix the values for α, β, γ and experiment with different values of ϕ which produced EfficientNetsB1-B7.

Compared to other state of the art networks, scaled EfficientNet models consistently reduce parameters and FLOPS by an order of magnitude. See Table 2.6 for a summary

Model	Top-1 Acc	#Params	#FLOPS
EfficientNet-B0	0.773	5.3M	0.39B
EfficientNet-B1	0.792	7.8M	0.70B
EfficientNet-B2	0.803	9.2M	1.0B
EfficientNet-B3	0.817	12M	1.8B
EfficientNet-B4	0.830	19M	4.2B
EfficientNet-B5	0.837	30M	9.9B
EfficientNet-B6	0.842	43M	19B
EfficientNet-B7	0.844	66M	37B

Table 2.6: EfficientNet performance results on the popular benchmark dataset, ImageNet. Top-1 accuracy increase with larger compound coefficient, at the cost of more parameters to update each training step. All models are scaled from baseline model EfficientNetB0 using different compound scaling coefficients.

of ImageNet performance with the respective numbers of parameters and Flops for each of the members of EfficientNet Family, models B0-B7.

2.3.6 Self learning with noisy student

Self-training is one of the most common semi-supervised (Section 2.3.1) methods used. Self-training means to use a trained supervised model, called a teacher, to select a subset of the unlabeled data by filtering out the predictions, also known as pseudo labels, using a threshold. These new pseudo labels are combined with the original labeled dataset and a new model, called a student model, is trained on the combined dataset. This can be repeated multiple times until the system converges, and thereby fully utilizing the wast amounts of unlabeled data to increase model performance.

In recent years, semi-supervised learning have been used in many different domains [60]–[63]. Although producing prominent results, these models have low accuracy and high entropy during early stages of training, and consistency training regularizes the model towards high entropy predictions which prevents it from achieving high accuracy [64].

Xie, Luong, Hovy, *et al.* proposed a novel self-training framework better suited to work well at scale and their model achieved 88.4% top-1 accuracy on ImageNet, which is 2.0% better than the previous state of the art model [64]. They found that that for self-training to work well at scale, the student model should be noised during its training while the teacher model should not be noised while generating pseudo labels. Noisy student improve self-training in two distinct ways: (1) it makes the student larger than, or at least equal to, the teacher so the student can learn from a larger dataset, and (2) it add noise to the student, forcing the student to better generalize on the unlabeled dataset and thereby learn more. The authors used multiple types of noise to improve the student models ability to generalize such as RandAugment data augmentation [65] as input noise, and dropout [66], stochastic depth [67] as model noise. Injecting noise on the input data has the benefit of enforcing local smoothness in the decision function on both labeled and unlabeled images. The student must be able to correctly classify the images with data augmentation transformations and this invariant constraint helps the student model to learn beyond the teacher, and make predictions on more difficult data. When model noise such as stochastic depth and dropout are used, the teacher behaves like an ensemble while it generates pseudo labels, whereas the student are forced to mimic a more powerful ensemble model. Below is the noisy student algorithm in more detail:

1. Learn teacher model θ_*^t which minimizes the cross entropy loss on labeled images
$$\frac{1}{n}\sum_{i=1}^{n}\ell(y_i, f^{noised}(x_i, \theta^t)) \tag{2.26}$$
2. Use an unnoised teacher model to generate soft or hard pseudo labels for unlabeled images
$$\tilde{y}_i = f(\tilde{x}, \theta_*^t), \forall i = 1, ..., m \tag{2.27}$$
3. Learn an equal-or-larger student model θ_*^s which minimizes the cross entropy loss on labeled images and unlabeled images with noise added to the student model
$$\frac{1}{n}\sum_{i=1}^{n}\ell(y_i, f^{noised}(x_i, \theta^s)) + \frac{1}{m}\sum_{i=1}^{m}\ell(\tilde{y}_i, f^{noised}(\tilde{x}_i, \theta^s)) \tag{2.28}$$
4. Iterative training: use the student model as a teacher and go back to step 2.

Where $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ is the labeled images and $\{\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_m\}$ is the unlabeled images.

The noisy student algorithm also works better with data filtering and balancing. The data is filtered by excluding the images the model has low confidence on since they are usually out of domain images. Balancing is used to assure the distribution of unlabeled images match the distribution of the labeled training set. Because the number of samples per class in ImageNet is also uniformly distributed they make sure to balance samples per class for the unlabeled dataset as well. This is done by taking images with the highest confidence for classes with too many samples, and duplicating images for classes with too few images.

To train the teacher and student models Xie, Luong, Hovy, *et al.* use EfficientNet (See Section 2.3.5) as the baseline model and further scale up EfficientNetB7 and obtain EfficientNetL2. EfficientNetL2 is deeper and wider than EfficientNetB7 but uses lower resolution, which enables the model to process more unlabeled images.

2.4 Model evaluation

Data is simply stimuli on a neural network, and a prediction is a reaction of that stimuli. It is difficult to fully understand why we ended up with one specific model after completed training opposed to another, and the reasons for why a particular decision was made. In this section, we will discuss some of the most common metrics used for evaluation, why it is not recommended to train on all of the available data, and some ways to test how good the trained model actually performs.

2.4.1 Dataset splitting

A model can be trained to fit some assortment of data, but to understand the level of assurance the model have on the data you need to test it on data which the model has not seen during learning. Therefore it is common to split the data into three parts:

1. Training dataset; this set should contain most of the samples in the dataset, and is the data used to fit the model. Model sees and learns from this data. Depending on how much data available should be in the range $60-80\%$ of the total number of samples.

2. Validation dataset; used to update the higher level hyperparameters during training. The model does not learn from this data, but is nonetheless incorporated into the final model because the model sees it, and update parameters based upon the evaluation of model fit. To get good mid-training evaluations the dataset should contain in the range of $10-20\%$ of the training data.

3. Test dataset; this dataset is hidden from the model during training and is used to provide an unbiased evaluation of a final model fit. Can be binned to contain carefully sampled data which spans all the features the model should have learned or be randomly selected from the original dataset. The size of the test dataset is usually the same size as the validation dataset $(10-20\%)$.

In Figure 2.14 is an example of a dataset split where 60% is used as training data and each validation and test dataset contain 15% of the samples. How the dataset should be split to achieve optimal results vary on model size and how much samples you have to train on. For the case of a large model it would most likely require a substantial amount of data while models with very few hyperparameters is often more easy to validate and tune, so you can get away with reducing the size of the validation set.

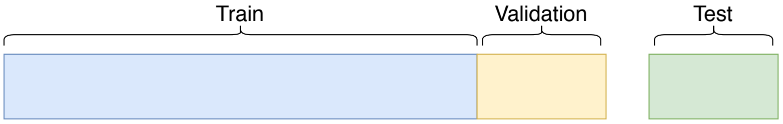


Figure 2.14: A visualization of the dataset splits used for all our experiments except for those performed on Kvasir-Capsule, as discussed further in Section 4.4.1. In this figure, the data is split $60-15-15$, meaning 60% is marked as training data, 15% is marked as validation data etc.

2.4.2 Performance metrics

While training a model is key, how the model generalizes on unseen data is an equally important aspect that should be considered in every machine learning project. It is vital to know whether the model actually works and, consequently, if we can trust its predictions. In the upcoming few sections, we will introduce some of the baseline metrics used to estimate the generalization accuracy of a model on future unseen data.

Confusion matrix

Confusion Matrix (CM) is a table used to describe the performance of a model on a set of test data with known true values. Every row in the confusion matrix table represents the instances in a predicted class while every column represents the instances in an actual class. In a confusion matrix the correct predictions are found along the diagonal line starting in the top left corner and ending in bottom right. Numbers outside this line is incorrect predictions. Confusion matrix is especially useful for calculating true positives (TP), false positive (FP), false negative (FN) and true negative (TN). In Table 2.7 is a layout of the CM for a binary classifier model. In the quadrant with TP, FP, FN and TN is the corresponding number of predicted and true class samples.

		Actual class	
		Class 1	Class 2
Pred class	Class 1	**TP**	**FP**
	Class 2	**FN**	**TN**

Table 2.7: Confusion matrix layout for a binary classifier, where TP = True Positive; FP = False Positive; FN = False Negative and TN = True Negative.

Accuracy

Accuracy (ACC) of a model is the degree of closeness of measurement of a quantity to the quantity's true value. That is, the accuracy is the proportion of correct predictions among the the total number of samples, given as a percentage. The way to calculate the accuracy is

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FN + FP} \tag{2.29}$$

Recall

Recall, also known as sensitivity, is the models ability to find all the relevant cases in the dataset. It is a especially important measure for imbalanced datasets. The definition for recall is the number of true positives divided by the number of true positive plus the number of false negatives

$$\text{recall} = \frac{TP}{TP + FN} \tag{2.30}$$

Precision

Precision (PREC) is the closeness of measurements to each other. While recall expresses the ability to find all relevant instances in the dataset, precision expresses the proportion of the data points the model says were relevant that actually were relevant. It measures the percentage of predicted samples classified as true, which were correctly classified. Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives.

$$\text{precision} = \frac{TP}{TP + FP} \tag{2.31}$$

F_1-score

F_1-score, also known as F-measure, is a combination of Recall and Precision. To create a balanced classification model which is good at both find relevant cases (recall) and relevant cases that are actually relevant (precision), the best practice is to calculate the F_1 score during training and make changes to maximize its value. F_1 score is calculated with the formula:

$$F_1\text{-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.32}$$

ROC Curve

Receiver Operating Characteristics (ROC) is one of the most important methods of checking a binary classification models performance. ROC is a probability curve which tells how well the model is capable of distinguish between two classes. It is plotted with true positive rate (TPR) on the y-axis against false positive rate (FPR) on the x-axis at various threshold settings. The more area under the curve, the better the model perform. TPR and FPR is defined by

$$TPR = Recall = \frac{TP}{TP + FN} \tag{2.33}$$

$$FPR = 1 - Specifity = \frac{FP}{TN + FP} \tag{2.34}$$

When scoring a classifier based on ROC curves it is often accompanied by a AUC value. AUC stand for area under curve, and represents the degree of separability. A perfect classifier has a AUC of 1 which means all classes are correctly classified and it has a perfect degree of separability. A AUC of 0.5 looks like a diagonal line across the ROC plot and means the classifier has no class separation capacity whatsoever. AUC of 0 means the classifier has the worst degree of separability, in fact, so bad that all classes are classified incorrectly.

While ROC Curves are mostly used for measuring performance of binary classifiers it is possible to use for multi-class models as well. For multi-class models we plot N number of ROC curves for N number of classes. Each ROC curve is a representation of the models ability to separate one class from all of the remaining (N-1) classes.

2.4.3 Cross validation

If the dataset used for training is to small to hold off an independent test set to measure the performance of the finished model cross validation is a good option. A popular cross validation method called K-Fold Cross Validation works by dividing the dataset into K splits, and one of these splits become the test data, and another one is the validation data. The rest, K-2, splits are then used as training data. Next, the error estimation is averaged over all k trials to get total effectiveness of the model. This way, every data point get to be in a validation set once, and in the training set K-1 times. By training on all the data the bias introduced to the model is reduced since it learns from all the data and it reduces variance because all the data is also used for validation.

Another popular evaluation method is to use an independent dataset for testing the model. This allows the model to train from all data points in the dataset, but also have a large corpus of data for testing the models fit, given the features of both datasets overlap. In a musical instrument recognition system a trained model achieved over 90% accuracy when tested on a subset of the original data (cross validation) but when using cross-dataset validation the model only reached accuracy in the $20 - 60\%$ range [68].

2.5 TensorFlow Framework

TensorFlow (TF) [69] is an open source library for machine learning created by Google. It is supported by a great community of software engineers and receive incremental updates and addons. We utilize TensorFlow through Python3 [70], which provide a convenient front-end API for building applications and frameworks and a high-performance back-end built in C++. The name TensorFlows arrives from *tensors* and *flowcharts*. This is because it takes input as a multi-dimensional array, known as tensors. These tensors are then used to create flowcharts - structures that describe how data moves through a series of processing nodes. Each node is a mathematical operation like addition or multiplication, and each connection between nodes is a tensor.

TensorFlows architecture works in three parts: (1); preprocessing the data, (2); build the model and (3); train and evaluate the model. This is a big benefit when working on complex data because TensorFlow is a tool which can encompass the workflow for most projects.

In late 2019 TensorFlow 2.0 was released. This was a large update and revamped the framework in many ways. Among the new features was easier model training with the tighter integration of relatively simple Keras [71] API which was integrated into TensorFlow. This introduces a new level of abstraction for machine learning developers.

2.5.1 tf.keras

Keras was introduced as a high level API built into TensorFlow 2.0. In this update, Keras' backend was also switched from running on multiple machine learning frameworks

[8] Application Programming Interface (API) is a set of functions and procedures allowing the creation of application that access features or data of another application.

to only run TensorFlow. It has three key advantages over TensorFlow native: (1); it is user friendly in that it has a consistent interface optimized for common use cases and provides clear feedback for the user. (2); It is modular and composable, so that it is easier to stitch configurable building blocks together. And (2); it is easy to extend with new layers, metrics, loss functions etc.

2.5.2 tf.data

The *tf.data* API enable the user to build complex input pipelines. As mentioned earlier in the section Tensorflow works in three parts, and the *tf.data* API is the first step in building and training a model in TensorFlow, which is the preprocessing of data. In the case of a model trained on images the pipeline API might aggregate data from files in a file system, apply random perturbations to each image, and merge those images into batches used for training. The API allows to build system which handles large amount of data, reads from a variety of different file formats and perform complex transformations.

This is mainly possible because of the *tf.data.Dataset* which is an abstraction that represents a sequence of elements, or in our case images. In such a image pipeline each element might consist of just a image, or a pair of image and its corresponding class label.

2.6 Related work

We have earlier in this chapter presented the reader with a large portion of the most vital related works for this project. Next we will introduce a bit wider range of related research in the domain of medical image recognition and tracking in the GI tract. This research is outside of the scope of our project, but regardless important for the field. We hope future advancements in the field allow for systems to be built which includes many of the subjects we will discuss next. But first we will look at some related research to deep neural networks and semi-supervised learning.

Zhu *et al.* have made a computer-aided lesion[9] detection system which uses a trainable feature extractor, also based on a CNN, and feed the generic features to a Support Vector Machine which enhance the generalization ability [72]. This method greatly outperform the earlier methods based on color and texture features. However we believe that by using neural networks to do the decision making we can further improve this detection system.

Yuan *et al.* have accomplished an average overall recognition accuracy of 98.0% for detecting polyps in VCE images by using a deep feature learning method, named stacked sparse autoencoder with image manifold constraint (SSAEIM). This method is built on a Sparse auto-encoder (SAE), a symmetrical and unsupervised neural network. It is an encoder–decoder architecture where the encoder network encodes pixel intensities as low dimensional attributes, while the decoder step reconstructs the original pixel

[9] a region in an organ or tissue which has suffered damage through injury or disease, such as a wound, ulcer, abscess, or tumor.

intensities from the learned low-dimensional features [73]. Detecting colorectal polyps are important because they are precursors to cancer, which may develop if the polyps are left untreated. Where we hopefully can build on this method is by using a larger dataset with pathology proof of other irregularities.

Jia *et al.* present a new automatic bleeding detection strategy based on a deep convolutional neural network and evaluate their method on an expanded dataset of 10,000 VCE images [74]. Gastrointestinal tract bleeding is the most common abnormality in the tract, but also an important symptom or syndrome of other pathologies such as ulcers, polyps, tumors and Crohn's disease. Their method for detecting bleeding have an increase of around 2 percentage in F_1 score, up to 0.9955. This method and its high score in somewhat limited to bleeding, and not very good at detecting other lesion. Our goal is to develop a method for using deep learning to find more generalized pathologies in the gastrointestinal tract.

Wu and Prasad propose semi-supervised deep learning for hyperspectral image classification, which uses limited labeled data and abundant unlabeled data to train a deep neural network. The paper describe a learning framework that use deep convoluational recurrent neural networks (CRNN) for hyperspectral image classification by treating each hyperspectral pixel as a spectral sequence [60]. They use all available data together with the pseudo labels (cluster labels) to pre-train the CRNN and then fine tune with the labeled data. They test their proposed system on spectral satellite images and their method outperforms state-of-the-art supervised and semi-supervised learning methods for hyperspectral classification.

Yalniz, Jégou, Chen, *et al.* present a study of semi-supervised learning with large convolutional neural networks based on the ResNet50 architecture, and proposed a pipeline based on the student-teacher paradigm. Some key findings of their study were;

1. Training with a teacher/student paradigm produces a better model for fixed complexity problems.
2. Fine tune the model with labeled data only.
3. Large scale unlabeled datasets are key to good performance.
4. Build a balanced distribution of pseudo labels.

Next we will go through some other important aspects which we believe is vital to achieve the goal of efficient screening of large populations.

2.6.1 Object tracking

Object tracking is one of the harder problem to overcome in computer vision and is key to achieving good results in endoscopic video analysis. Tracking algorithms are developed to determine the movement of the object or objects in each video frame. The algorithm has to take into account the dynamic environment such as differences in lightning, occlusions and scaling changes. Also the absence of any prior knowledge to the object and its position further increase the complexity of the problem. Zhang *et al.* proposed an approach for visual tracking in videos that learns to predict the bounding box locations of a target object at every frame in the paper “Deep Reinforcement Learning for Visual Object Tracking in Videos” [76]. While other models depends on the capability of a CNN to learn a good feature representation for the target location in the new frame, which means that the model only tracks properly if the target lies in the spatial vicinity of the previous prediction. This is not always the case for VCE videos, where the lens of the camera can suddenly and unpredictably rotate towards the wall of the intestine. This method integrates convolutional network with recurrent network, and builds up a spatial-temporal representation of the video which means that the model is able to predict the target object’s location over time.

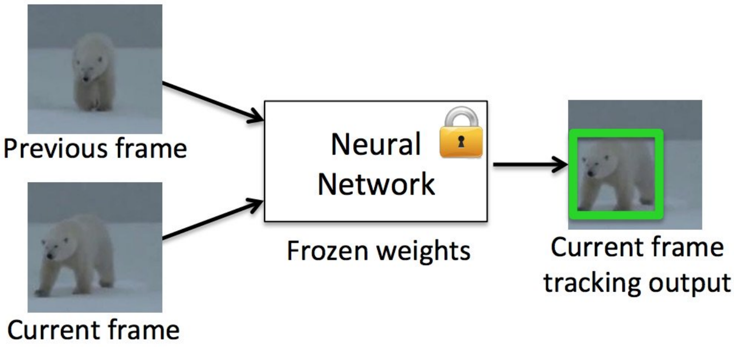


Figure 2.15: Illustration of how object in two frames is tracked with a bounding box[10].

Our hope is that by implementing an object-tracking algorithm we can use it to classify irregularities in the colonoscopy video, and then track that object in the later frames until it disappear out of frame. This will hopefully help with reducing the robustness of the network so that the classifier will not have to check every frame for irregularities.

2.6.2 Segmentation

Image segmentation is the process of partitioning a image into multiple segments of pixel, usually each segment describing some feature of the image or an entire object or class of objects. The goal of segmentation is to simplify the image and make it easier to analyze or further process. Ronneberger *et al.* propose a method in the paper “U-Net: Convolutional Networks for Biomedical Image Segmentation” [77] for using a network and training strategy that relies on the strong use of data augmentation to use the available labeled samples more efficiently. This network outperform the old method of sliding-window-convolution by a great deal. They extend the ”fully convolutional network” [78] such that it works with very few training images and yields more precise

[10] https://www.learnopencv.com/goturn-deep-learning-based-object-tracking/

segmentation. The way this is achieved is to supplement a contracting network by successive layers, where instead of using pooling operators, upsampling operators are used. This means that these successive layers increase the resolution of the output. The high resolution features from the contracting path are combined with the up-sampled output to localize objects and with that a convolution layer can then learn to produce more precise output based on this information.

Another important feature in this architecture is that in the upsampling portion of the network there is also large number of feature channels. These channels allow the network to pass on context information to the higher resolution layers.

A common problem in training neural networks are too little labeled training data. This is also the case for us. We require a lot of medical data, and personnel with the expertise to correctly label our data are of high demand and they usually have very little time for projects like these. This is why Ronneberger *et al.* use different methods of data augmentation to generate more training data. They apply elastic deformations to the available images, and this allows the network to learn invariance to such deformations without the need to see these transformations in the annotated image corpus. Which is particular important in biomedical segmentation since deformation used to be the most common variation in tissue and realistic deformations can be simulated efficiently [77]. By doing this Ronneberger *et al.* were able to achieve very good results (Table 2.8).

Name	PhC-U373	DIC-HeLa
IMCB-SG (2014)	0.2669	0.2935
KTH-SE (2014)	0.7953	0.4607
HOUS-US (2014)	0.5323	-
second-best 2015	0.83	0.46
u-net (2015)	**0.9203**	**0.7756**

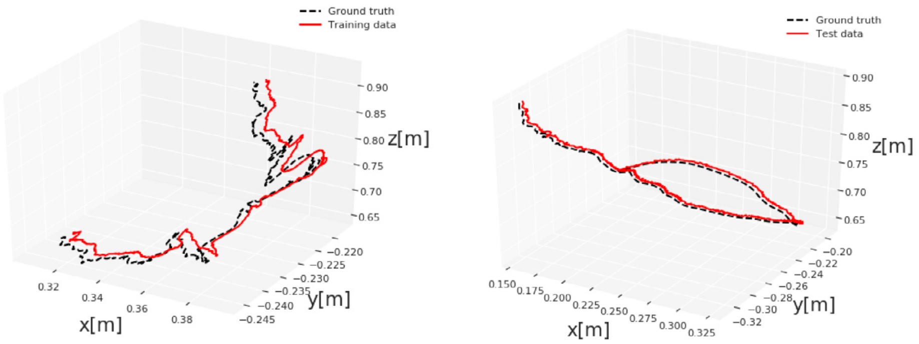
Table 2.8: Segmentation results on the ISBI cell tracking challenge in 2015.

2.6.3 Mapping

As mentioned in Section 2.1.4, a concern when processing the images taken with a VCE is not having the spatial data you get when using a normal fiber-optic endoscope. This is why Turan *et al.* has recently made substantial progress in converting passive capsule endoscopes to active capsule robots, enabling more accurate, precise, and intuitive detection of the location and size of the diseased areas by developing reliable real time pose estimation functionality of the capsule with Deep Recurrent Convolutional Neural Networks (RCNN) [79]. See Figure 2.16 for an example.

This architecture uses inception modules for feature extraction and a RNN for sequential modelling of motion dynamics to regress the robot's orientation and position in real time. By taking multiple of RGB Depth images with time stamps it can calculate the 6-DoF pose of the capsule without the need of any extra sensors. For obtaining the depth images Turan *et al.* use the shape from shading (SfS) technique of Ping-Sing and

Shah [80]. This model outperforms state-of-the-art models like LSD SLAM and ORB SLAM.



(a) Training data vs ground truth.

(b) Test data vs ground truth.

Figure 2.16: An example of Deep EndoVO accuracy [79][11]

2.7 Summary

In this chapter, we have discussed the background and related works of the fields related to our three primary research objectives. This include a introduction to the digestive system and more specific the GI tract, where we we have looked at different types of endoscopy. Regular screening of the GI tract is essential for the discovery of disease, which may be life-threatening. We have seen that most screening examinations of the upper and lower GI tract are performed by traditional endoscopy, which cause discomfort for the patient. Then we looked at the video capsule endoscopy, which are performed by swallowing a wireless video capsule and a video feed is transmitted to a receiver and stored for later diagnose. We then explored some of the currently available dataset taken by both traditional endoscopy and VCE.

After covering the medical background, we began looking into the use of machine learning in the domain of medical image recognition. We started with a brief introduction into the different types of machine learning and introduced the term semi-supervised learning. Next, we went a bit deeper into the backbones of a convolutional neural network, presenting common building blocks and how image data ”flow” through the network in both directions to update the network parameters. We then looked at gradient descent optimization methods before venturing into more complicated architectures such as Google's EfficientNet. We then put that into the context of this thesis by looking at self learning with Noisy Student which was used as inspiration for our work. At the end of the chapter we introduced some background for model evaluation, the TensorFlow ML framework and some other interesting related works.

In conclusion, we have learned that there is a clinical need for a more efficient approach to patient screening, and machine learning has an essential part in achieving

[11] Image credits: paper authors: Turan, Almalioglu, Araujo, *et al.*

this. A dominant obstacle is the lack of annotated VCE datasets, and a lack of frameworks for handling severe class imbalances. Based on this, we researched and developed a semi-supervised teacher-student framework which use self learning by first training sparse labeled images, then generating pseudo labels from a large corpus of unlabeled images, combining these labels with the labeled images, and training a larger model on the combined images. This framework is presented in the next chapter.

Chapter 3

Methodology

We developed a semi-supervised teacher-student based image classification system to take advantage of the wast amounts of unlabeled data, and thereby, reduce the estimated cost of creating medical classification models. As we discussed in Section 3.2.8, medical datasets are often highly skewed, and rarely contain enough labeled data. First, we discuss the process of collecting and labeling data and the challenges that entails. Next we discuss the inner workings of the input pipeline which provides images to the model, and our choice of network architecture and its hyper-parameters. Finally we discuss the various tools and technologies used for the implementation of the teacher-student framework. Here we will mostly argue the reasoning behind our choices of technologies, describe how they effected development of our teacher-student framework, and discuss the premise of which they were chosen.

The motivation behind our teacher-student framework is to reduce the cost of creating a good and reliable classification model in the medical sector. This cost is most often caused by one dominant factor; a large, labeled and balanced dataset is essential to develop a high performing classification system. The doctor-hours needed and the lack of medical personnel to perform the cumbersome and tedious data labeling makes this task very expensive and time consuming. We propose a framework which will make use of unlabeled data to improve model performance and thereby reduce cost.

As with most modern software projects, our teacher-student framework is not written from scratch, but with the aid of various tools, libraries and frameworks. We have focused on only using these aids if they meet our requirements, which are (1) being open-source and easily accessible, and (2) the technology should be mature and widely tested in the field.

3.1 Data collection

As discussed in Section 2.2, there is number of publicly available datasets online, and some which are restricted. Some of these datasets are difficult to access, and there is a need for more publicly available datasets which are diverse and well documented. To assist the field of research within medical computer assisted analysis tools the datasets need to be large and well annotated. Some of the mentioned datasets lack adequately documented, annotated samples from a good source and is not well suited for our research. Thus, as a vital part of our research, we aim to produce a collection of well annotated and adequately big dataset that can be used not only in this study, but also contribute to the research community and have a impact on the research comparability in future. We achieve this by collecting medical data, sorting and annotating it and making the dataset publicly available and free to use.

3.1.1 Privacy, Legal and Ethics Issues

To obtain medical patient data from a hospital in Norway is very difficult and not straight forward. All medical data is considered personal and is therefore strongly protected from unauthorized use and distribution by the Privacy Data Protection Authority. A medical study conducted at two academic hospitals in Southern California from May 2017 to September 2018 found that most patients are willing to share their data and bio-specimens for research purposes [81]. Regardless of the patient opting in to share their data and bio-specimens, it is still difficult for researchers to access it due to strict General Data Protection Regulation (GDPR) laws [82]. The GDPR requires that patients provide consent for clinical data use for research.

We solved this problem by collaborating with a number of Norwegian hospitals and a research company, mainly Vestre Viken Hospital Trust and Augere Medical AS. Through Augere Medical, we got in contact with Vestre Viken Hospital Trust (a hospital in Norway), allowing us to download anonymous data from hospital systems and transfer it using a secure media to our facility. Upon downloading the data, we further stripped the metadata files for potential information regarding patients like time stamps and dates.

The study was approved by the Privacy Data Protection Authority. It was exempted from approval from the Regional Committee for Medical and Health Research Ethics - South East Norway. Since the data is anonymous, the dataset is publicly shareable based on Norwegian and GDPR laws.

3.1.2 Kvasir-Capsule

The VCE videos were collected prospectively from consecutive clinical examinations performed at the Department of Medicine at Bærum Hospital, Vestre Viken Hospital Trust in Norway between February 2016 and January 2018 with an Olympus VCE system. Initially we received 44 VCE videos, which were first analyzed by a trained clinician, whom selected thumbnails of region of interests of both lesions and normal findings as part of their clinical work. Later, we received an additional 74 videos which were used for unlabeled data.

Originally, the videos were captured at a variable frame rate of 3-5 frames per second (FPS), in a resolution of 336×336, and encoded using H264 (MPEG-4 AVC, part 10). The videos are exported in AVI format using the Olympus system's export tool packaged and encapsulated in the same H264 format, i.e., the frame formats are the same, but the frame rate specification is changed to 30 FPS by the export tool.

Prior to being exported the videos were anonymized by removing all metadata and renaming the files with randomly generated filenames. A few videos had to be shortened to cut out images taken during the examination prior to entering the mouth of the patient. After that the videos are uploaded to Augere Medical AS[1] tagging tool. Three MSc students went through all the frames of the videos in collaboration with an expert endoscopist and labeled and marked findings with bounding boxes. When the students encountered images they were uncertain of, the expert reviewed the case.

When all 44 videos had been labeled, the dataset was exported from Augere Medical tagging tool and split into folders for each class. The number of images per class are given in Figure 3.2, and in Figure 3.1 are example images taken from each class of the labeled dataset.

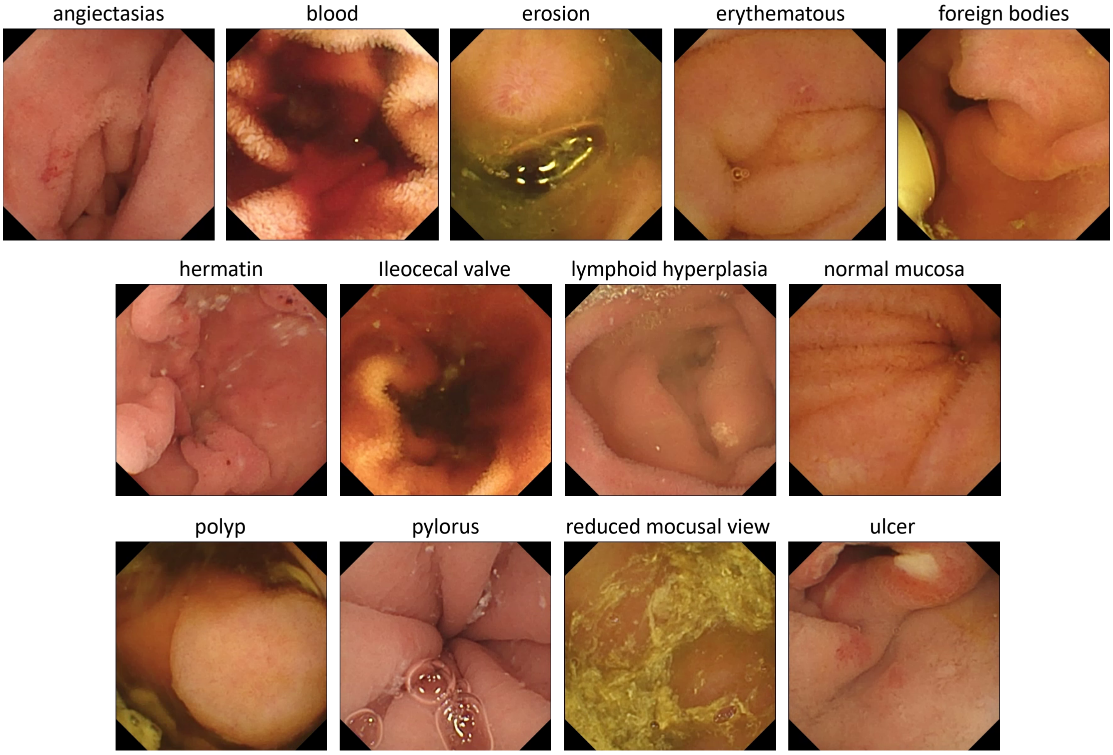


Figure 3.1: Image examples of the 13 various labeled classes for images and their corresponding class names.

Additionally, we provide a comma separated value (CSV) file named *image-labels.csv* that gives the mapping between an image (filename), the label for each image, the corresponding video it originate from, and the video frame number. The Kvasir-Capsule [5] dataset will be an open-source dataset available for others scientists, and will later be grown to include more VCE videos, and more labeled and unlabeled samples. Kvasir-Capsule dataset is available from the Open Science Framework (OSF) accessible

[1] `https://augere.md/`

via the link `https://osf.io/dv2ag/`.

Labeled images

In total, the dataset contains 44,260 labeled images stored using the PNG format, where Figure 3.2 shows the 13 different classes representing the labeled images and the number of images in each class. We defined three main categories of findings, namely anatomical landmarks, quality of mucosal view, and pathological findings. The classes are structured according to the category of finding.

Each class and the images belonging to it is stored in the corresponding folder of the category it belongs to. As observed in Figure 3.2, the number of images per class is not balanced. This is a global challenge in the medical field because some findings occur more often than others, and adds a challenge for researchers since methods applied to the data should also be able to learn from a small amount of training data.

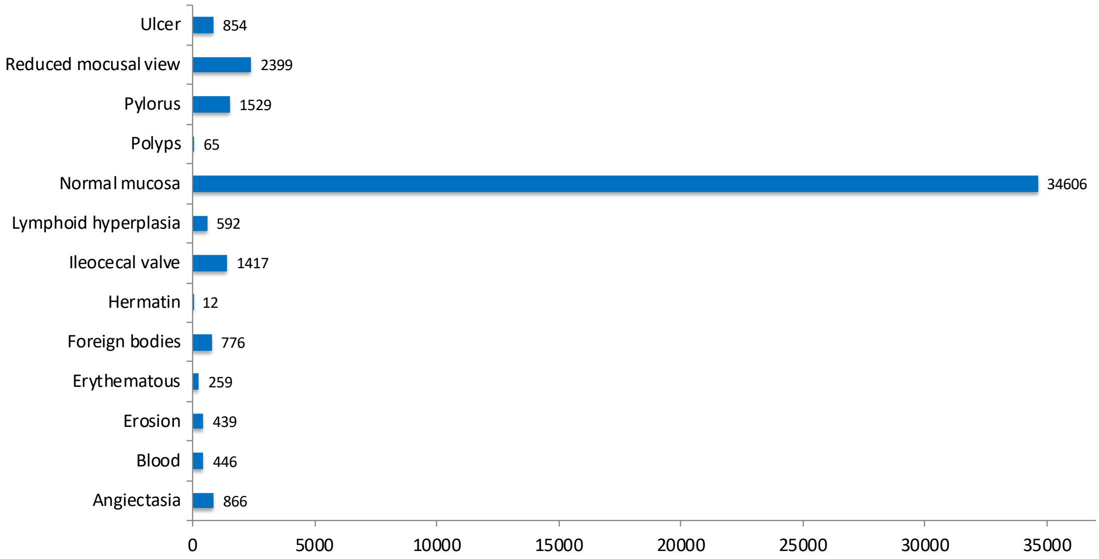


Figure 3.2: The distribution of labeled samples per class in the Kvasir-Capsule dataset.

Anatomical landmarks

Anatomical landmarks are characteristics of the GI tract used for orientation during endoscopic procedures. Furthermore, they are used to confirm the complete extent of the examination. We have labeled two classes of *anatomical landmarks* which delineates the proximal and distal end of the small bowel. The *pylorus* is the anatomical junction between the stomach and duodenal bulb and is a sphincter (circular muscle) regulating the emptying of the stomach into the duodenum. The *ileocecal valve* marks the transition from the small bowel to the large bowel and is a valve preventing reflux of colonic contents, stool, back into the small bowel.

Quality of mucosal view

A complete visualization of the mucosa is crucial to ensure one discovers all pathological findings. For the *quality of mucosal view* assessment, we have labeled two classes from normal images without any other findings. *Normal mucosa* depicts relatively clean small bowel mucosa with healthy villi and no pathological findings. This class can also double as a ”normal” class versus the pathological findings classes (see below). The class *reduced mucosal view* shows small bowel content reducing the view of the mucosa, like stool or bubbles, meaning the mucosa can not be adequately assessed.

Pathological findings

All parts of the GI tract can be affected by abnormalities or findings due to disease, and the small bowel is no exception. Abnormalities, called *pathological findings*, in the small bowel can be seen both as irregular content in the mucosal lumen or as changes to the mucosal surface. These findings are classified according to the Minimal Standard Terminology, defined by the World Endoscopy Organization [83].

Normally, the small bowel contains only a certain amount of yellow or brown liquid in some cases the color of the liquid change and become red because of bleeding from abnormalities either in the upper GI tract or the small bowel and cause the appearance of fresh *blood* in the lumen. In cases with minimal bleeding, one may observe small black stripes called *hematin* on the mucosal surface. *Foreign bodies* like tablet residue or retained capsules can also be observed in the lumen. Typical mucosal changes, sometimes cover lager segments, such as a reddish appearance called *erythematous mucosa.* The mucosal wall can also have different focal lesions, which can be flat, excavated or protruding compared to the surface of the normal mucosa. The flat lesions represented in the Kvasir-Capsule dataset are *angiectasias*; small superficial dilated vessels causing chronic bleeding and subsequently anaemia. It mostly occurs in people with chronic heart and lung diseases [84]. Excavated lesions erode to different extents the surface of the mucosa. Most common are *erosions*, covered by a tiny fibrin layer, while larger erosions are called *ulcers.* As an example, Crohn’s disease is a chronic inflammation of the small bowel characterized by ulcers and erosions of the mucosa. It may cause strictures of the lumen, making the absorption and passage of nutrients difficult [85]. The classes of protruding lesions in this dataset are *polyps*, that may be precancerous lesions, and *lymphoid hyperplasia*, which represents normal lymphoid tissue in the mucosal wall.

Unlabeled images

Later, we received an additional 74 videos from Vestre Viken Hospital Trust, taken from examinations around the same time as the initial 44 videos. These videos were not annotated but used for generating the unlabeled images for our project. In total there are 2.6 million unlabeled images exported from the 74 unlabeled videos. The original videos are included in the dataset. We hope this can be used for other semi-supervised research projects, where the temporal information of the data is also included in the experiment, or to generate more labeled data with the help of medical experts.

Videos

In addition to the labeled images, all 118 videos used for extracting frames are included in the dataset in MP4 format. These videos were processed exactly the same as the videos used for generating the annotated image dataset. The videos use the same codex, and same frame-rate and resolution. This is to make sure the labeled and unlabeled dataset are compatible.

3.2 Data pipeline

The input pipeline is implemented in TensorFlows [69] *data.Dataset* library. This was chosen over *datagenerator* due to the easy of use, but later became an issue due to the complexity and some diffuse runtime errors. The main benefits by using data.Dataset is that all data is handled in tensors and computation is automatically distributed to the Graphical Processor Unit (GPU) which enhance the load distribution of processing power. In Figure 3.3 is a course look at how our network is trained and the scripts used to do so. The input pipeline sits between the training and test dataset and the TensorFlow model.

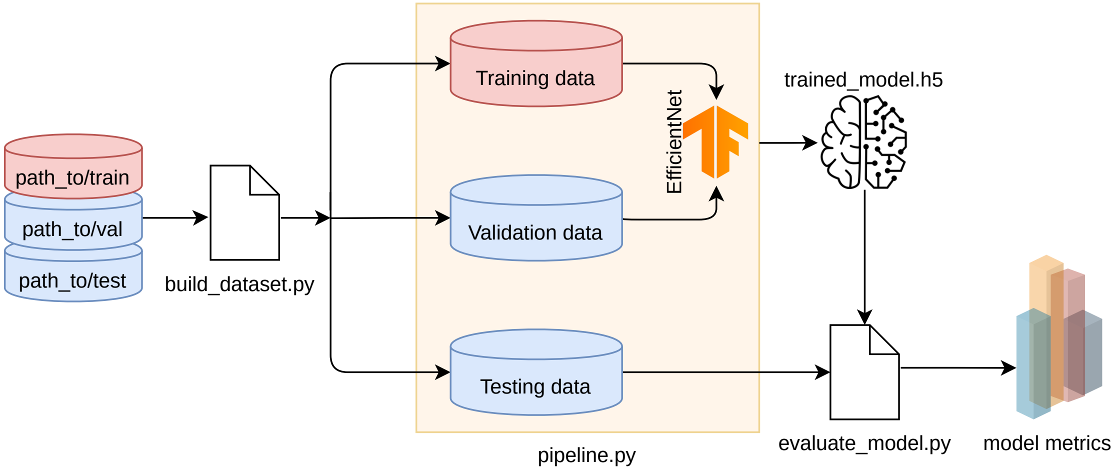


Figure 3.3: The pipeline used for training our models. The scripts in this figure are available on GitHub[2].

All the code for handling the data pipeline is managed in a separate script called pipeline.py. The input pipeline outline can be seen in Figure 3.4. In this figure, we visualize the different paths of data used for training, and data used for testing and validation. The main difference in the two paths is that training data is resampled to create a uniform distribution of samples, and augmented to create small variations in the images to help the model generalize better.

[2] `https://github.com/henriklg/master-thesis`

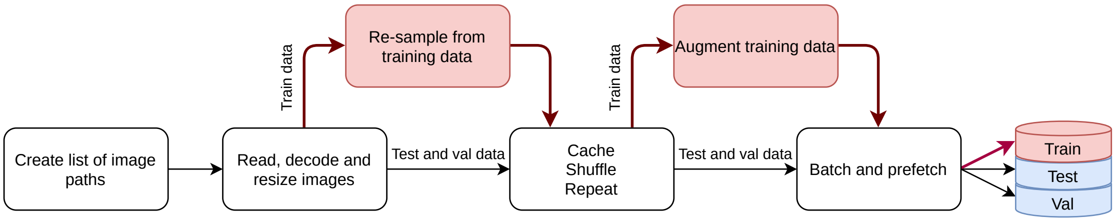


Figure 3.4: The input pipeline we use for providing our network with image-label pairs during training. The red connections represent the path of the training data.

3.2.1 Splitting and resize images

Although not strictly a part of the input pipeline itself, preprocessing is a vital step in machine learning as the quality of the data and the useful information that can be derived from it directly affects the ability of our model to learn.

Initially the dataset was split into three; training data, test data, and validation data. This was done by using tf.data.Dataset core operations *take* and *skip*. The take function, when called upon, returns a sub-dataset with the same number of samples as the number it receives as a argument. Skip function returns a sub-dataset where it skips the number of elements as stated in input argument and return the remaining samples.

```
>>> dataset = tf.data.Dataset.range(10)
>>> dataset = dataset.skip(7)
>>> list(dataset.as_numpy_iterator())
[7, 8, 9]
```

Listing 3.1: Demonstration of *Dataset.skip*, used to split a dataset object into subsets.

However, because of the imbalanced dataset we have been working with, this turned out to drop minority classes in some runs. This happened because the take and skip methods picks samples from the entire dataset as whole, and not per class. In Hyper-Kvasir, the class with smallest number of samples, called a minority class, is hemorrhoids with 6 samples. Depending on the random shuffling for each run these samples would often all end up in one of the dataset splits and not be represented in the other two. To mitigate this issue, we created a separate script for pre-splitting dataset into sub folders for each, training, testing and validation dataset. The outline of this script is given below.

```
for every class name in directories:
  sort the images
  shuffle the filenames
  split the class into train, test and val
  for each split_ds in datasets:
    for filename in sub-split:
      resize and save the image
```

Listing 3.2: Pseudo code for critical parts of *build_dataset.py*. Used for resizing and splitting a dataset.

We split the data into 60% training data, and leave 15% for test data and validation data respectively. To make the split reproducible we first sort the data in alphabetically order after filenames then use seeded random to shuffle the filenames so the three datasets contains a random assortment of images from the original dataset. In this step, we also reduce the image dimensions to 256 × 256 pixels to make the data easier to use during the next preprocessing steps. For downscaling the images, we use a resize function from the highly optimized library OpenCV [86], with a bilinear interpolation. We used OpenCV because for this particular task it was about 30% faster than Pillow [87], another popular Python imaging library. However, this downscaling might introduce aliasing and artifacts to the images as the Hyper-Kvasir's median image dimensions are 768 × 576 pixels, which means we are reducing the dimensions with a factor of three. Area based interpolation [88] or Gaussian resampling, with a suitable chosen radius, may give better results. This is something I would like to have tested if we had more time.

Another benefit we get from running all images through this processing script is the reduction in dataset file size. In Hyper-Kvasir this size reduction correspond to a magnitude in total file size reduction, from 28 gigabytes to 2.8 gigabytes. This helps to efficiently load the images into the pipeline during training.

At this step it would be natural to also apply normalization to the images, but TensorFlow handles this gracefully while reading the images from disk, so we have opted to leave this out of the script. To normalize an image in the context of machine learning means to squeeze the pixels values into a range from zero to one. This is done because usually when reading an image from disk it is represented with an integer value between 0 and 255. Although this integer value can be directly represented to the neural network models, this can result in challenges during training like slower learning. Finally the images are saved to a corresponding directory for each train, test and validation data in the given output directory.

Because Kvasir-Capsule dataset contain frames taken from video we had to take this into account when splitting the dataset. What will happen if special care is not taken is we will see models which perform better than expected when tested on the test dataset, but might be under performing when used in production. This happens because nearly identical images from the same video and finding will be split among both the training dataset and the testing dataset, and the model therefore 'sees' the test data. This is why we split Kvasir-Capsule in a way no same finding are in both splits.

3.2.2 Loading images into the pipeline

To efficiently read and process the images in the pipeline we use TensorFlow's function *list_files* from the tensorflow.data API. This function creates a Python iterable dataset object of all files matching a glob pattern. An example of a glob pattern could be "/source/datasets/*.jpg". In this example, the "*.jpg" is a wildcard followed by a image format extension, which means that a dataset will be created out of all JPEG images inside the "datasets" directory. This function then returns a dataset of strings corresponding to filenames. Although not strictly necessary because the images were randomly shuffled before being split into train, test and validation datasets, the filenames are shuffled again upon entering the dataset. This is repeated three times for each of the train, test and validation datasets. If we get 5 samples from this dataset of strings this is how they would be represented:

```
>>> for path in dataset.take(5):
>>>     print (path)
tf.Tensor(b'/normal-cecum/cc6ed77fbc04.jpg', shape=(), dtype=string)
tf.Tensor(b'/polyps/119100adf1de.jpg', shape=(), dtype=string)
tf.Tensor(b'/esophagitis/f3be6279f5f7.jpg', shape=(), dtype=string)
tf.Tensor(b'/dyed-lifted-polyps/dfb00c142d.jpg', shape=(), dtype=string)
tf.Tensor(b'/dyed-lifted-polyps/aa0268867b.jpg', shape=(), dtype=string)
```

Listing 3.3: Tensor representation of image filepaths to the original location of dataset samples.

Next we apply a transformation function to each element of the dataset to create image-label pairs from the list of filepaths. This transformation function does three things; one-hot encodes the label based on the parent directory; decodes the image so the image is represented with a tensor with dimensions (image width, image height, color channels); resize the image to the correct dimensions. Once we have a dataset object we can transform it into a new dataset by chaining method calls on the dataset object.

3.2.3 Optimize performance

To build an efficient pipeline requires that the GPU is provided data at the correct time. If the GPU is waiting to receive data, the pipeline is not optimized and training a model will take longer. Preferably the pipeline delivers data for the next step before the current step has finished.

The main steps involved in training a model is (1); opening a image file from storage, (2); reading the data from that file and (3); sending the data through the model and update the weights. If these operations are performed in synchronous and sequential order the model can not train while it is waiting idle for an image to be opened and read. Therefore the training step time is the sum of all three steps, see Figure 3.5.

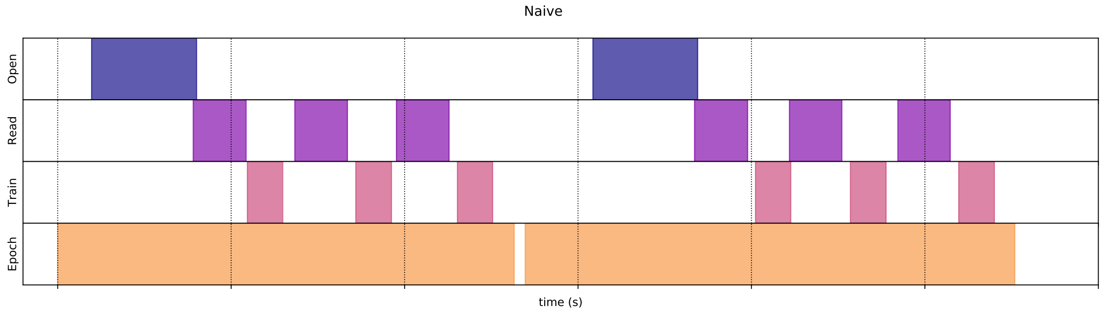


Figure 3.5: The time spent for reading a file from storage, opening the contents of the file, and running the image through the network - the naive method, in which every step is performed in sequential order.

The tf.data API has a couple of approaches that aim to make this as fast as possible. The first one is prefetching. Prefetching overlaps the preprocessing and model execution of a training step. While the model is executing training step s, the input pipeline is reading the data for step $s + 1$. Doing so reduces the training step time by a significant amount as seen on Figure 3.6. Because we are using a tf.data.Dataset object for holding all our images it is very easy to introduce prefetching to the pipeline. The tf.data API provides the tf.data.Dataset.prefetch transformation which decouple the time when data is produced by the central processing unit (CPU) from the time when data is consumed by the GPU.

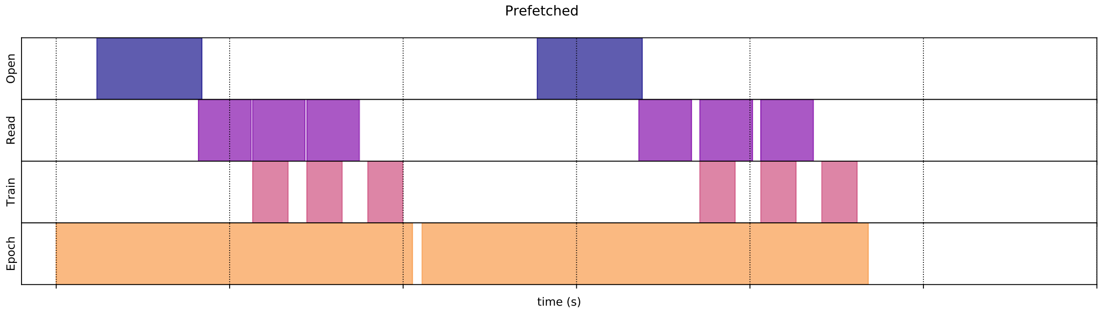


Figure 3.6: The time spent for reading a file from storage, opening the contents of the file, and running the image through the network - with image prefetching. With prefetching, the CPU can prepare images to the network while the GPU is updating the weights for the previous image.

We rely heavily on data augmentation to help the model to generalize and not overfit. This is done when preparing the data by using tf.data.Dataset.map transformations, which applies a user-defined function to each element of the input dataset. Because the samples are independent of one another, the process can be run in parallel across multiple CPU cores. The tf.data API provides a num_parallel_calls argument to specify the level of parallelism, this argument can be set manually or automatically. We have chosen to go with the automatic delegation, which sets the level of parallelism on runtime. See Figure 3.7 for the naive approach, and Figure 3.8 for overhead when using parallel mapping.

The last step, we take to optimize the training efficiency is to cache the dataset to local storage (an NVMe Solid State Drive in our case). This will save some operations, like opening and reading data, from being executed each epoch during training of the model. When the dataset is cached, the images will be opened, read and in our case some pre-processing are performed, the first epoch during training, and the following epochs will reuse the data stored in the local cache. See Figure 3.9 for an example of how this affects time consumption. We split the pre-processing steps into two steps. The first step is applied before caching the dataset and the last step is performed after. This is done because some pre-processing steps are to be performed on every element of the dataset in a deliberate way, and some steps are randomized every time the process is applied.

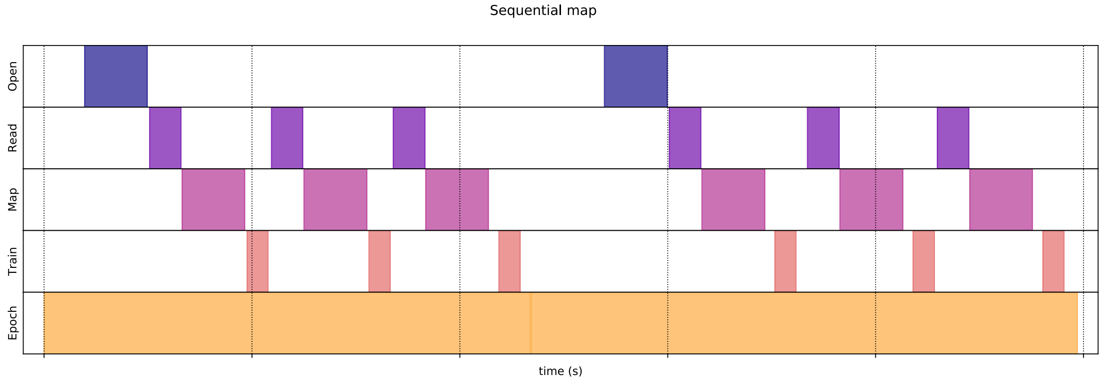


Figure 3.7: Naive pipeline, here the times spent on opening, reading, pre-processing and update network weights, steps sum together for a single iteration.

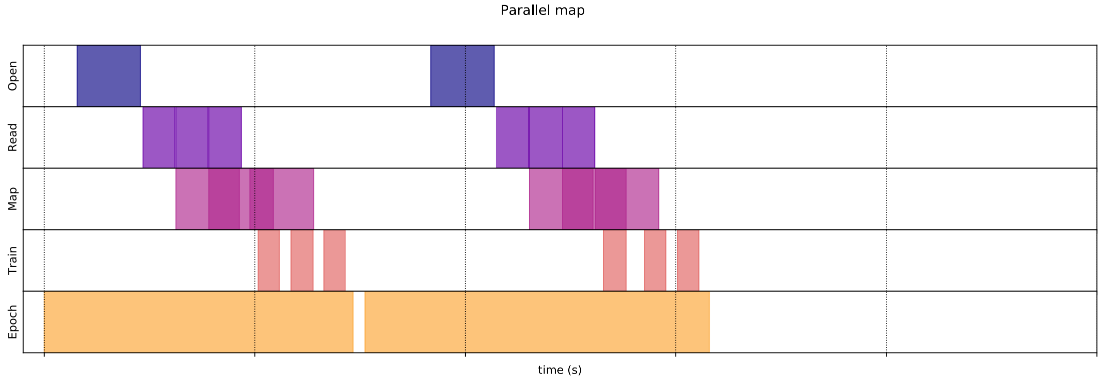


Figure 3.8: Here you can see pre-processing steps overlap, and the overall time for each iteration is reduced due to CPU parallelism.

The mapping that are applied before the caching is: read the label and one-hot encode to integer, read and decode the image, normalize image and resize image dimensions. The mappings that are performed after caching is: shuffling, batching, augmenting.

3.2.4 Shuffle the dataset

We shuffle the data to reduce variance and to make sure the model remains general and less overfit. This is especially important as our dataset is sorted by class. In our pipeline the data is shuffled twice for redundancy. We can afford this because the tf.data API has a low computational performance hit the way it applies the shuffle transformation. The pipeline shuffles the entire list of image filepaths initially, and then a shuffle transformation is applied a second time after the dataset is cached, this ensure that the dataset is shuffled between every epoch during training as well. This is

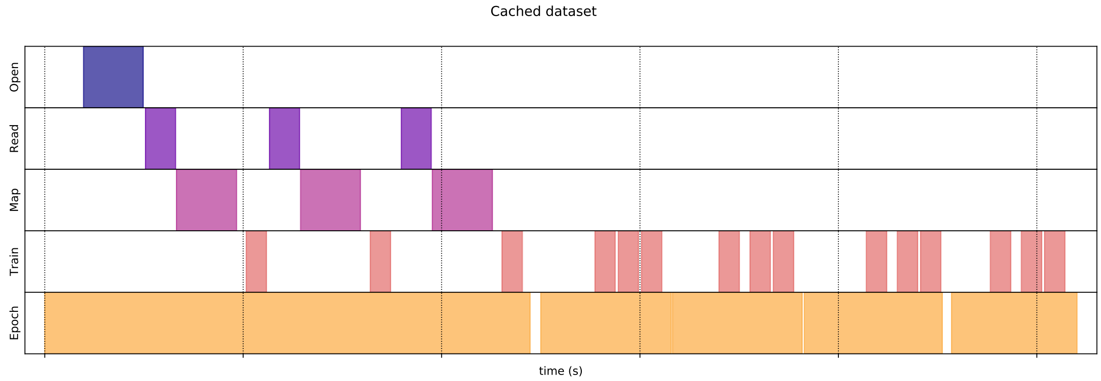


Figure 3.9: Cached pipeline, here the pre-processing is only executed during the first training step, and in the following steps the data is accessed from cache.

important because we have the risk of creating batches that are not representative of the overall dataset, and therefore gradient estimate will be off.

TensorFlow's data API have a dedicated shuffle function which randomly shuffles the elements of the input dataset. It does this by filling a buffer of n elements, then randomly sample elements from this buffer, replacing the selected elements with new elements. For perfect shuffling of the entire dataset we use $n > total\_samples$. The argument *reshuffle_each_iteration* is set to *True* so every epoch get a unique batch of images. Another method of creating unique batches is to first repeat the dataset, then shuffle and create batches.

3.2.5 Repeat

In some situations it is desirable to extend the dataset with duplicates of past samples. One example of this is for oversampling the dataset. We use the *repeat* function from the tf.data API to "infinitely" repeat the dataset samples.

```
>>dataset = tf.data.Dataset.from_tensor_slices([1, 2, 3])
>>dataset = dataset.repeat()
>>list(dataset.as_numpy_iterator())
[1, 2, 3, 1, 2, 3, 1, 2, 3, .....]
```

Listing 3.4: Demonstration of *Dataset.repeat*, which repeats the dataset indefinitely.

This repeat transformation is applied after caching and shuffling, so that we are sure the model sees every sample of the dataset each epoch. We then apply image augmentation at random to each dataset sample so for every finished epoch during training every image is shown once, with a distinctive augmentation filter. Because the dataset is infinitely repeated we must specify how many steps the model should iterate during training, or else it would not know where to stop iterating through the dataset object.

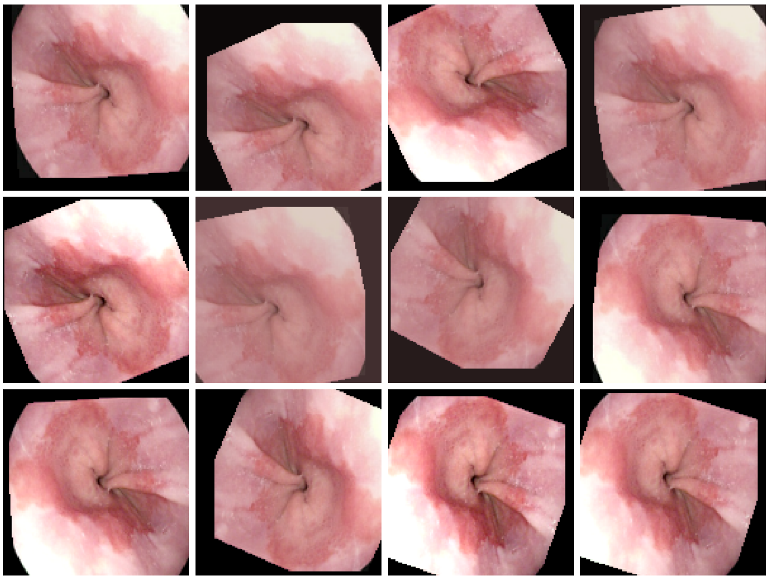


Figure 3.10: The effect of data augmentation on an image from Hyper-Kvasir. Image taken from barrets-short-segment class. In this example, the augmentations performed are a bit excessive to underline the effects.

3.2.6 Data augmentation

The datasets we will be performing our experiments on, like many medical domain datasets, share a common unbalanced data problem. That is images of the target classes, only appear in a very small portion of the the entire dataset. Having a large amount of labeled images is important for the performance of all medical image classification models to effectively learn. Data augmentation overcomes this issue by artificially inflating the training set with label preserving transformations. This helps the model to generalize better and to overfit less - overall a more robust model.

This augmented data is acquired by performing a series of pre-processing transformations to existing data. These transformations can include horizontal and vertical flipping of the image, skewing, cropping, rotating and much more. By doing this, the augmented data is able to simulate a variety of subtly different data points, as opposed to just duplicating the same data over and over. In Figure 3.10 we have taken one Image from the Hyper-Kvasir dataset and created a tf.data.Dataset object which has then been repeated and gone through the same data augmentations that we use in the input pipeline for the training data.

This is especially important in the medical domain of VCE videos due to two reasons; (1), the camera capsule will orient itself randomly as it travels through the small intestine, and the model have to able to detect a polyp regardless of the orientation and (2), we have a very uneven class balance within the dataset and oversample the minority classes requires some sort of data augmentation to reduce overfitting. Data augmentation will however not solve all data problems, but it has been proven to be very effective for training neural networks. By implementing data augmentation in our pipeline we saw that during training the model would overfit far less. The augmentation transformation we use is as follow:

- **Flip**; mirroring the images across its vertical or horizontal axis. It is computationally efficient and easy to implement as it only requires rows or columns of image matrices to be reversed.
- **Rotation**; rotates the image around its center via mapping each pixel (x, y) of an image to (x', y') with the following transformation

  $$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.1}$$

  We find that setting θ to a random number between -30 degrees and $+30$ degrees give good results.
- **Crop**; pad the image by adding black pixels around the image and then randomly crop it back down to the original image dimensions. We have chosen that the image is padded with 20% of its original dimensions. For example a image with the dimensions 128×128 will be padded with 25 pixels.
- **Brightness**; convert RGB image to float representation, adjusts the brightness, and then convert the image back to original data type. We have set a value, $max\_delta = 0.25$, and randomly pick a value from the interval [-max_delta, max_delta] which is the amount to add to the pixel values.
- **Saturation**; converts the image to HSV, adds an offset to the saturation channel and then converts the image back to RGB. The offset is randomly selected from the interval [lower, upper]. In our experiments we have set the interval to [0.6, 1.5].
- **Contrast**; converts images to float representation, adjusts their contrast, and then converts them back to the original data type. Contrast is adjusted independently for each channel of each image. This is done by computing the mean of the image pixels for each channel and then adjusts component x of each pixel to $(x - mean) * contrast\_factor + mean$. Where contrast_factor is randomly picked from the interval [lower, upper] $= [0.6, 1.5]$.

To selectively choose how much the dataset is augmented during training of a model we have implemented a system which dial back the aforementioned parameters by a percentage. This enables us to train the teacher model with very little augmentation and then increase the image augmentations for the student model. In Figure 3.11 is an example of a batch of 12 images which is shown to the network during training.

One possible draw-back from using data augmentation is the network might miss important features during training if those features are cropped or rotated out of view.

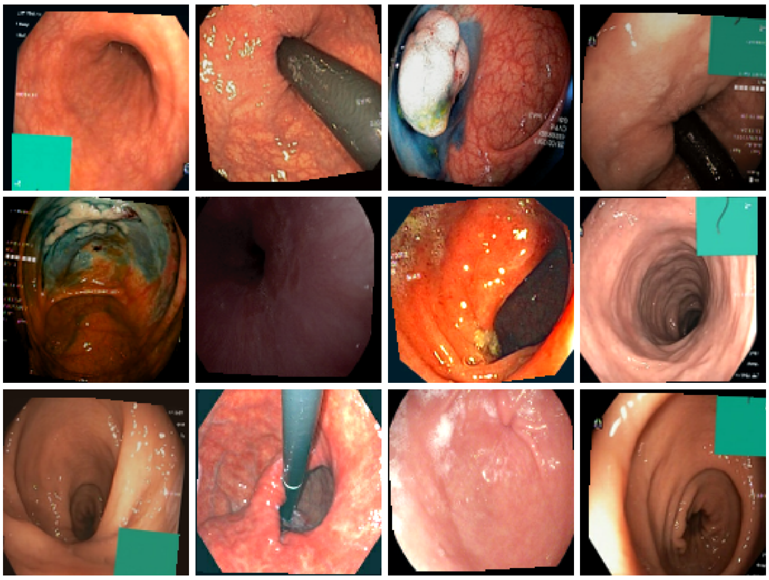


Figure 3.11: The effect of reduced data augmentation on images from Hyper-Kvasir dataset. Here the effect of cropping is reduced by padding the image with 10% of its original dimensions, and rotation is reduced to max 10 degrees.

With this in mind we use cropping and rotation with care. For cropping we dial back the padding to just 10% and rotation is dialed back to only 10 degrees rotation at most.

In recent years, automated augmentation strategies have led to state-of-the-art results in image classification and object detection [65], [89]. These novel automated augmentation policies have shown to improve accuracy, model robustness, and performance on semi-supervised learning for image classification without no additional computational cost at inference time. Due to time constraint we did not test this on our network, but is something we would like to implement in future studies.

tf.image

We have used tf.image API from TensorFlow for all our data augmentations within the input pipeline. This module contains functions for image processing and decoding-encoding operations, and use little computational overhead. All functions that select a random value from an interval have the option to be seeded so that our data is reproducible.

3.2.7 Batching

The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. At the end of each batch, the image predictions are compared to the expected output of the model and an error is calculated.

The update algorithm use that error to improve the model, e.g move down along the error gradient. In the domain of machine learning is is common to name the learning algorithm based on three criterion:

- **Batch Gradient Descent**: all training samples are used to create one batch.
- **Stochastic Gradient Descent**: batch size is one element of the training data.
- **Mini-Batch Gradient Descent**: batch size is more than one sample and less than the size of the entire dataset

Like with the rest of the operations performed in the input pipeline, TensorFlow tf.data API have its own module for batching a dataset. This module have two parameters, one for batch size, and the other for whether the last batch should be dropped in the case it has fewer elements than set by the first parameter. Since our input pipeline repeats the dataset we never run into the issue of having batches which are truncated, so we set this to False.

```
>>>dataset = tf.data.Dataset.range(8)
>>>dataset = dataset.batch(3, drop_remainder=True)
>>>list(dataset.as_numpy_iterator())
[array([0, 1, 2]), array([3, 4, 5])]
```

Listing 3.5: Demonstration of how *Dataset.batch* works. Notice how [6, 7] is missing because drop_remainder is set to True.

One advantage of using mini-batch gradient descent is that it requires less memory. In our case the complete dataset will not fit in memory regardless so batch gradient descent is not possible to test. Another advantage is that typically networks train faster, achieves better training stability and generalization performance with mini-batches [90]. Masters and Luschi found that when experimenting with different batch sizes on ImageNet, CIFAR-10 and CIFAR-100, batch sizes of 32 or less provides more up-to-date gradient calculations, which yields more stable and reliable training. Other scientific papers also support that batch size of 32 is a good choice [91], [92]. We find that the limiting factor for setting the batch size is GPU memory, especially for larger models like EfficientNetsB2-B7. All models are trained on NVIDIA GTX 1080 Ti with 11GB of video memory, which can endure at most 64 images with dimensions of 256 × 256 being trained on EfficientNetB0 which has 5.3 million parameters.

3.2.8 Handling dataset class imbalance

To combat the class imbalance in our dataset, we experimented with two methods for handling skewed classes. The first thing we tested, which was also the easiest to implement with TensorFlow was class weighting. Weighting the classes means that we calculate a score for every class within the dataset, and during training of the network each classes weight is carried when computing the loss. Normally each class has a weight of 1, but we don't want this because of the nature of class imbalance. Instead, if there are twice as many samples in class A than in class B, we give class A a weight score which is half that of class B. The second method was to sample from the training dataset during model training. This ensures the model is fed with images from an uniform distribution of the overall training data. Next we will discuss the two methods and how they were implemented. By running preliminary experiments further discussed in Section 4.2.2, we discovered that sampling the dataset by using uniform distribution of samples from each class gave better results than using class weighting.

Re-sampling

Sampling the training dataset is handled by a resample() module inside *pipeline.py*. This module takes the training dataset as input parameter and then split the dataset into as many separate datasets as there are classes. Because the dataset is stored in a tf.data.Dataset object this must be done according to the tf.data API. We use a transformation method which filters out all image, sample pairs not belonging to the according class. Doing so gives us a python list of dataset objects for each class, containing all original samples for that class. The next step is to cache this list of datasets to reduce computational strain when we later want to iterate over the dataset again. Lastly we repeat every dataset so for each class in the list there is infinite array-like structure of repeating samples. TensorFlow has a convenient function for interleaving elements at random from a list of datasets which we use to go from having a list of repeating datasets to one datasets which produces a uniform distribution of randomly picked image and label pairs when iterated over. The process is over when all classes have the same distribution of $\frac{1}{k}$, where k is the number of classes. This means that the minority classes are over-sampled and the majority classes are under-sampled. When creating this interleaved dataset, TensorFlow use a seeded random number to sample from the datasets to make our results reproducible.

```python
datasets = []
for i in range(num_classes):
    # Get all samples from class i [0 -> num_classes], repeat the dataset
    # indefinitely and store in datasets list
    data = ds.filter(lambda img, lab: lab==i)
    data = data.cache(cache_dir+'{}_ds'.format(i))
    data = data.repeat()
    datasets.append(data)

target_dist = [ 1.0/num_classes ] * num_classes
balanced_ds = tf.data.experimental.sample_from_datasets(
    datasets, target_dist, seed=conf["seed"]
)
```

Listing 3.6: A demonstration of how we sample from the dataset. We first create a list with one dataset per class, repeat all datasets and sample from them according to a target distribution which in our case is $\frac{1}{23}$ for Hyper-Kvasir dataset with its 23 classes.

When we run this on Hyper-Kvasir dataset we get the following output. Here we see that the classes are highly unbalanced. The class with least samples is *hemorrhoids* with a ratio of 0.0005 samples, while the class *bbps-2-3* have a ratio of 0.1077, that is on a order of two magnitudes more samples. The output from after the oversampling shows that all classes have close to the same distribution of $\frac{1}{23} = 0.04348$.

Class	Before	After
barretts-short-segment	0.0049	0.0411
retroflex-stomach	0.0716	0.0450
ulcerative-colitis-0-1	0.0032	0.0459
ulcerative-colitis-grade-3	0.0125	0.0441
esophagitis-b-d	0.0244	0.0444
dyed-resection-margins	0.0928	0.0419
hemorrhoids	0.0005	0.0403
normal-z-line	0.0875	0.0392
esophagitis-a	0.0378	0.0445
ulcerative-colitis-1-2	0.0009	0.0462
barretts	0.0038	0.0409
bbps-2-3	0.1077	0.0463
ileum	0.0008	0.0435
bbps-0-1	0.0606	0.0416
impacted-stool	0.0122	0.0445
cecum	0.0947	0.0430
ulcerative-colitis-grade-2	0.0416	0.0458
ulcerative-colitis-2-3	0.0025	0.0406
pylorus	0.0938	0.0456
retroflex-rectum	0.0366	0.0440
ulcerative-colitis-grade-1	0.0188	0.0424
polyps	0.0965	0.0426
dyed-lifted-polyps	0.0940	0.0465

Table 3.1: Distribution of samples per class before and after dataset re-sampling. After sampling the dataset every class should have the same distribution.

This worked well during our initial experimentation with small datasets. But as we later discovered, there is a problem with the way this sampling method is implemented. tf.data.experimental.sample_from_datasets requires a separate tf.data.Dataset per class, this is solved by using the filter mapping in a loop over each class of the dataset to create one dataset per class, but results in all the data being loaded N times, where N is the number of classes. Our worst case scenario is when loading the Hyper-Kvasir dataset with its 23 classes, which leads to $N_c = N^2 = 529$ classes being loaded into memory at once. For large training datasets this leads to high memory consumption and we were limited to use image resolution of 128 × 128 pixel during training when the training dataset exceed approximately 10,000 images. Testing done in Section 4.2.3 show that lowering the resolution from 256×256 pixels to 128×128 pixels have minimal impact on

```
_, cnt = class_distribution(ds, conf["num_classes"])
total = cnt.sum()
weights = total / (cnt*conf["num_classes"])

# Set weights lower than 1.0 to 1
weights[weights<1.0] = 1.0

class_weights = dict(enumerate(weights))
```

Listing 3.7: A demonstration of how class weights are calculated according to equation 3.2.

```
#  class_weights = {
#        "class 0": 0.5,
#        "class 1": 0.25,
#        etc ...
#}

model.fit(train_ds, epochs=10, batch_size=32, class_weight=class_weight)
```

Listing 3.8: An example class weight dictionary is given to Keras' *fit* function, and a hypothetical class weight dictionary for presenting purposes.

model performance, but further decrease in resolution have increasingly larger penalty. This issue could be solved by using tf.data.experimental.rejection_resample which only loads the dataset once and drop elements from the dataset to achieve balance, however with the structure of our dataset we did not get this function to work.

Class weights

Computing the class weights is handled by 'get_class_weights' method in *create_model.py*. This method return the class weights for each class as a python dictionary, which can be directly into TensorFlow's model training method. The way we calculate the class weights for our data is by the following formula.

$$w_j = \frac{n}{kn_j} \tag{3.2}$$

where w_j is the weight to class j, n is the total number of observations, n_j is the number of observations in class j, and k is the total number of classes. This is implemented in Python in Listing 3.7.

In tf.Keras we then feed the class weights as a dictionary to the model.fit function, like shown in Listing 3.8. This tell Keras that class 0 should hold 50% of the weight for the loss function since it is more important than class 1 which we accordingly set to 25%. In the case of Hyper-Kvasir dataset the computed class weights we use for our experiments are given in Table 3.2.

Notice that the minimum range for class weights are set to 1.0. This is done to increase the weights of the minority classes while holding the larger classes constant.

Class	Weight
0	8.75911
1	1.0
2	13.5036
3	3.48481
4	1.7807
5	1.0
6	81.0217
7	1.0
8	1.14924
9	46.2981
10	11.5745
11	1.0
12	54.0145
13	1.0
14	3.5614
15	1.0
16	1.04544
17	17.0572
18	1.0
19	1.18713
20	2.31491
21	1.0
22	1.0

Table 3.2: Hyper-Kvasir class weights. All weights below 1.0 are set to a minimum of 1 to hold the majority classes constant.

We trained a few models without this minimum class weight, but achieved same results. We found that when weighting the classes the model learns slower in some cases while not learning at all in other cases. The experiments are discussed in more detail in Section 4.2.2 were we present all the results and findings. We could have tested this more but because sampling the dataset was less prone to these errors we use sampling for the rest of our experiments.

3.3 Training and parameter tuning

In this section, we will discuss how we trained the teacher model to be able to classify the training data. We will also discuss some of the limiting factors of training large models. In Table 4.3, is a overview of the system we used to train our models.

3.3.1 Batch size

Due to memory limitations, we are forced to reduce the batch size when training with larger models or higher input image resolutions. If we run with a too large batch size the model will not fit in memory and we get a Out of Memory (OOM) Error. In Table 3.3 is a overview of the maximum allowed batch sizes we are using for which models during training on our system. In this table, we trained the model seen in Figure 3.12 with the EfficientNet for 8 different network dimensions and measured the maximum batch size which would fit in memory as well as the average time spent on a single iteration of the training data (one epoch) for two different image resolutions. Here we see that batch size and training time is proportional to the number of pixels in the input image. When we test different hyper-parameters we use the same model, only switching out the base layer for other variants of EfficientNet.

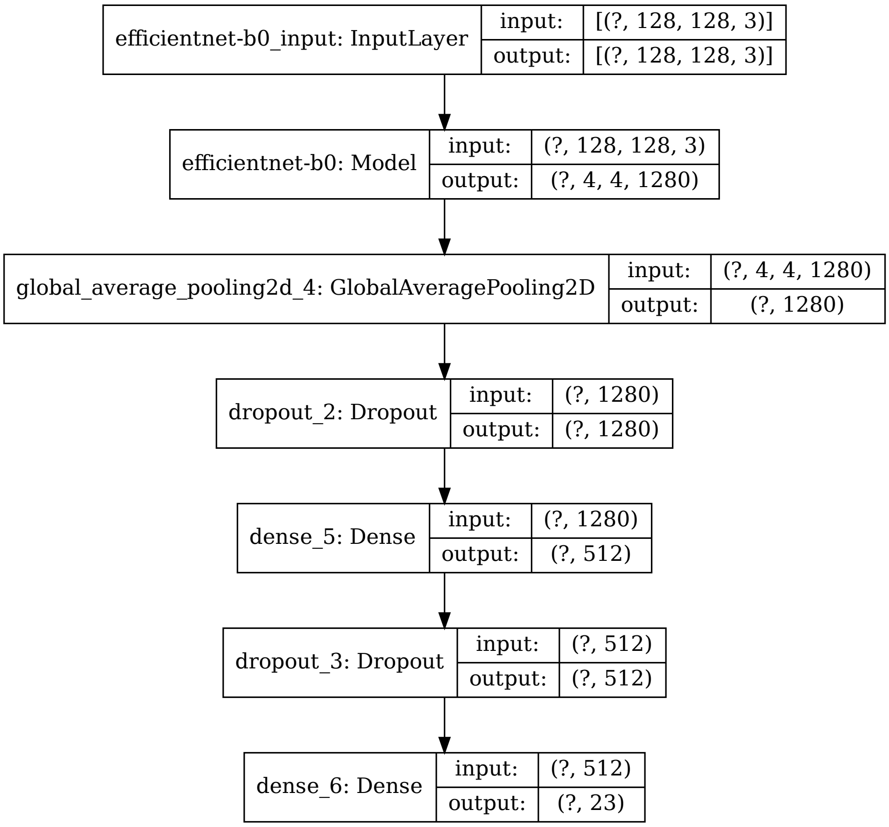


Figure 3.12: EfficientNetB0 with custom top consisting of pooling layer, dropout layers and dense layers.

Based on this table, empirical testing and research papers [90], [93] we chose to use a rather low batch size in the range of 16-32. This was selected as it yielded good training performance and allow us to fit the entire batch in memory on our local system, for even the largest model, if we limit the input image resolution to 128×128. The way we chose input image resolution is further discussed in Section 4.2.3. This was less of an issue in later experiments run on Kvasir-Capsule as we utilized a fully capable AI enabled system called Nvidia DGX-2.

3.3.2 Weight initialization

The weights was initialized with weights pretrained on ImageNet with AutoAugment [89]. The weights we used were downloaded from GitHub[3]. This GitHub repository contain

[3] `https://github.com/Callidior/keras-applications/releases`

	128×128		256×256	
Model	**Batch Size**	**Time**	**Batch Size**	**Time**
EfficientNetB0	256	15	64	61
EfficientNetB1	128	21	32	91
EfficientNetB2	128	22	32	98
EfficientNetB3	128	28	32	113
EfficientNetB4	64	38	16	153
EfficientNetB5	64	52	16	206
EfficientNetB6	32	72	8	288
EfficientNetB7	32	95	8	400

Table 3.3: Max batch size and average epoch time for each EfficientNet model with image dimensions 128 × 128 and 256 × 256 respectively, for our hardware. Time is given in seconds.

pretrained weights for ImageNet with and without the use of AutoAugment, as well as options to download the weights both with and without a top, which is the fully connected layer at the top of the network. We use the weights which does not include the top, as we are using our own fully connected layer because our output layer has 23 nodes instead of ImageNets 1,000 nodes. The reason our output layer have 23 nodes is because there is 23 classes in the Hyper-Kvasir dataset. We further tested with random initialized weights as well as initializing student models with the weights from the teacher model but was not better than using ImageNet weights. See experiments in Section 4.2.2.

3.3.3 Learning rate

Many models train better if learning rate is gradually reduced during training. In the beginning of our experiments, we used SGD with learning rate which followed a inverse time decay which means that the learning rate is decreased by the hyperbolic function

$$\eta = \frac{\eta_0}{1 + dr \cdot \frac{step}{ds}} \tag{3.3}$$

Where η is the learning rate, η_0 is the initial learning rate, dr is decay rate (how quickly the learning rate drops), step is an integer which is incremented for each iteration and ds is decay step, or how often to apply the decay. This yielded great improvements over normal SGD. However, empirical testing showed that Adam gave far greater performance over SGD with learning rate scheduler. We held on to the learning rate scheduler as we switched the optimizer to Adam, which has an built in adaptive learning rate. We find that by using Adam with a inverse time decay schedule, we get a model which has a less fluctuating loss during training. In Figure 3.14 we have trained two models, identical except in Figure 3.14a the learning rate is set to $\eta = 0.001$ during all 30 epochs, while in Figure 3.14b the learning rate starts at $\eta_0 = 0.01$, and then gradually lowers itself for every epoch and reaches 0.001 by epoch 20, and finally 0.0007 at epoch 30, where the models stop training. In Figure 3.13 is a visualization of the learning rate with inverse time decay learning rate scheduler, going from $\eta_0 = 0.01$ and ending at $\eta = 0.0007$ in the 30th epoch with $dr = 0.2$. With regards to the accuracy, both models performed equally well at the end of the thirty epochs. However, the model trained with the use of inverse time decay scheduler learns slower than the model trained without inverse time decay scheduler.

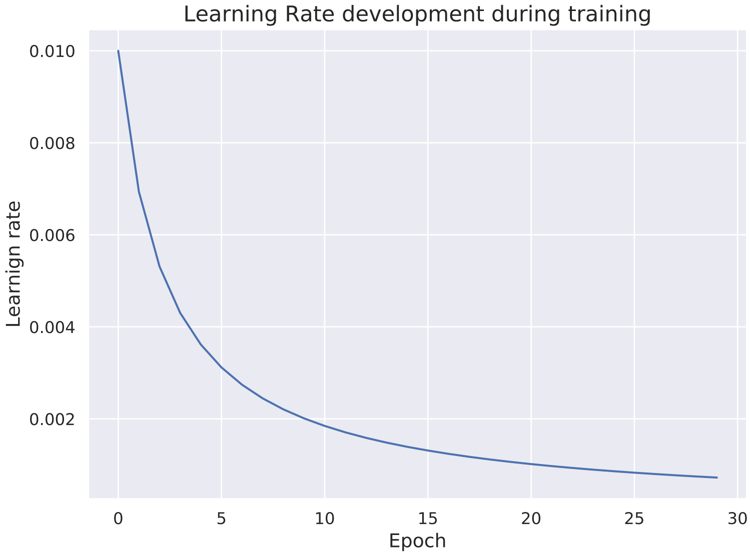
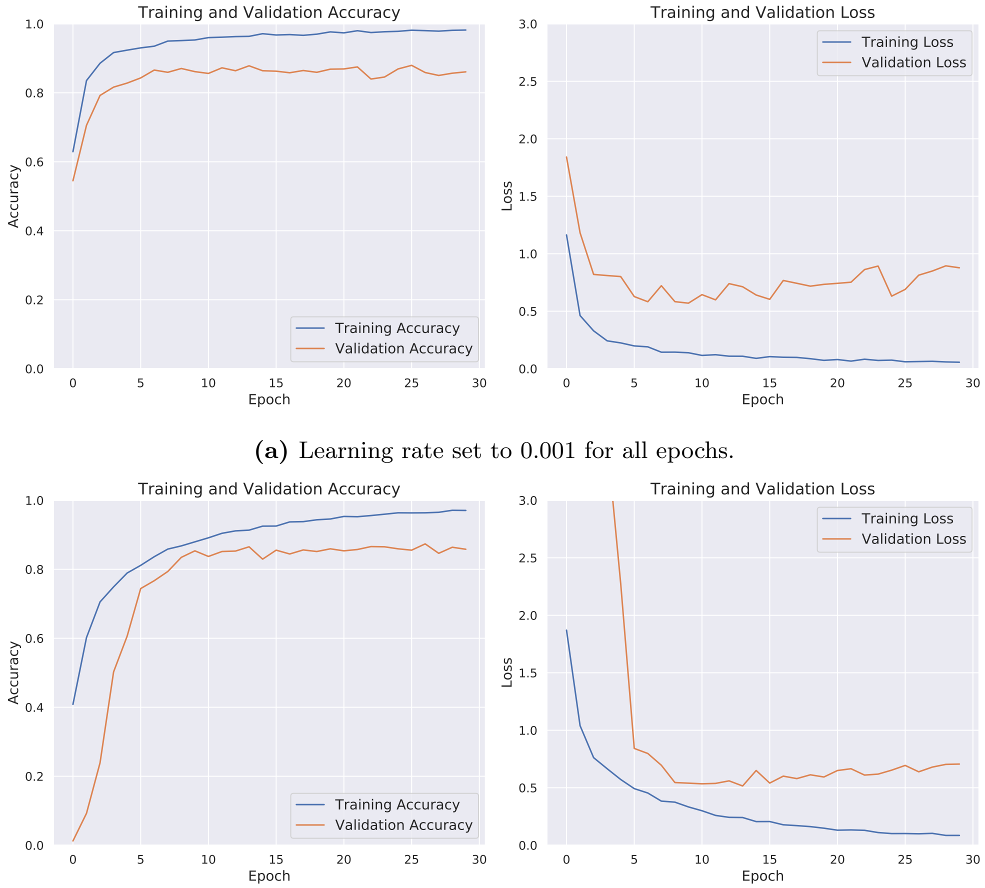


Figure 3.13: Learning rate with inverse time decay. $\eta_0 = 0.01$ and $\eta_{30} = 0.001$. Decay rate is set to 0.25.

When using a learning rate scheduler, the model is slow to converge. With reasonable initial weights the loss will reach its lowest point after around ten epochs, whereas when tested with Adam with its adaptive learning rate the loss is at its lowest after only 2-3 epochs. Because of this the training takes much longer when using the likes of a learning rate scheduler with inverse time decay. Another critical issue we experienced by using Adam with inverse time decay scheduler was that for some experiments the model would freeze during training and cease to update the weights. This would cause us to re-initialize the model and start training from scratch.

For the rest of our experiments we used Adam with learning rate set to 0.001, which is the default value in Keras. We leave $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 1e-07$ as the default as well.



(a) Learning rate set to 0.001 for all epochs.

(b) Learning rate set by inverse time decay learning rate scheduler.

Figure 3.14: Effect of learning rate inverse time decay on the computed accuracy and loss during training. Here both models (EfficientNetB0) are trained using Adam as optimizer, batch size of 128, with data augmentation and sampling, and 20% dropout on the final layers.

3.4 Teacher-student architecture

The next step is to use the trained model, which we from here on out will call the teacher model, to infer pseudo labels from the unlabeled dataset, then train a student model on the combination of the pseudo labels and the original labeled dataset, and finally we switch the teacher with the student and repeat in an iterative process. This process is shown in Figure 3.15. Our system is based upon the noisy-student paper discussed in Section 2.3.6, but with main focus on improving class imbalance in the medical image classification domain.

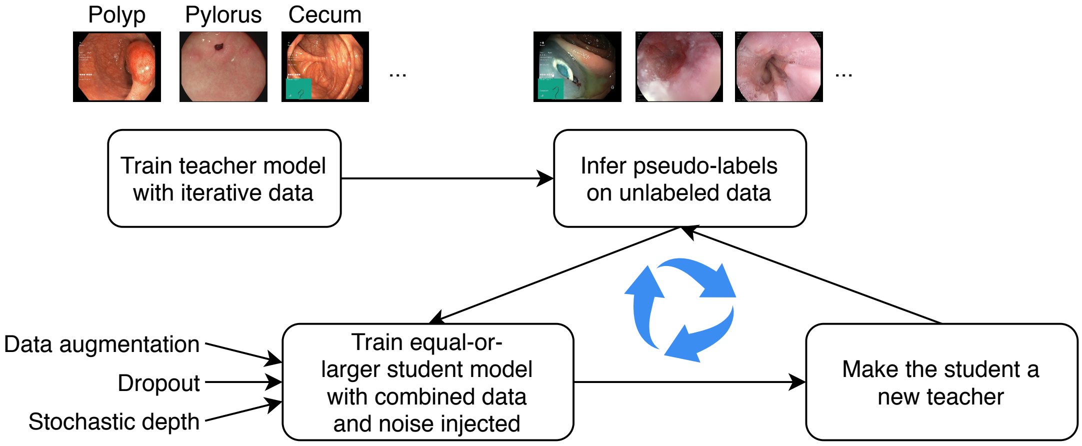

Polyp
Pylorus
Cecum
...
...
Train teacher model with iterative data
Infer pseudo-labels on unlabeled data
Data augmentation
Dropout
Stochastic depth
Train equal-or-larger student model with combined data and noise injected
Make the student a new teacher

Figure 3.15: Illustration of the Noisy student method. Example images taken from Kvasir-V2 dataset [17].

The specific implementation we use is the family of EfficientNets (Section 2.3.5) release v1.1.0 by qubvel[4]. This tf.keras implementation comes with pre-trained weights for both ImageNet and Noisy-Student. We use the weights trained on ImageNet with AutoAugment which are made open source by the authors of EfficientNet on the official github repository[5].

The inputs to our system are both labeled and unlabeled images. We use the open source dataset Hyper-Kvasir (Section 2.2.4), this dataset contains 10,662 labeled images and 99,417 unlabeled images taken from colonoscopy examinations. unlike commonly used datasets for benchmark state of the art models this dataset is more representative of a real world classification problem where there rarely is (1) enough data and (2) same number of samples for each class.

[4] `https://github.com/qubvel/efficientnet`

[5] `https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet`

```
conf = {
    # Dataset settings
    "data_dir": './kvasir-capsule/labeled_splits/',
    "unlab_dir": './kvasir-capsule/unlabeled/',
    "ds_info": 'kvacap',
    "neg_class": None,
    "augment": ["flip","brightness","contrast","rotate"],
    "resample": True,
    "class_weight": False,
    "shuffle_buffer_size": 2000,
    "seed": 2511,
    "outcast": None,
    # Model settings
    "weights": "imagenet",
    "num_epochs": 20,
    "batch_size": 8,
    "img_shape": (256, 256, 3),
    "learning_rate": 0.001,
    "optimizer": 'Adam',
    "final_activation": 'softmax',
    # Callbacks
    "tensorboard": False,
    "decay_rate": 0,
    "checkpoint": False,
    "early_stopp_patience": 5,
    # Misc
    "verbosity": 1,
    "keep_thresh": 0.90,
    "pseudo_thresh": 2000,
    "class_limit": 200000,
    "cache_dir": "./cache",
    }
```

Listing 3.9: An example of the configuration dictionary which stores framework and model specific parameters.

3.4.1 Python implementation of system

In this section, we look at some of the details of our system implementation in the programming language Python. This will hopefully help the reader, if familiar with coding, to understand the inner working of the system, some of the results produced in the next chapter, and some of our limitations. All of the code for this project is available at GitHub[6].

The first step in our implementation is to create a configuration dictionary which holds all system-specific parameters like learning rate, batch size, augmentation settings, growth limit of training dataset, and more. The configuration dictionary, as seen in Listing 3.9, contain four parts: dataset settings, model settings, callback settings and miscellaneous settings. Next we will go through the four categories in more detail.

[6] https://github.com/henriklg/master-thesis

1. In the **dataset settings** are parameters specific to the dataset and how it is handled before reaching the end of the input pipeline. Outcast takes a list of classes to exclude from the training, validation and test dataset objects. Seed is the number which are used for all random number generators in Numpy and TensorFlow. Shuffle buffer size determines how well the data is shuffled. Class weight and resample decides which action to take against class imbalance. Augment takes a list of image adjustments to apply to the training data. Negative class parameter determine if the dataset is binary, and if so, which class to set as negative class - the remaining classes are set as positive. Dataset info is a name for which dataset is used. Data and unlabeled directories point to the location of each folder which contain the dataset images.

2. The **model settings** are data variables which establish the parameters for the model to be used for training. Final activation layer sets which activation function to use for the final dense layer, namely softmax for multiclass and sigmoid for binary. Optimizer determines the algorithm to be used for computing loss. Learning rate is the parameter for how large step size to take in the direction of least cost. Image size is the image resolution, and also the width of the models input layer. Batch size determines how many images to show the network before updating the network weights. Epochs is the number of times to run the dataset through the network before terminating training.

3. **Callbacks settings** contain some settings for creating a callback which are used during training. TensorBoard[7] is Tensorflow's dashboard tool, which allows users to visualize their TensorFlow models, plot quantitative metrics about their execution, and show additional data like images that pass through the models. Decay rate a parameter which sets the decay rate for inverse time decay learning rate scheduler. When set to zero it is disabled. Checkpoint is a setting for saving checkpoints during training. Early stopping monitor the validation loss, and stop the training if it don't decrease within the given number of epochs.

4. **Miscellaneous settings** are settings which don' fit into any of the aforementioned categories. Verbosity parameter sets the level of informational output from the program. Keep threshold is a parameter which determines the threshold for which a image is added as a pseudo label, and if below this threshold it is determined to be out of domain. For most of our experiments this threshold is set to 90%, but is something we would have liked to experimented with other parameters. Pseudo threshold is a parameter for setting the max number of samples per class during iterative training and injection of pseudo labels. Class limit value determines how many images to sample from the unlabeled dataset upon generating the pseudo labels. If set to zero it will use the full dataset. Cache directory parameter set the location to store the dataset object's cache to speed up training.

[7] `https://www.tensorflow.org/tensorboard`

```
teacher = {
    "name": "teacher",
    "model": "EfficientNetB0",
    "aug_mult": 0.1,
    "dropout": 0.1
}

student = {
    "name": "student",
    "model": "EfficientNetB6",
    "aug_mult": 0.8,
    "dropout": 0.3
}

models_list = [teacher, student, teacher, student, teacher, student]
```

Listing 3.10: Additional setting dictionary for each the teacher and student model.

In addition to the aforementioned settings, there are a couple more model specific settings which are defined separate from the rest. These settings determine the difference of the teacher and student model, every other setting is the same for both. See Listing 3.10 for an example of teacher-student settings. The model parameter sets the network architecture to use. Augmentation multiplier set the level of augmentation to apply to the training data. Zero is no augmentation. We set the teacher to have 10% augmentation and the student to have 80% augmentation. Dropout is mostly determined by which EfficientNet network we use, but we also apply some dropout to the last few dense layers in the network. In this example, we apply 10% dropout to the teacher and 30% dropout to the student. Lastly we create a list of how many teacher and student models to iterate over.

The code in Listing 3.11 represents the main execution order of our system. It consist of a for-loop which iterates over the teacher and student models, and for each iteration the settings specific to the teacher or student is copied to the main configuration dictionary. If it is the first iteration in the loop, which means the first teacher model, the program will call on the method which prepare the input pipeline to get ready to read, open, resample, cache, shuffle, repeat, augment, batch, and finally prefetch the training images. For the rest of the iterations, meaning all models except for the first teacher, the pipeline is reconfigured to accept a combination of original training data and generated pseudo labels, which go through the same preparation steps as before. In the final line of the for-loop the program calls *run_iteration* method which we will discuss next.

The *run_iteration* method in Listing 3.12 handles model initialization and model training and evaluation. After the model is trained the method generate, sort, and resample, the new pseudo labels. The new labels are then combined with the previous training images, and finally, the pseudo labels which are added to the training images are then removed from the dataset object of unlabeled images. In the following section, we will discuss in more detail what happens inside this method.

```python
for idx, curr_model in enumerate(models_list):
    # Update settings and hyper-parameters
    iteration = int((np.floor(idx/2.0)))   # 0,0,1,1 etc
    dir_name = str(iteration)+'_'+curr_model["name"]
    conf["log_dir"] = "./logs/{}/{}".format(project_time, dir_name)
    # Copy model hyper-parameters to config dictionary
    for (key, value) in curr_model.items():
        conf[key] = value

    # Prepare the dataset
    if idx is 0:
        # If first iteration: create dataset
        ds = create_dataset(conf)
        ds["unlab"] = create_unlab_ds(conf)
        ds_bin = [tf_bincount(ds["clean_train"],conf["num_classes"])]
        ds["combined_train"] = ds["clean_train"]
    else:
        # Rest of iterations: refresh training data
        ds["train"] = prepare_for_training(
            ds=ds["combined_train"],
            ds_name='train_'+dir_name,
            conf=conf,
            cache=True
        )

    # Run one iteration of teacher-student semi-supervised training
    run_iteration(conf, ds, ds_bin, sanity)
```

Listing 3.11: Code excerpt from our framework backbone which handles switching out datasets and models during the iterative training, mostly by calling on other custom methods.

```python
def run_iteration(conf, ds, datasets_bin, sanity):
    model = create_model(conf)
    callbacks = create_callbacks(conf)
    class_weights = get_class_weights(ds["train"], conf)

    history = model.fit(
            ds["train"],
            steps_per_epoch = conf["steps"]["train"],
            epochs = conf["num_epochs"],
            validation_data = ds["val"],
            validation_steps = conf["steps"]["val"],
            class_weight = class_weights,
            callbacks = callbacks,
    )
    # Evaluate the model
    evaluate_model(model, history, ds, conf)

    # Generate pseudo labels from unlabeled dataset
    count = {"findings": 0, "total": 0}
    pseudo = {"pred_list": [], "lab_list": [], "name_list": []}
    pseudo, count = generate_labels(
        pseudo, count, ds["unlab"], model, conf
    )

    # Sort pseudo labels in order of highest confidence to lowest
    pseudo_sorted = custom_sort(pseudo)

    # Sample from pseudo labels, and combine with training data
    datasets_bin, added_samples = resample_and_combine(
        ds, conf, pseudo, pseudo_sorted,
        datasets_bin, limit=conf["class_limit"]
    )

    # Remove pseudo labels from unlabeled dataset
    ds["unlab"] = reduce_dataset(ds["unlab"], remove=added_samples)
```

Listing 3.12: The *run_iteration* method handles one step in the iterative teacher-student framework. It sets up the model, callbacks, potential class weights, train and evaluate the model, generates pseudo labels, sorts them, and combine them with the training data to be used by the model in the next iteration. Lastly the combined pseudo labels are removed from the unlabeled data to stop them from being generated in multiple iterations.

3.4.2 Generating new pseudo labels

We generate pseudo labels from the corpus of unlabeled data by iterating over it and running every image through our predictive model. For the first iteration through the unlabeled dataset, it is the teacher model which handles the predictions, and the second time it's the student. This is the most time consuming process of our system, and depending on the size of the unlabeled dataset and the depth of the model, it can take from half an hour to many hours for each run through the dataset. For Hyper-Kvasir, with its almost one hundred thousand unlabeled images and the shallowest EfficientNet model (EfficientNetB0), the process takes roughly forty minutes. The model predicts a probability distribution over the set of classes. The class with the best score is checked against a set threshold; if above this threshold the image is deemed to be of interest and marked to bee incorporated into the training data, and if it is below the threshold it is assumed to be out of domain and rejected. In the case of class probability above the set threshold we mark the image by adding it's file path, together with its suggested label and probability score to a python dictionary. The dictionary of pseudo labels is saved to file system so that the system can easily be resumed from any point in the event of a system failure.

In Figure 3.16 is a plot outputted by the system every 500 image throughout the predictive procedure. In this plot, every bar is representing the number of marked images of interest and its corresponding class name. This particular run is obtained from a student model in one of my experiments.

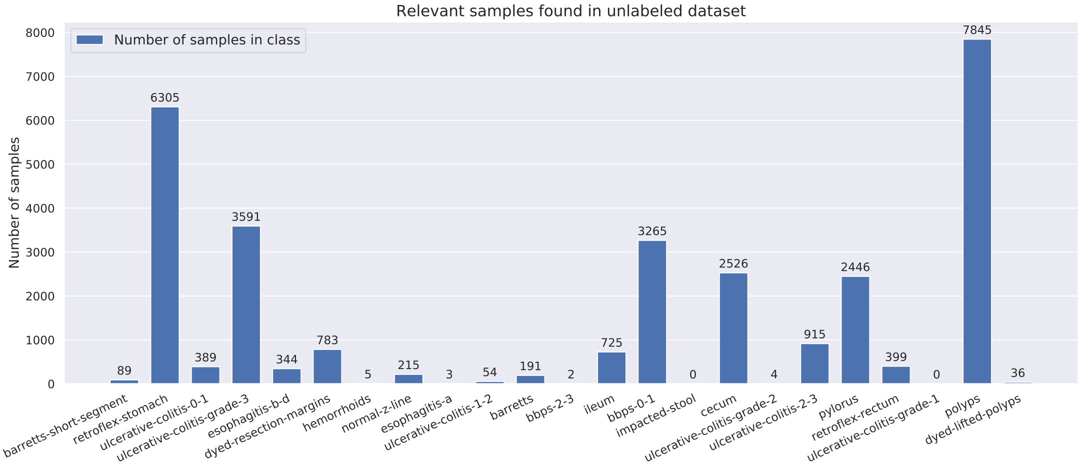


Figure 3.16: The number of images of interest found from the unlabeled dataset. In this figure, we processed 99 thousand images in the Hyper-Kvasir unlabeled dataset and kept all samples which had greater than 90% probability of belonging to one of the 23 classes in the labeled dataset.

When we use the trained teacher model to generate new pseudo labels from the dataset we set a threshold for which the predicted image is saved if it is above the set

```
sorted_list = list(zip(pred, lab, name))
sorted_list.sort(key=lambda x: x[0], reverse=True)

pred_sorted = [row[0] for row in sorted_list]
lab_sorted = [row[1] for row in sorted_list]
name_sorted = [row[2] for row in sorted_list]
```

Listing 3.13: Implementation of parts of a custom sorting method. A dictionary is sorted in decreasing order according to the first element of the dictionary, which in our case is the models class probabilities.

value. Here we have two options, either to set the threshold low and extract a large dataset with high uncertainty, but get the most of our minority classes. Or we can set the threshold high, and gather a dataset with lower uncertainty, but might miss more samples for the minority classes. In our experiments we have set the threshold relatively high due to prevent too many images added, and running out of memory, which occasionally happened on our system.

The next step is to sort the images of interest in descending order, so the first images are the ones the model has the highest confidence in. Reason for this is so the system can select the best pseudo labels in the case of adding a subset and not all. To handle the sorting, we implemented a custom sorting method, the critical part is showed in Listing 3.13. The sorting method first unpack the dictionary with relevant findings, then sort everything with respect to the probabilities.

The sorted samples from the unlabeled dataset is then resampled based on the original distribution of samples in the training data, or by a set threshold in the configuration dictionary. Since the original training dataset is resampled (See Section 3.2.8) we also want the the dataset to be resampled to mimic the previous distribution of samples. This is achieved in two steps; first, for every class of the generated pseudo labels we only add a particular number of samples corresponding to the dissimilarity between the respective class and the majority class. The motive for this is to reduce the effect of class imbalance for the next model. As an example, given a dataset with a majority class of 100 samples, we keep adding new pseudo labels until every class has 100 samples and the process is saturated. See Figure 3.17 for a visualization of this process. In this figure, each same colored column represents the number of samples in the training dataset. The first column (blue) is number of samples for each class in the original training dataset, second column (orange) is the total number of samples after combining the original training dataset with the first round of generated pseudo labels, and so on. For the third class from the left (*ulcerative-colitis-0-1*) we see from Figure 3.16 that the model found 389 new samples, and in the dataset distribution (Figure 3.17 third green column from left) we see that all 389 samples were added to the dataset. While for the fourth class from the right (*retroflex-rectum*) the model found 399 new pseudo labels but only added 113 of them before being fully saturated.

For further improvements in efficiency, a method for active resampling can be implemented. This method would find new pseudo labels with prediction score higher

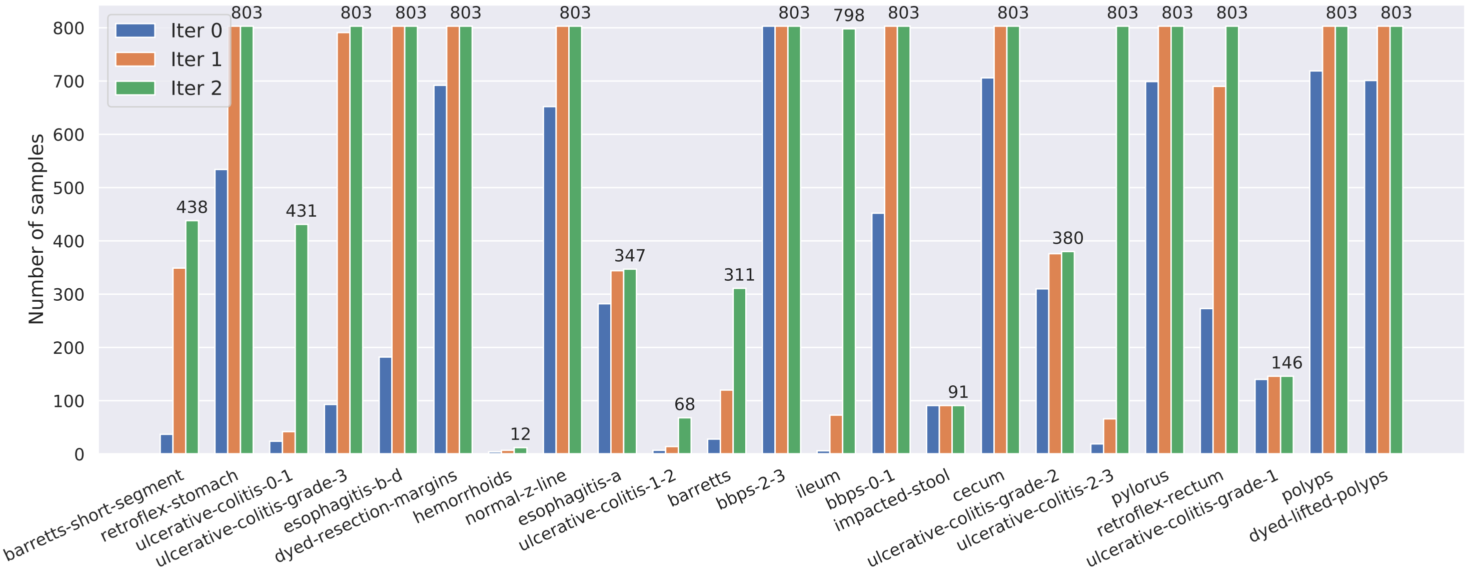


Figure 3.17: Class distribution of training dataset through two iterations of adding new pseudo labels generated by the teacher. Blue bars are the initial training dataset. The threshold for the max amount of samples per class is set by the majority class which in this case is *BBPS-2-3* with its 803 original samples.

than the set threshold and replace pseudo labels with lower probability score instead of concatenating them to the other labels.

Subsequently, we create a new tf.data.Dataset of the combined training dataset and generated pseudo labels. And before we switch the teacher model with the student model and start the process over, we send the new training dataset containing pseudo labels through parts of the input pipeline to sample the dataset to get a uniform distribution of class samples. In this step, the dataset is also passed through the normal transformations to augment the images, cache, and batch the dataset.

Inspecting the inferred pseudo labels

Since the unlabeled dataset does not contain labels, it is difficult to fully comprehend the results of the extracted data and professional verification is necessary. We create a module to visualize the different pseudo labels generated by the teacher model. This module insert six pseudo labels from each class into a figure with one row for each class. If the teacher found less than six pseudo labels then black images are inserted instead. Above each image is the probability score produced by the model. The images are sorted according to probability score in descending order to make sure the endoscopist sees the most prominent samples, which also is the same order the generated pseudo labels are injected into the training data for the next model to train on.

We have contacted a field expert to verify some of our results after generating pseudo labels from the unlabeled dataset. The report we presented to the expert endoscopist is seen in Figure 3.17.

In Table 4.2 on page 98, we provide a report given by a medical physician which shows how well this particular teacher model correctly classified pseudo labels. Occasionally the pseudo label samples generated by the aforementioned module did not adequately transfer confidence in the models ability to correctly learn important class features. To solve this we added a second module which output a grid of generated pseudo labels for a given class. This made it easier to inspect and verify that the model had picked up good features.

3.4.3 Feature drifting

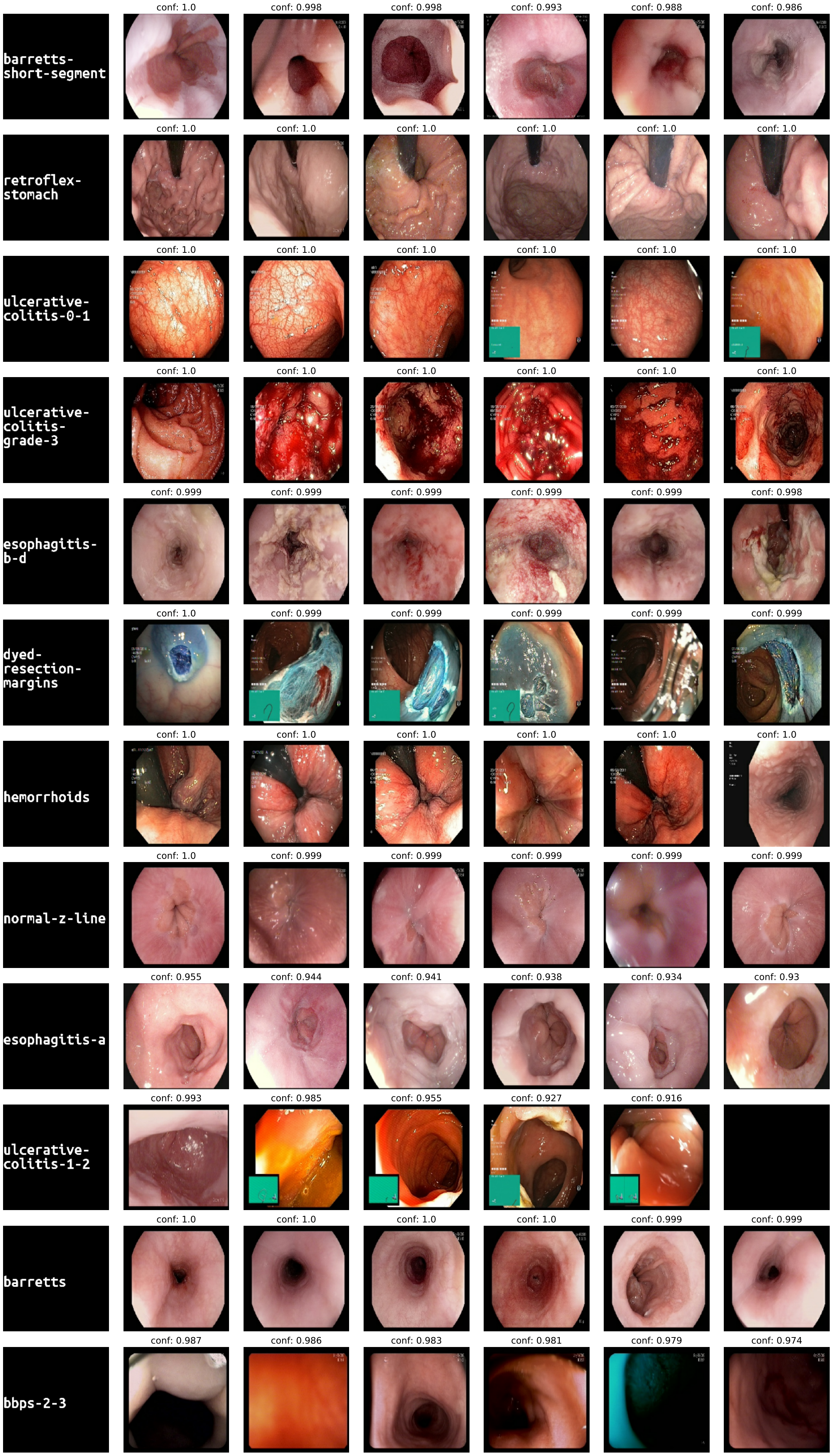
After generating new pseudo labels with the teacher student and training the student model, we noticed that the minority classes had a tendency to drift away from its originally learned features. An example of this is seen in Figure 3.18. In this example, we have first trained the teacher on the labeled data and used it to generate pseudo labels from the unlabeled dataset. We then resampled and combined the new pseudo labels with the original labeled training data and initialized a student model with the same parameters as the teacher model to train on it. Then in Figure 3.18b, we see that when the teacher model is replaced by the student model and we run the process again, the teacher in the second iteration have learned different features than the initial teacher model.

One remark which we suspect is connected to the previously mentioned feature drifting is that most of our experiments inherit a common predicament, is that for the smallest minority classes the model don't seem to be able to correctly learn the class features. In the case of Figure 3.18, the respective class have only 6 samples in the original dataset, of which only 4 is accessible by the training dataset. This causes the predictive model in turn to add either samples distant in feature space, or not add any samples from the unlabeled dataset at all. In either case this is not good for our system as it depends on the initial classes being learned carefully so the model have the means for predicting new and relevant samples. This in turn leads to 2 distinct cases;

1. No new samples are added from the unlabeled dataset.
2. Unwanted samples are added.

In the first case (1) the predictive model did not find new samples from the unlabeled dataset. This is most likely because the class have too few samples to train a model which generalize well. Consequence of this is the predictive model will come up short if the threshold TH_H is set relatively high, like $TH_H = 0.90$. Tweaking the threshold value more could moderately improve the results, but we chose to not follow this thread.

For the second case (2) the model learns some features with high confidence, but perhaps not the important features relative to the particular class. Then when we run the model over the unlabeled dataset it will find new samples within the learned feature space and the samples are added to the training dataset. These unwanted new


barretts-short-segment
conf: 1.0
conf: 0.998
conf: 0.998
conf: 0.993
conf: 0.988
conf: 0.986
retroflex-stomach
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
ulcerative-colitis-0-1
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
ulcerative-colitis-grade-3
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
esophagitis-b-d
conf: 0.999
conf: 0.999
conf: 0.999
conf: 0.999
conf: 0.999
conf: 0.998
dyed-resection-margins
conf: 1.0
conf: 0.999
conf: 0.999
conf: 0.999
conf: 0.999
conf: 0.999
hemorrhoids
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
normal-z-line
conf: 1.0
conf: 0.999
conf: 0.999
conf: 0.999
conf: 0.999
conf: 0.999
esophagitis-a
conf: 0.955
conf: 0.944
conf: 0.941
conf: 0.938
conf: 0.934
conf: 0.93
ulcerative-colitis-1-2
conf: 0.993
conf: 0.985
conf: 0.955
conf: 0.927
conf: 0.916
barretts
conf: 1.0
conf: 1.0
conf: 1.0
conf: 1.0
conf: 0.999
conf: 0.999
bbps-2-3
conf: 0.987
conf: 0.986
conf: 0.983
conf: 0.981
conf: 0.979
conf: 0.974

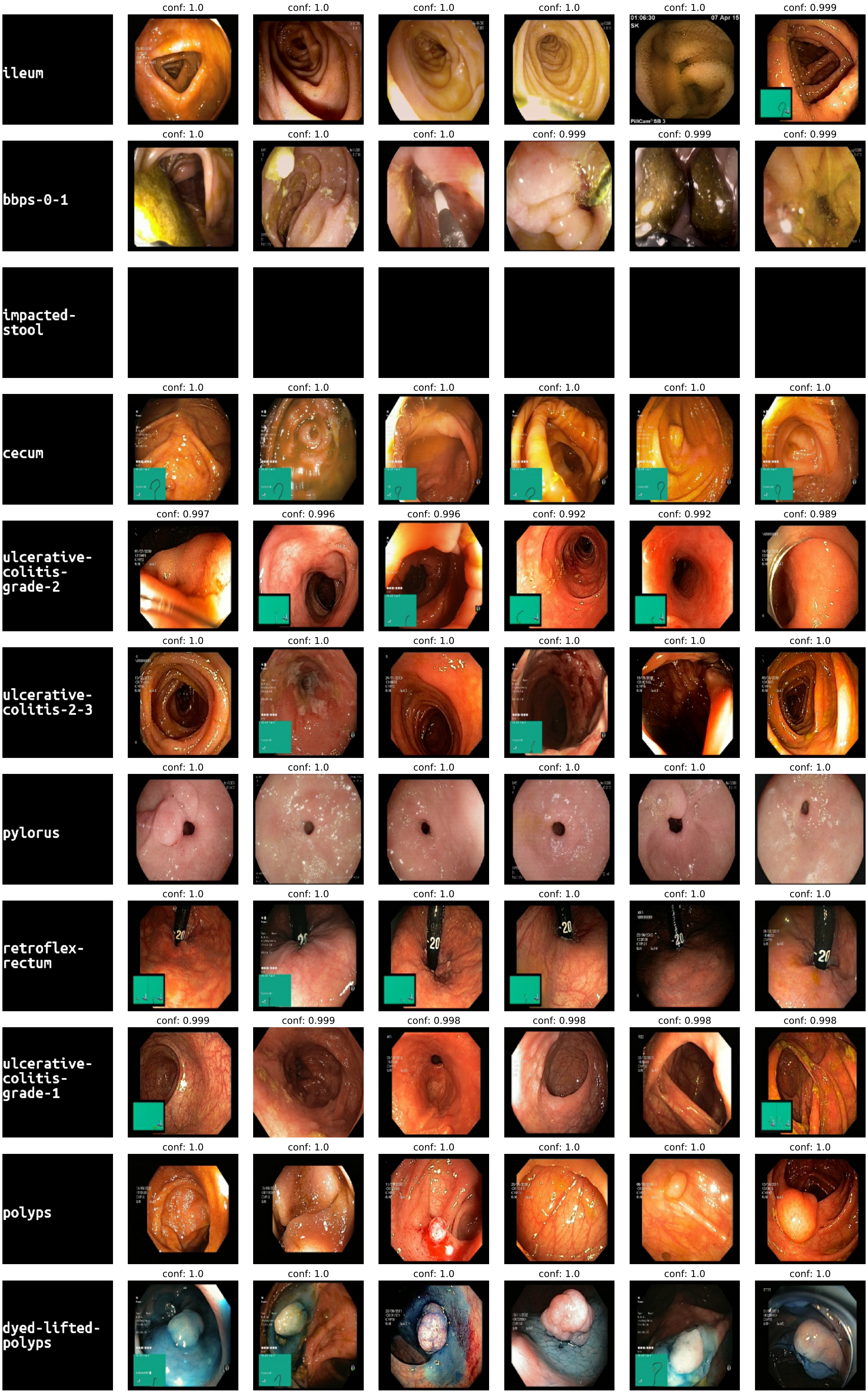
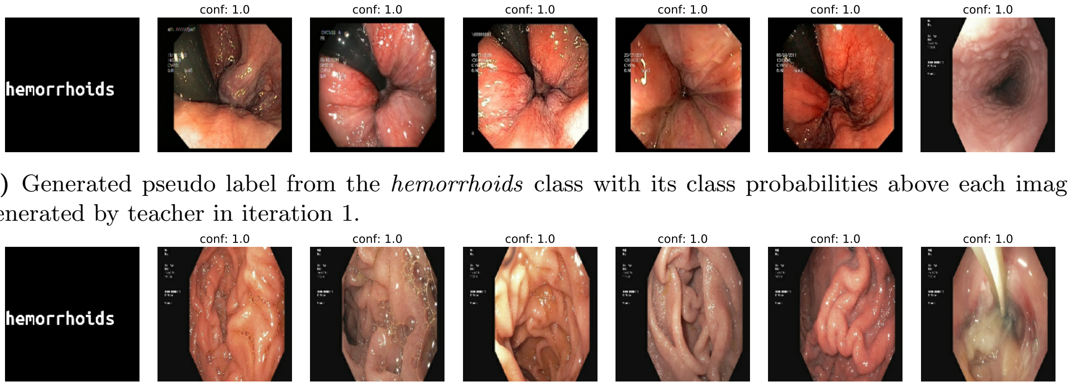


Figure 3.17: Generated pseudo label by teacher model trained on labeled data. The images which are blacked out represents no detected pseudo labels for that class.



(a) Generated pseudo label from the *hemorrhoids* class with its class probabilities above each image. Generated by teacher in iteration 1.

(b) Generated pseudo label from the *hemorrhoids* class with its class probabilities above each image. Generated by teacher in iteration 2.

Figure 3.18: Pseudo labels of the hemorrhoids class, generated by teacher 1 and teacher 2 from Hyper-Kvasir unlabeled dataset.

samples have a especially large impact on the minority classes because so few samples are available, and adding a large amount of new samples are simply adding noise for the student model which again will learn unimportant features.

To improve upon the performance we suggest the following changes; (1), collect more data for the minority classes, this will help the model learn the correct features. (2), do K-fold validation splits, or cross dataset validation to have more data available for training.

3.4.4 Evaluation methods and metrics

To evaluate our semi-supervised teacher-student model we create a script, evaluate_model.py, which is part of the training pipeline. This script is initiated for every iterative step in our teacher-student system. This way every model is evaluated equally, and the system is more transparent when it comes to tracking changes in performance, or to monitor performance gain during program execution.

The scripts takes the trained model, TensorFlow's training history, and the test dataset as input. The test dataset is filtered to only include the images and is then run through the model to create an array of predicted labels. With the predicted labels and true labels we generate a confusion matrix with the python package scikit-learn [94], and generate a classification report which includes metrics for recall, precision, F_1-score and accuracy. The scripts also generates and saves plots of the loss and accuracy metrics from model training.

If the model is configured for handling binary datasets the ROC-curve and its AUC value is generated and saved. The ROC curve is a great diagnostic tool that helps in the interpretation of binary models, but it can be misleading in cases with severe class imbalance and few samples in the minority classes. This is because a small number of correct or incorrect predictions can result in a large change in the ROC Curve or ROC AUC score. Precision-recall curves (PR curves) are recommended for highly skewed domains where ROC curves may provide an excessively optimistic view of the performance [95]. However, this was considered out this projects scope due to time constraints.

3.5 Summary

In this chapter, we have presented the work of creating a VCE dataset, Kvasir-Capsule, and the details of how it was gathered from hospital examinations, labeled by three MSc students and verified by a endoscopist expert. This dataset relates back to the first research objective as stated in Section 1.2, which stated the need for a open source labeled VCE dataset from the GI tract. Class imbalance is a common issue for image classification tasks, and we have in this chapter taken direct actions to minimize the effect introduced in the models performance. An efficient and specialized input pipeline was then introduced which handles sparse medical image data with ease. This pipeline was used to deliver images to a EfficientNet model which we manually tuned to a point were the results were deemed sufficient.

The next step was to develop the semi-supervised teacher-student framework, which by introducing pseudo labels from the unlabeled image corpus to the training data makes the input training data less sparse. By experimenting with a open-source colonoscopy dataset, Hyper-Kvasir, we discovered and discussed some flaws in the system. The development of this framework relates back to the second research objective, where we stated the need for self-learning framework to incorporate the vast amounts of unlabeled medical images. In the next chapter, we will look at our conducted experiments with the framework, and discuss the results.

Chapter 4

Experiments and results

Artificial intelligence is predicted to have profound effects on the future of video capsule endoscopy technology. The potential lies in improving anomaly detection while reducing manual labor. However, medical data is often sparse and unavailable to the research community, and qualified medical personnel rarely have time for the tedious labeling work. In this respect, we presented Kvasir-Capsule, a large VCE dataset of labeled and unlabeled images of the GI tract, and a self-learning teacher-student framework in the previous chapter. Recent work in the machine learning community has shown great improvements regarding unlabeled data value, and semi-supervised learning algorithms are successfully applied in different medical image analysis[96] using self-learning [97], [98].

In this chapter, we look at using our teacher-student framework introduced in the previous chapter, for the analysis of semi-supervised learning on gastrointestinal tract endoscopy with the purpose of discovering potential methods of improving the quality of models trained on Hyper-Kvasir, and perform initial experiments with the newly created Kvasir-Capsule dataset. Taking advantage of the large amounts of unlabeled data could potentially lead to higher quality models for a given dataset, and to put this theory to the test, we will do extensive analysis of our framework on both datasets. Before presenting our results, we first look at some important evaluation method and metrics to define and standardize our testing methods, then we will move on to some initial testing to ensure our teacher models behave as expected. In the final part of this chapter we will run experiments with iterative teacher-student training on both Hyper-Kvasir, and Kvasir-Capsule.

4.1 Experiment management

While working on this thesis, we have found that getting good results from a single model trained on a single dataset is one thing, but keeping all experiments and corresponding results and findings organized and having a process that lets us draw valid conclusions from them is quite another. This is why we have dedicated this first section to lay some ground rules to standardize further experiments. At later stages when we will be training up to 12 models for a single experiment this become essential for keeping up an efficient workflow. Another benefit of proper experiment management is that our results will be easier to reproduce for others. Next, we will discuss how we have chosen to keep track of our experiments, and then move on to some important evaluation methods and metrics.

4.1.1 Keeping track of experiments

Training multiple networks using a wide range of hyperparameters can become chaotic. To overcome this issue, we have all our hyperparameters stored in a single python dictionary which easily can be stored in a configuration file. This dictionary handles all framework parameters, like the number of added unlabeled samples and how the number changes during the iterative process of a teacher-student model. One of the parameters in this configuration dictionary is where the log directory is placed. For every run we dedicate a directory for saving all plots, lists, evaluation metrics and trained models and its weights so that we can keep track of what experiments have been run in the past, and what the results where. This log directory became very handy in the case of 'out of memory' errors, in which we could go back and load a trained model, or pseudo labels generated from running through the unlabeled dataset, saving us from having to run experiments multiple times.

4.1.2 Evaluation method and metrics

To evaluate whether or not the model performance increase during training, we monitor sparse categorical cross-entropy. Ideally, we would also monitor other metrics like recall, precision and F_1-score during training but due to time constraints we were not able to implement this. Instead we focus on measure the metrics on the test dataset after the model is trained.

The model evaluation script, *model_evaluation.py*, generates a classification report as seen in Table 4.1 which presents precision, recall and F_1-score per class, and the support for each given class. The metrics for each class are also given as macro and weighted averages.

To visualize how the model perform against each other we use the data from the classification report, see Figure 4.1. This figure has data points for each class of the teacher (red) and student (blue) models for three iterations of switching the teacher with the student. Figure 4.1a has a linear regression model fit for each teacher and student, while Figure 4.1b has a plot of the contours to make it easier to spot where the bulk of the data points are located.

From the matrix we get by measuring how well the model perform on the test dataset we can create a confusion matrix to visualize the performance further. Figure 4.2 is an example of such a confusion matrix generated for every model trained by our system. This particular confusion matrix is generated after training on the Hyper-Kvasir dataset which has 23 classes and is therefore quite large. The diagonal line going from top left to bottom right is the data points which have been correctly classified. The colors in the confusion matrix represents the percentage of total samples.

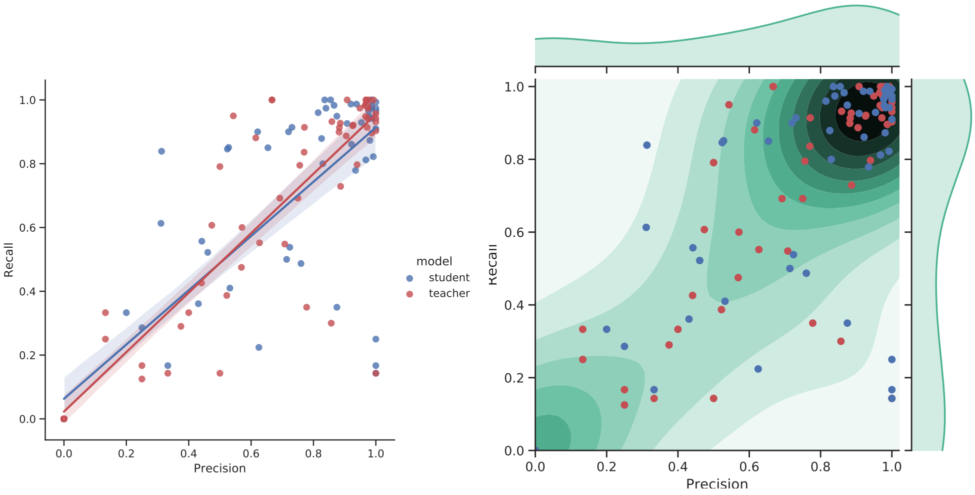
To verify the performance of how well our teacher model selected pseudo labels from

Class	Precision	Recall	F_1-score	Support
barretts-short-segment	0.250	0.125	0.167	8
retroflex-stomach	0.966	0.983	0.974	115
ulcerative-colitis-0-1	0.133	0.333	0.190	6
ulcerative-colitis-grade-3	0.778	0.350	0.483	20
esophagitis-b-d	0.692	0.692	0.692	39
dyed-resection-margins	0.883	0.913	0.898	149
hemorrhoids	0.000	0.000	0.000	1
normal-z-line	0.770	0.836	0.801	140
esophagitis-a	0.441	0.426	0.433	61
ulcerative-colitis-1-2	0.000	0.000	0.000	2
barretts	0.333	0.143	0.200	7
bbps-2-3	0.987	0.896	0.939	173
ileum	0.000	0.000	0.000	2
bbps-0-1	0.970	0.990	0.980	97
impacted-stool	0.543	0.950	0.691	20
cecum	0.949	0.974	0.961	152
ulcerative-colitis-grade-2	0.627	0.552	0.587	67
ulcerative-colitis-2-3	0.000	0.000	0.000	5
pylorus	0.974	0.980	0.977	150
retroflex-rectum	0.859	0.932	0.894	59
ulcerative-colitis-grade-1	0.375	0.290	0.327	31
polyps	0.967	0.948	0.958	155
dyed-lifted-polyps	0.905	0.887	0.896	151
accuracy			0.855	1610
macro avg	0.583	0.574	0.567	1610
weighted avg	0.858	0.855	0.854	1610

Table 4.1: Classification report generated by system after training of model. At the bottom in the table are a summary of accumulated metrics.

the unlabeled dataset we had an expert endoscopist review the predicted pseudo labels outputted by our system. The system-generated report we tested is seen in Figure 3.17 on page 90. The physician found that the results varied depending on the specified class. For some classes all of the generated pseudo labels were correctly classified, and some classes had mixed results and for one class the model had not been able to correctly learn the relevant image features and would therefore not predict any pseudo labels. In Table 4.2 is a summary of the report made by the expert endoscopist. As a metric for comparing the review with a later pseudo label report we note that the system correctly classified 60.14% of the images according to the endoscopist.

From the verification report it is especially noticeable that the model has not correctly learned the features of the class *impacted-stool* since no images were found in the unlabeled data. This problem persisted through all our experiments. In some



(a) Linear regression fit.

(b) Contour plot

Figure 4.1: Scatter plot of precision and recall for teacher (red) and student (blue) models after three iterations. Each data point represents a class.

experiments impacted-stool would be classified as the class *polyps* which share similar features. This is seen in Figure 4.3. Here the two selected samples have overlapping features, and the dividing boundary which separates the two classes are further clouded by the color changing augmentation performed on the training data. For further tweaking of our system we believe that fine-tuning the augmentation performed by the input-pipeline, or use AutoAugment to automatically search for improved data augmentation policies, would effect the models ability to separate the classes better.

4.1.3 Hardware details

To ensure that our results are easily reproducible, we include the hardware specifications used for all training and evaluation sessions. We have used two separate systems, one which is equipped with commercial grade hardware easily accessible (Table 4.3) and the other is a powerful Nvidia DGX-2 AI system capable of 2-petaFLOPS tensor performance. The DGX-2 AI system is part of Simula Research Laboratories heterogeneous cluster and has dual Intel(R) Xeon(R) Platinum 8168 CPU@2.70GHz, 1.5TB of DDR4-2667MHz DRAM, 32TB of NVMe scratch storage, and 16 of NVIDIAs Volta 100 GPUs interconnected using Nvidias NVlink fully non-blocking crossbars switch capable of 2.4TB/s of bisectional bandwidth. The system are running Ubuntu 18.04 LTS OS and use Cuda version 10.1.243. All experiments run on this system are performed on a single Volta 100 GPU equipped with fast 32GB HBM memory. This was needed because we ran into video memory issues when training with full image resolution on large networks, and main memory issues when growing the training dataset with pseudo labels fetched from a large corpus of unlabeled images.

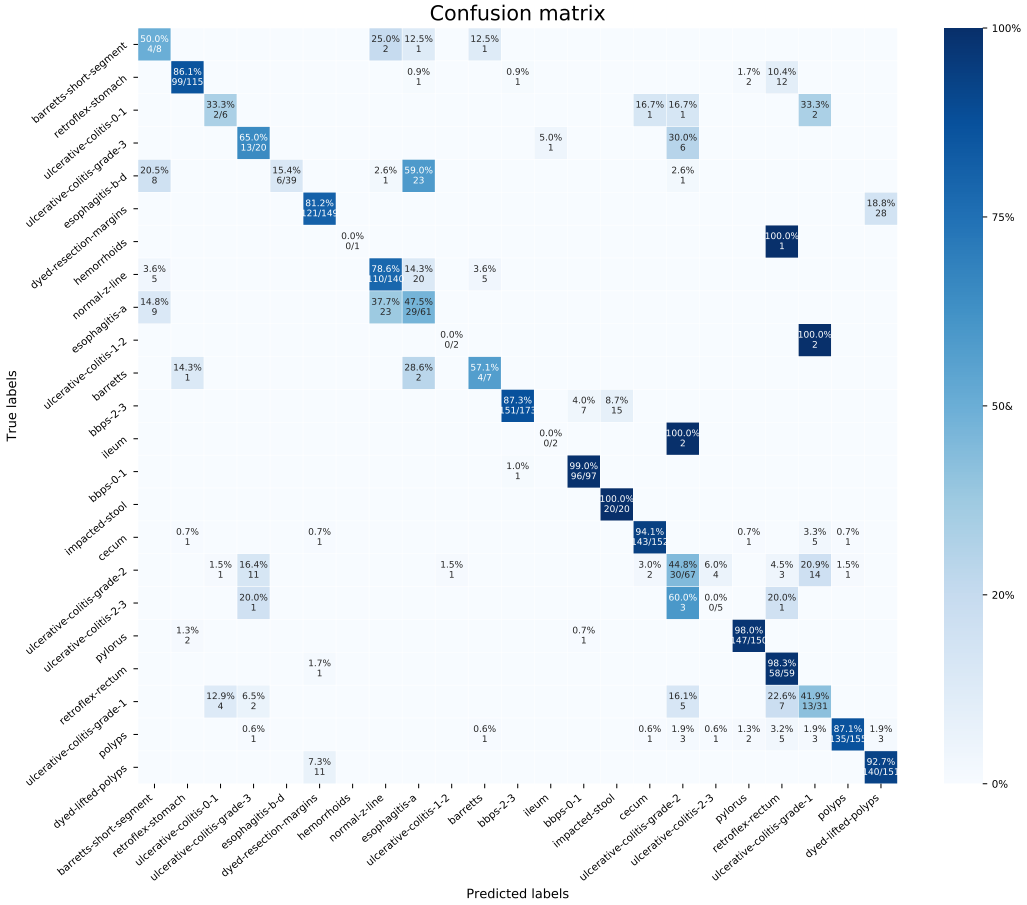


Figure 4.2: Confusion Matrix of a EfficientNet model trained on Hyper-Kvasir dataset with 23 classes. The columns represents the models predicted labels and the rows represent the true dataset label.

4.1.4 Network architecture

For our experiments, we are more concerned with the possibility of improving medical image classification tasks with greatly imbalanced datasets than getting state-of-the-art results. We therefore use the tested and proven EfficientNet for our training, instead of creating a custom model architecture. This network suits us good for three reasons; (1), pretrained ImageNet weights are made accessible by the author, (2) compound coefficient scaling makes it easy to test different network dimensions, and (3) it handles large datasets well. The EfficientNets (B0-B7) are used for all our experiments to standardize the testing and not introduce more levels of uncertainty. This EfficientNet networks is then connected with a pooling layer, a dropout layer, a fully connected layer, another dropout layer and finally a output layer.

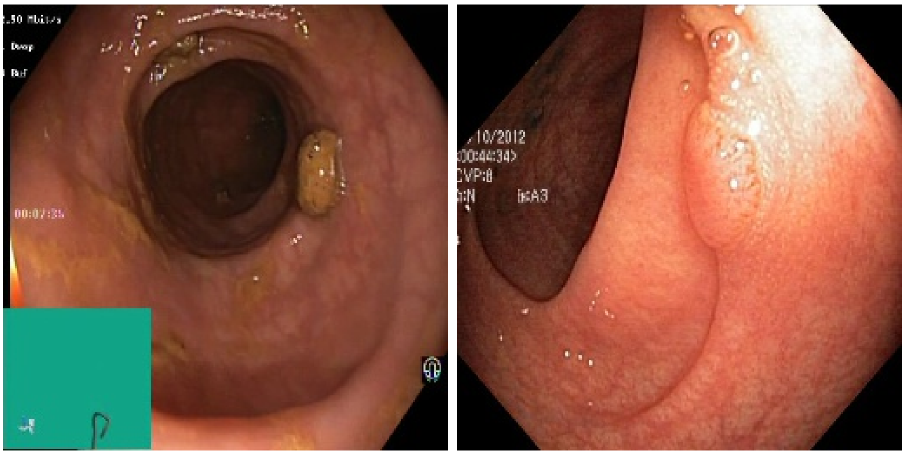
Class	Wrong classification	Correct classification
barrets-short-segment	2,3,6	1,4,5
retroflex-stomach		1,2,3,4,5,6
ulcerative-colitis-0-1		1,2,3,4,5,6
ulcerative-colitis-grade-3		1,2,3,4,5,6
esophagitis-b-d	1,6	2,3,4,5
dyed-resection-margins		1,2,3,4,5,6
hemorrhoids	6	1,2,3,4,5
normal-z-line		1,2,3,4,5,6
esophagitis-a	1,3,4,5	2,5,6
ulcerative-colitis-1-2	1,2,3,4,5	
barretts	1,2,3,4,5,6	
bbps-2-3	1,2,3,4,5,6	
ileum	1, 6	2,3,4,5
bbps-0-1	2,3,4,5,6	1
impacted-stool		
cecum	4	1,2,3,5,6
ulcerative-colitis-grade-2	1	2,3,4,5,6
ulcerative-colitis-2-3	1,3,5,6	2,4
pylorus		1,2,3,4,5,6
retroflex-rectum		1,2,3,4,5,6
ulcerative-colitis-grade-1	1,2,3,4,5,6	
polyps	2,4	1,3,5,6
dyed-lifted-polyps		1,2,3,4,5,6
# Total samples	55	83

Table 4.2: Validation report for Figure 3.17 made by an expert endoscopist. ”1” represent the left pseudo label from the corresponding class, ”2” represent the second pseudo label etc. The class ”impacted stool” had no generated pseudo labels.”

The EfficientNet network is especially flexible because of its use of compound coefficient scaling. The compound scaling coefficient allows us to up-scale and down-scale our network, and introduce more or less stochastic noise to our model. By introducing less noise to the smaller teacher model than to the bigger student model the system will perform better than by using the same network size for both models [64].

4.1.5 Network parameters

For labeled images we re-scale the images to 128 × 128 pixels to reduce memory usage during training. For the smallest model, EfficientNetB0 with 4.7 million parameters, we use a batch size of 128 by default and reduce the batch size when we could not fit



(a) Impacted stool

(b) Polyps

Figure 4.3: Similarities between impacted-stool and polyps classes. Samples taken from impacted-stool and polyps classes of Hyper-Kvasir dataset.

Level	Category	Name	Version
Hardware	GPU	Nvidia GTX 1080 ti	
	CPU	Intel i7-8700K 3.7GHz	
	Memory	G.SKill 3200MHz 32GB	
Software	Operating System	Ubuntu Focal Fossa	20.04
	Library	Python	3.7.6
		TensorFlow	2.1.0
		Keras	2.3.1
		Cuda	10.1.243
		cuDNN	7.6.5

Table 4.3: A table showing the system specifications (hardware and software) for the machine used for training and evaluation sessions of the smaller networks.

the model into the memory. For the largest model, EfficientNetB7 with 65.4 million parameters, we reduce the batch size to 32. We find that using a batch size of 256, 128, 64 and 32 leads to the same performance as long as the dataset are resampled. When the data have dominant class imbalances the smaller batch sizes would lead to faster overfitting than the large ones.

We determine the number of training steps and the learning rate schedule by the batch size for labeled images. The training steps used are calculated by

$$steps = \frac{ds\_size}{bs} \tag{4.1}$$

where ds_size is the number of samples in the given train/test/val dataset and bs is the value for batch size we are using. We find that the network have tendencies to overfit the training data when we use a large value for epochs (see Figure 4.4 for an example). Because of this we use early stopping from the *tf.keras.callbacks.EarlyStopping* package. This monitors the validation loss during training, and if the model do not improve for 5 epochs the training is terminated and the epoch with the lowest loss, and therefor the best weights, is restored. Depending om model size we use, this would happen somewhere around epoch number 30. The larger model would take longer to hit early stopping threshold than the smaller ones.

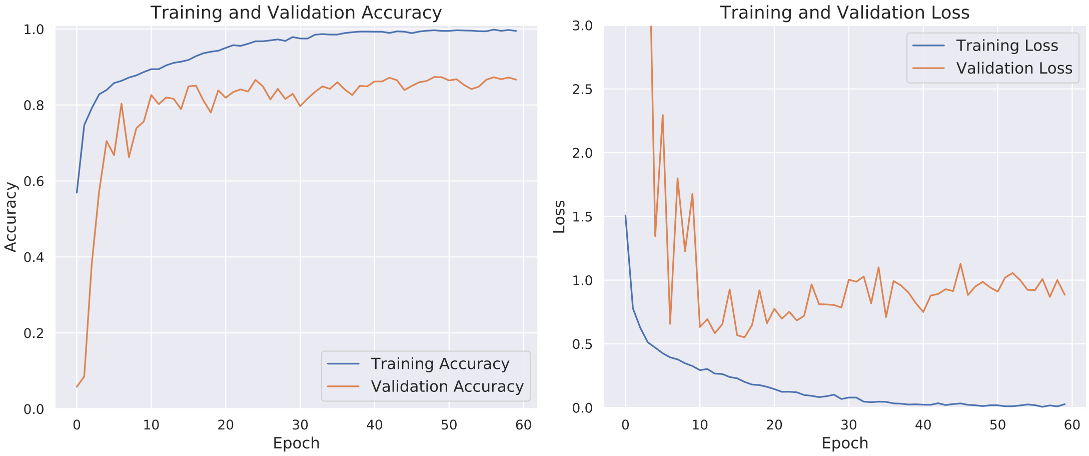


Figure 4.4: Example where the model accuracy for training data greatly outperforms the accuracy for testing data.

The optimizer we use is for our experiments are Adam [49], we use this optimizer because it has shown to perform well for image classification tasks in other studies. Compared to SGD algorithm, we get a stable accuracy gain for every epoch and the loss and accuracy converges quickly. By default in Keras, the learning rate for Adam is set to 0.001, this is a good starting point for us as well. Although Adam uses an adaptive learning rate we tested our model with a inverse time decay learning rate scheduler with $\eta = 0.01$ as initial learning rate, and by an hyperbolic function we gradually reduce the learning rate to slightly below 0.001. This gave slower convergence and confirmed Adam without learning rate scheduler was a superior option.

We find that by using dropout and image augmentation, the model generalizes much better on the validation data, and the teacher model is better at learning the features for the minority classes, which are only represented by a few samples in the training data. For training the teacher model, we use image augmentation multiplier set to 10% and 10% dropout for the final dense layer, and after switching the teacher with the student we increase the image augmentation multiplier to 80% and 30% dropout.

4.1.6 Labeled and unlabeled dataset

We conduct experiments on Hyper-Kvasir dataset since it is open-source and to the best of our knowledge, the largest colonoscopy dataset available. We filter the images into three datasets for training, testing and validation purposes. The data is split so that 60% of the images go to training, 15% go to test data and the remaining 15% go to validation. The training dataset is reduced to 128×128 pixels with bilinear interpolation, shuffled, batched and augmented.

The unlabeled dataset contain 100 thousand images, and the corpus of images are taken from a wide variety of colonoscopies. This dataset is used for generating new pseudo labels which will later be concatenated with the training data. Due to memory and time restrictions we only keep the images which get a probability score for one of the domain classes of 90% and above, the images which receive a lower probability score are considered out of domain images. We then sort the unlabeled data by the probability score within each of Hyper-Kvasir's 23 classes.

4.2 Optimizing the teacher model

A vital part of our teacher-student framework is pseudo label quality. In this regard, we have chosen to dedicate this section for experimenting with different ways to tweak the teacher model to a point where we are pleases after doing a visual inspection of generated pseudo labels. Additionally, an expert endoscopist has also reviewed the labels.

We begin with looking at initialization of model weights and biases, next we take a look at how different class imbalance countermeasures effect the model performance. In the final experiments of this section we test with multiple different image resolutions and neural network dimensions, before summarizing the results and discussing the quality of pseudo labels generated by the teacher model thus far.

4.2.1 Benefits of pre-trained weights

There are many different practices when it comes to initializing the layer weights in neural networks. The reason to why adequate weight initialization is so important in training deep neural nets is to prevent layer activation outputs from exploding or vanishing during the course of the forward passes. If this happens, gradients of the loss function will either be too large or too small to flow backwards, and the network takes longer to converge or could stop learning completely. We use a pre-defined model which handles the layer weights initialization for us, and it has three options; (1) train from scratch, (2) initialize the model with weights trained on ImageNet, and (3) initialize the model with weights trained on Noisy-Student.

We did some initial experimenting with setting a better initial bias for our model, due to the imbalance of our dataset. We followed *A Recipe for Training Neural Networks* by Karpathy who discuss the importance of initialize the layer weights correctly. From this, we derive the following equation:

```
    output_bias = tf.keras.initializers.Constant(output_bias)

    global_average_layer = GlobalAveragePooling2D()
    output_layer = Dense(1, activation=conf["final_activation"],
                         bias_initializer=output_bias)

    resnet50_model = tf.keras.Sequential([
            resnet_model_top,
            global_average_layer,
            output_layer])
```

Listing 4.1: Example code used for creating a model with a given *output_bias* for the final layer. The top model we use here is a ResNet50 model.

$$
\begin{aligned}
p_0 &= \frac{pos}{(pos + neg)} = \frac{1}{1 + e^{-b_0}} \\
b_0 &= -log_e\left(\frac{1}{p_0} - 1\right) \\
b_0 &= log_e\left(\frac{pos}{neg}\right)
\end{aligned}
\tag{4.2}
$$

where *pos* is the number of positive samples and *neg* is the number of negative samples. For testing purposes, we then select all *normal-cecum* images as negative class and all other classes from Kvasir-V2 as positive. By doing this we have seven times as many positive images as negative. Next we enter this into equation 4.2 and find $b_0 = 1.946$. To create our model with the initialized final layer weight we use the code from Listing 4.1. In Figure 4.5 we have plotted calculated model loss for training and validation data for two models, the first has no added initial bias, and the second have initial bias $b_0 = 1.946$. From these results we see that there is a large difference in validation loss on the first epoch, but the two models performance equalize quickly. Because of these results we decided to not go any further with setting initial biases.

The following experiments regard the usage of pre-trained weights. We found that training on Hyper-Kvasir datasets yield best results when using EfficientNet initialized with ImageNet weights, trained using AutoAugment. When we began our venture, these weights were not yet available, and we instead used the original EfficientNet weights. We observed that the model would initialize the training with slightly lower loss when trained with ImageNet-AutoAugment weights. Due to this observation we used these weights for the remaining duration of the study.

We also tested initializing the teacher model with original noisy-student weights but found that loss were fluxing more early in training. When trained for same number of epochs as with ImageNet weights the model would perform worse on the evaluation data, with lower accuracy and lower recall. Also, the weights would cause an worse loss estimate for the first epochs.

As part of this experiment we also tested with initializing the teacher model with no pretrained weights and train from scratch. As seen in Figure 4.6, this reduces the

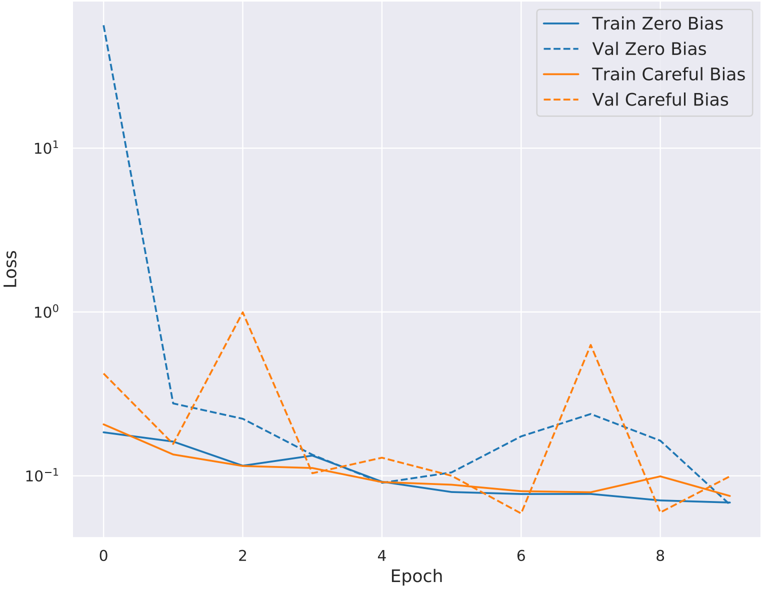


Figure 4.5: Loss computed on training and validation data using carefully initialized weight bias compared against zero initial bias.

training speed, and after 20 epochs of training we got a 15% lower accuracy than when training with ImageNet weights. EfficientNet's building block, MBconv, are initialized with a normal distribution for its normal and depth-wise convolutional layers, and it's dense layers are initialized with a uniform distribution.

The authors of the noisy-student paper remarked that initializing the student with the weights from the teacher model yielded as good results as training from scratch. We did not test this, because it was out of our scope, but is something we want to study at a later time.

4.2.2 Re-sampling versus weighting classes

Given our research topic, it is important to measure how well the model behaves on imbalanced data, and what measures can be taken to improve the model. In this experiment, we want to train three classifiers with three different methods of handling imbalanced distribution of class samples. The first one is a baseline test where we will omit any class imbalance counter measures, the second classifier is trained with class weighting and the third test is with a classifier trained on data sampled in a uniform distribution. See Section 3.2.8 for a more detailed explanation of how the experiments were carried out.

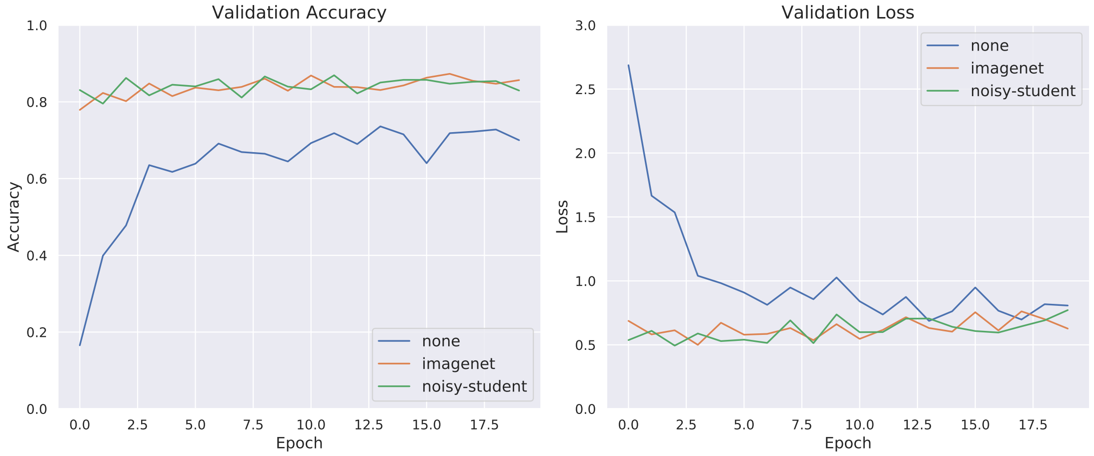


Figure 4.6: EfficientNetB0 trained for 20 epochs with different layer weights initialization. The data points are collected from measuring the model performance on the validation set during training. Blue line; no pretrained weights, red line; pretrained weights on ImageNet with AutoAugment, green line; pretrained weights from noisy-student.

This will help us to determine which method is best suited for handling the class imbalance of our semi-supervised learning algorithm. For our sampling method, we chose a uniformly distribution of samples which means minority classes are oversampled and majority classes are undersampled depending on the size of the class, see Section 3.2.8 for how this was implemented. For class weighting, we found each weight as explained in Section 3.2.8, and the weights are found in Table 4.4. To fully understand which method perform better we have conducted a total of three experiments. In the first experiment we created a new binary class dataset of Hyper-Kvasir to create a ROC curve (Section 2.4.2). We marked the original class *pylorus* as negative, and all the remaining 22 classes were marked as positive. We chose *pylorus* as the negative class because from earlier experiments we know this class have well defined features which are easy to learn by the model. The final dense layer in the model were changed from softmax to a sigmoid and we then trained the model with data augmentation and dropout, for 15 epochs. In the second experiment, we used the full multiclass Hyper-Kvasir dataset, and the models final dense layer with sigmoid activation function were switched back to a softmax activation function. Multiclass dataset makes it more difficult to create a ROC curve so we instead look at training loss and accuracy. With this new output layer and dataset we repeat the previous experiment, for the three baseline, weighted and resampled models. The final experiment were conducted to verify we achieved the same results on Kvasir-Capsule dataset.

We found that the model trained on the resampled dataset performed better than both the baseline model and the model trained with weighted classes. In Figure 4.9, we have visualized the ROC curve from the first experiment where a new binary dataset was created. Here we see the model with resampled data slightly outperformed the baseline

Name	Samples	Weight
Lymphoid Hyperplasia	224	9.3091
Foreign Bodies	590	2.9459
Ulcer	272	6.4646
Erosion	206	8.8658
Blood	22	93.091
Pylorus	938	2.1449
Angiectasia	771	2.5434
Erythematous	132	14.322
Reduced Mucosal View	932	2.1014
Normal mucosa	15,853	0.1209
Ileo-cecal valve	1120	1.6743

Table 4.4: The weights used for computing loss in backwards propagation while training the model in Figure 4.8 and the amount of samples for each class in the training dataset. The weights are calculated as explained in Section 3.2.8. In the middle column is the number of samples in the training data which the network use to update its parameters, and it is these numbers which are used for calculating the class weights.

model, and the model trained with weighted classes performed the worst. We also include the accuracy, loss, recall, precision and AUC for all three models in Table 4.5, as well as the confusion matrix for the model trained on the resampled dataset, in Figure 4.7. The second experiment on Hyper-Kvasir is visualized in Figure 4.10 and gave almost the same results as the first, but in this experiment, the baseline model perform as good as the resampled model. For the final verification experiment, we trained a model using weighted classes on the Kvasir-Capsule dataset. The results are seen in Figure 4.8. The respective weights for each class are found in Table 4.4. In this experiment, the model was trained on split_0 with EfficientNetB4 model for 25 epochs. Here we see how the model fails to learn from the training data when using class weights.

	Accuracy	Loss	Recall	Precision	AUC
Baseline	0.9925	0.0294	0.9951	0.9965	0.9993
Weighted	0.8719	0.4040	0.8608	0.9976	0.9846
Resampled	0.9931	0.0200	0.9979	0.9945	0.9995

Table 4.5: Accuracy, loss, recall, precision and AUC metrics for three models, with three different methods of handling class imbalance.

4.2.3 Effect of varying image resolution

In Section 2.3.5, we discussed the diminishing effect of image resolution for model performance. In this section, we want to test how model performance degrade for a reduction in image resolution in colonoscopy images. This will enable us to chose an

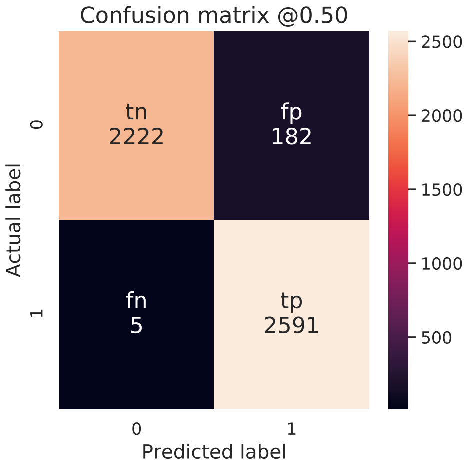


Figure 4.7: Confusion matrix from model trained on resampled data.

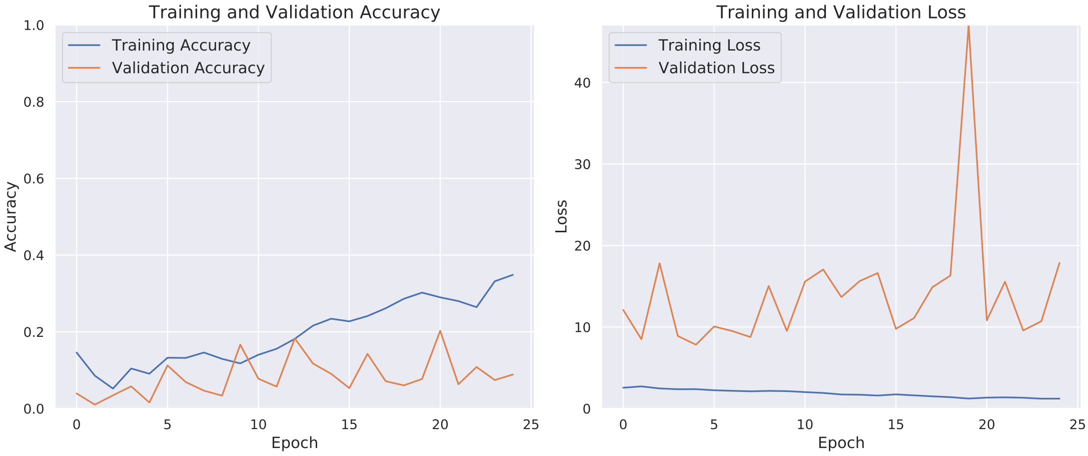


Figure 4.8: Loss and accuracy calculated on training and validation data respectively for a EfficientNetB4 model trained for 25 epochs with weighted classes. The weights are listed in Table 4.4.

image resolution to use for further experiments, which suits our requirements of (1) being large enough to allow the network to learn the proper features and (2) fit a batch of images in GPU memory.

The ImageNet weights are trained on images with resolution of 224 × 224, so to make sure pre-trained ImageNet do not bias our model towards any particular image resolution we randomly initialize the weights and train from scratch. Because our model do not get any assistance from pre-trained weights we increase the number of epochs

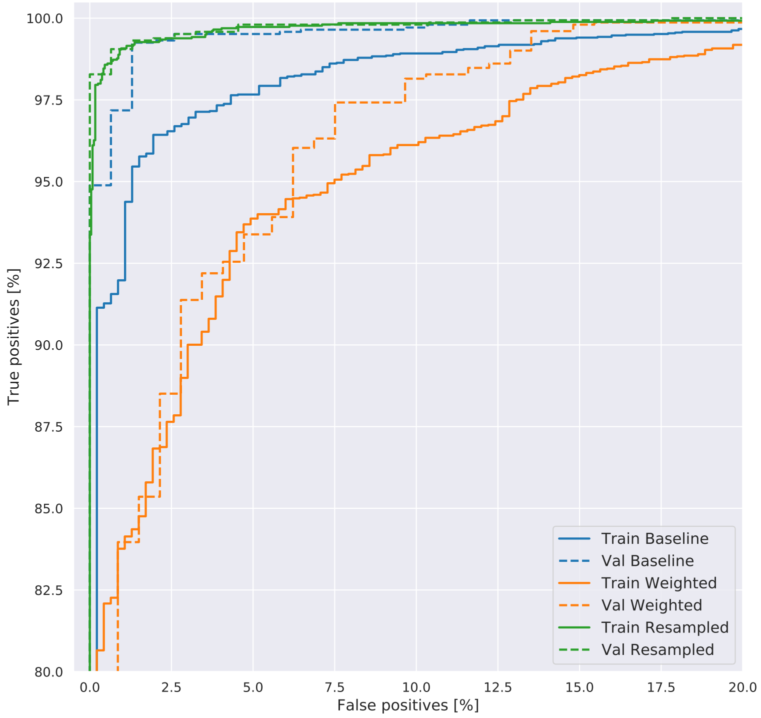


Figure 4.9: ROC curve with results for baseline, weighted and resampled models trained on a binary version of Hyper-Kvasir where the anatomical landmark ”pylorus” is positive and all other classes are marked as negative.

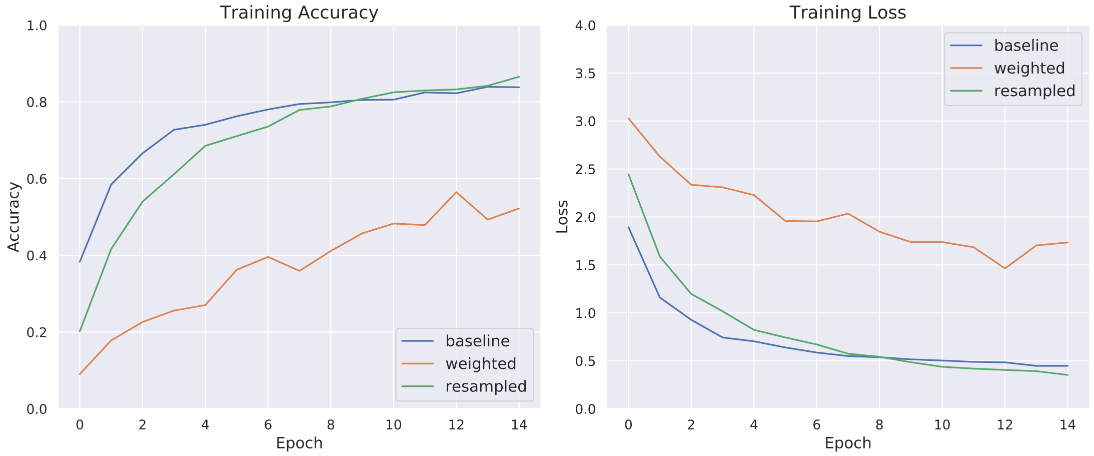


Figure 4.10: Accuracy and loss calculated on Hyper-Kvasir labeled training data for three models. Baseline model uses neither class weighting nor resampling. Weighted model use class weighting and resampled model resample the training dataset.

to 40. We train four models based on the EfficientNetB4 network, with batch size of 16, augmentation multiplier set to 0.7 and learning rate at 0.001. We then measure the accuracy and loss of the model during training of four models. Only difference is input image resolution which starts at 32 × 32 pixels, then 64 × 64, 128 × 128 and finally 256 × 256, see Figure 4.12 for an example of the different input image resolutions. We find that the model which performs overall better than the rest is the one trained on images with the highest resolution of 256 × 256. 128 × 128 and 64 × 64 both performed just slightly below the initial model, and finally the model trained on 32 × 32 pixels were not able to converge during the 40 epochs. In Figure 4.11 you'll see the model accuracy and loss development during training for the four models.

From this data, we chose to use 128 × 128 pixel images because model performance is almost on pair with 256 × 256 pixel images while reducing the training time to a fraction of the time compared to 256 × 256 pixels as well as to reduce the video memory consumption. This will allows us to train bigger models and run more experiments due to faster training.

4.2.4 Neural network dimensions

Depending on the dataset it is important to select a model of proper specifications for the classification problem at hand. In this section, we want to test how different network dimensions effect the models ability to correctly predict labels from the test images. Due to time and computation constraints we run the experiments on down-scaled images with pixel dimensions of 128 × 128. We used this resolution due to preliminary findings explained in Section 4.2.3.

In this experiment, we do not use iterative training like we do later in Section 4.3.

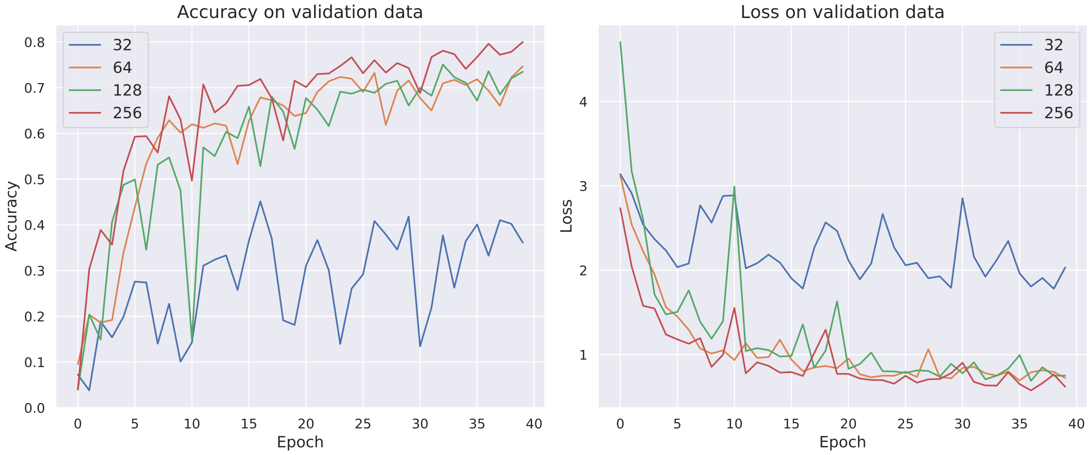


Figure 4.11: Model accuracy and loss measured on the validation dataset during training. Blue: 32 × 32, Orange: 64 × 64, Green: 128 × 128 and Red: 256 × 256.

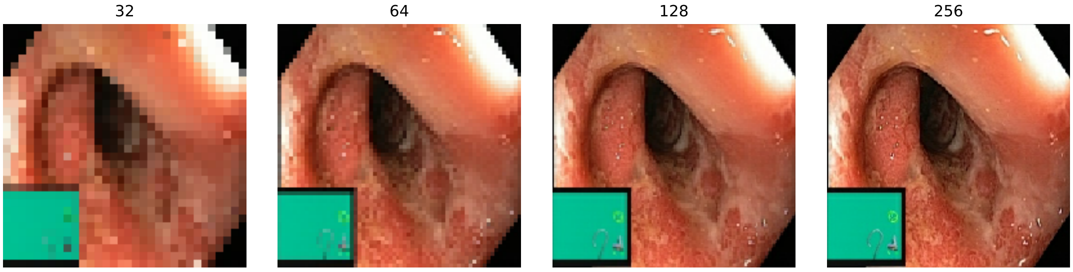


Figure 4.12: Each image starting from right contain four times as many pixels then the one to the left. For each reduction of image resolution, less feature information is available. Image taken from 'ulcerative colitis grade 3' class of Hyper-Kvasir dataset.

We set up a script which creates and train a EfficientNet model of types B0, B2, B4 and finally B6. For this study we randomly initialize the weights for each model, instead of using the pretrained ImageNet weights. The pretrained weights could potentially introduce model bias, where one models weights are favored over another model and therefore perform better. This means the networks will train a bit slower and perhaps not perform as good as with the pretrained weights. For this experiment that is fine because we want to look at the deviations in the data, which will become clearer as the model train slower. In Figure 4.13 we have trained the models for 40 epochs each on Hyper-Kvasir dataset, and plotted the accuracy and loss for training (4.13a) and validation (4.13b) data during training.

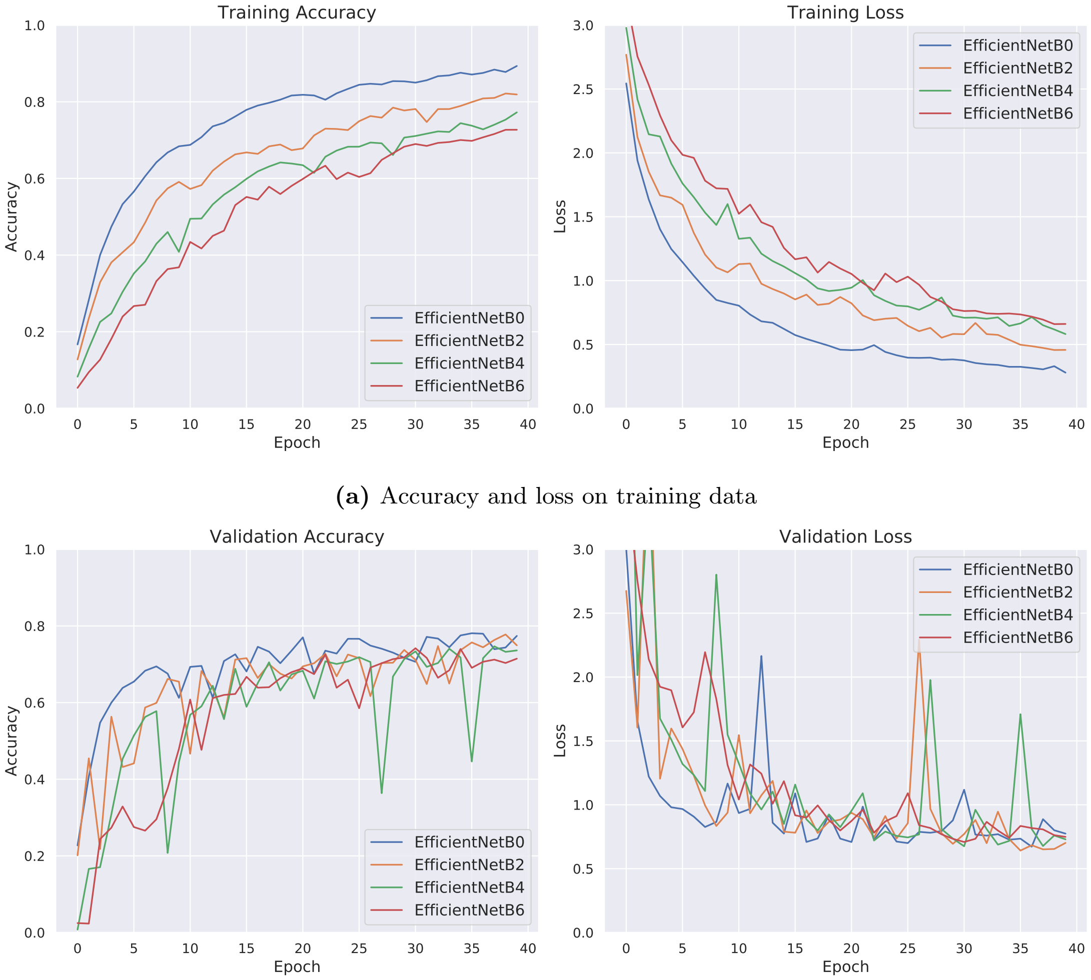
As we expected, the smaller models converge faster and achieve higher training accuracy because the smaller networks have fewer parameters to update. Based on these observations, it seems likely that smaller models perform better than larger ones, but as we can see in Figure 4.13b, this is not the case. We see that while terminal training accuracy increase by 16.5% when going from the most complex to the least complex, the accuracy only vary by a couple of percentages on the validation data, which makes it non-significant. This is because the more complex models tend to generalize better and therefore perform better on unseen data than the smaller model. For the case of the smaller models they start to overfit the training data due to less model noise, such as stochastic depth and dropout.

We have studied the learning speed and accuracy for different network dimensions of the EfficientNet family and found that networks with small compound coefficients, like EfficientNetB0 and B2, converge faster on the training data used to update its weights, but are also less capable of generalizing the data and therefore cause more overfitting. Increasing the networks compound coefficient, like in EfficientNetB4 and B6, makes the model slower to converge on the training data due to its greater amount of parameters and stochastic noise. In turn, the larger networks will generalize better and perform better on unseen data.

4.2.5 Models ability to learn class features

A main focus for developing a medical detection assistance system is good reliability. To have good reliability means that our system will perform good in a wide variety of situations, in which physicians (the end-user) and the patient can trust the system.

A underlying requirement for this is that the predictive model have learned the feature space of all its detectable classes. Upon training a model on labeled data, or a combination of labeled data and generated pseudo labels, we compute the F_1-score for each of the classes. See Table 4.1 for one of the generated classification reports accompanied by its computed F_1-scores. Preferably we would see high F_1-scores for each class, which would imply that the model correctly predicts most of the images in the test dataset correctly. In reality this is difficult to achieve as no model learns class features uniformly. Imbalanced datasets complicate the issue further as the test dataset can deviate from the represented samples within the training data, and number of samples per class, also known as the 'support', can range from single to triple digit.



(a) Accuracy and loss on training data

(b) Accuracy and loss on validation data

Figure 4.13: Four EfficientNet models with different network dimensions (compound coefficient). In Figure (a) we compute accuracy and loss on the training dataset, and in (b) we compute accuracy and loss on validation dataset. Blue line is EfficientNetB0, orange is EfficientNetB2, green is EfficientNetB6 and red is EfficientNetB6.

As a worst case scenario, our test dataset represent the class *hemorrhoids* by a single sample. If the model correctly classifies the lone image depicting hemorrhoid, the class attains a perfect F_1-score, correspondingly the class attains a F_1-score of zero if the image is miss-classified. Because of this all class scores are weighted based on the test dataset support.

We use a combination of the metrics from the classification report with visual inspection of the inferred pseudo labels to estimate the models learning aptness. Visual inspection is done by examining the generated pseudo labels from the unlabeled dataset to look for irregularities which the training data does not represent. From an extensive search through different network dimensions and hyperparameters, we find that particular classes are harder for the model to learn than others. Among the classes which the model repeatedly perform less well than expected is *barretts*, *hemorrhoids*, *ulcerative-colitis*, *esophagitis* and *impacted-stool.* The classes which repeatedly gives good results on the test dataset and generates promising pseudo labels is *retroflex stomach*, *retroflex rectum*, *dyed-resection-margins*, *normal-z-line*, *bbps*, *cecum*, *pylorus*, and finally *dyed-lifted-polyps* and *normal polyps.*

What separates the well performing classes from the below average is a blend of lower number of class samples and more diffuse and harder to learn features. As shown in previous experiments in this section, we have utilized methods to counteract the performance hit brought on by such a sparse and skewed dataset, but they do not solve all problems.

Class distribution in unlabeled dataset

A benefit of using a publicly available dataset is that we have the option to compare our model results with other research. The authors of Hyper-Kvasir paper, Borgli, Thambawita, Smedsrud, *et al.*, presented the results of how pre-trained networks predicted the unlabeled images. They predicted labels for the unlabeled images using Pre-Trained DenseNet-161 and Averaged ResNet-152 + DenseNet-161 to form an idea of the data distribution of the unlabeled data. The authors distribution of predicted images are found in Figure 4.14, here we have average the number of samples from two models trained on different data splits, both from Averaged ResNet-152 + DenseNet-162. They found a large portion of the predictions were assigned to normal pylorus, while a smaller number of predictions were assigned to the classes hemorrhoids and ulcerative colitis grade 1-2. This is similar to our results in Section 4.2.5.

In Figure 4.15, we present a bar chart combining the distribution found by the authors of the Hyper-Kvasir dataset with our own findings from a model trained on EfficientNetB0 architecture on labeled images and then generated pseudo images from the unlabeled dataset. The distribution of class-samples fit quite well, which in turn mean that the two models have learned many of the same image features during training.

The predictions on the unlabeled images are very similar to the class-level accuracy of the model trained on labeled images. We can therefore assume that the classes which achieved a high number of correct predictions on the labeled images will perform similarly on the unlabeled images.

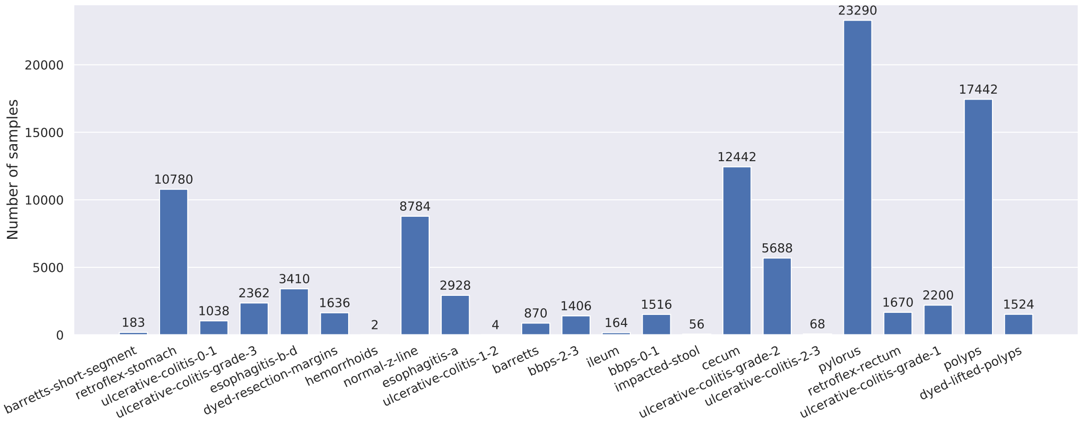


Figure 4.14: Data prediction for Averaged ResNet-152 + DenseNet-161 on unlabeled images from Hyper-Kvasir dataset made by the authors. The authors trained the model on two data-splits, here we have averaged the samples for each split.

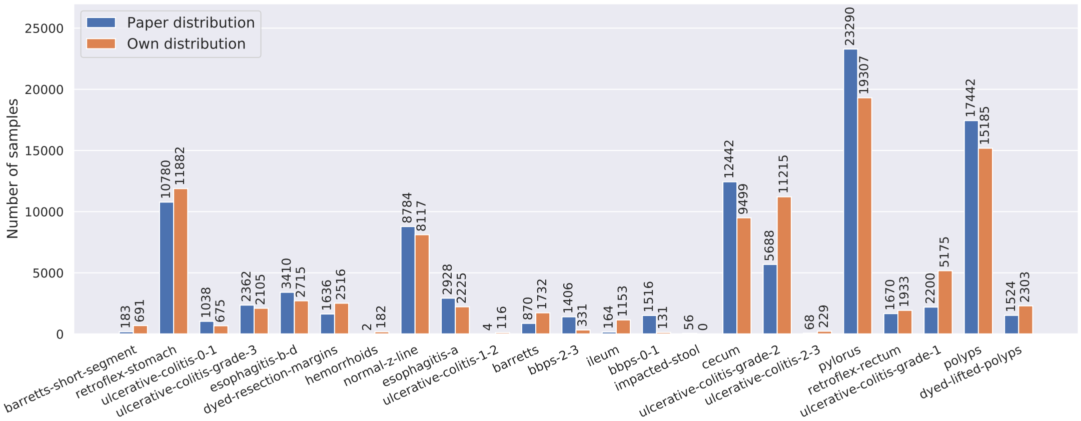


Figure 4.15: Inferred pseudo labels of Hyper-Kvasir unlabeled dataset from the authors of Hyper-Kvasir and model used in our system. Blue: Authors model trained on Averaged ResNet-152 + DenseNet-161. Orange: EfficientNetB0, architecture used in our system.

4.3 Teacher-student model on Hyper-Kvasir

In this section, we detail how we tested different parameters for our system. We first run experiments with the well tested, open-source Hyper-Kvasir [6] dataset, and in the following section we will experiment with the new Kvasir-Capsule [5] VCE dataset.

In these experiments performed on Hyper-Kvasir, we chose to use early stopping to make sure we got a optimal teacher and student model after their respective training session. However this was not used in later experiments with Kvasir-Capsule because it became harder to compare results when the model training terminated at inconsistent times. Early stopping is used during training of both teacher and student models to ensure low loss on validation data at the termination of training. We use a high early stopping patience of 10 epochs. To minimize how much the model overfits the data we introduce a low amount of noise in the form of data augmentation and dropout for the teacher, and a large amount of noise for the student. By using early stopping the final model weights are selected from the last 10 epochs with the lowest validation loss.

4.3.1 Evaluation metrics

To understand how the model behaves after being injected with inferred pseudo labels, we continuously check in on F_1-score during the experiments. Why we focus on F_1-score is explained in Section 4.1.2. We tested with some different methods for visualize model performance. Initially we measured F_1-score for each class of the dataset and studied how the class-metrics developed. An example of this is seen in Figure 4.16. This was a bit chaotic, as we get a lot of data points to compare, and the individual classes do not express the overall performance of the system.

For the rest of our experiments, we instead average out the metrics for all the classes and weight them against the support. In Figure 4.17, you see a teacher-student model trained on EfficientNetB0 for 6 iterations. With this method it is easier to notice the performance progression at the end of each iteration.

4.3.2 Model complexity: iterative training

In this section, we want to test how different model dimensions effect how well our system performs. To do this, we opted to run four experiments. We use EfficientNet with different model dimensions, all other hyper-parameters are the same for all four experiments. We select four of the eight available versions of EfficientNet due to time constraints. Each experiment takes about 10 hours to complete. Because we select every other version of EfficientNet we get about a doubling of model parameters each iteration. For every experiment, we run three iterations of swapping the student with the teacher, and we measure the performance of both the teacher model and the student model for a total of 6 data points per experiment. In this experiment, the teacher model and student model are equal. The model hyperparameters are batch size 16, learning rate 0,001, image resolution 128 × 128, 30 epochs with early stopping, augmentation

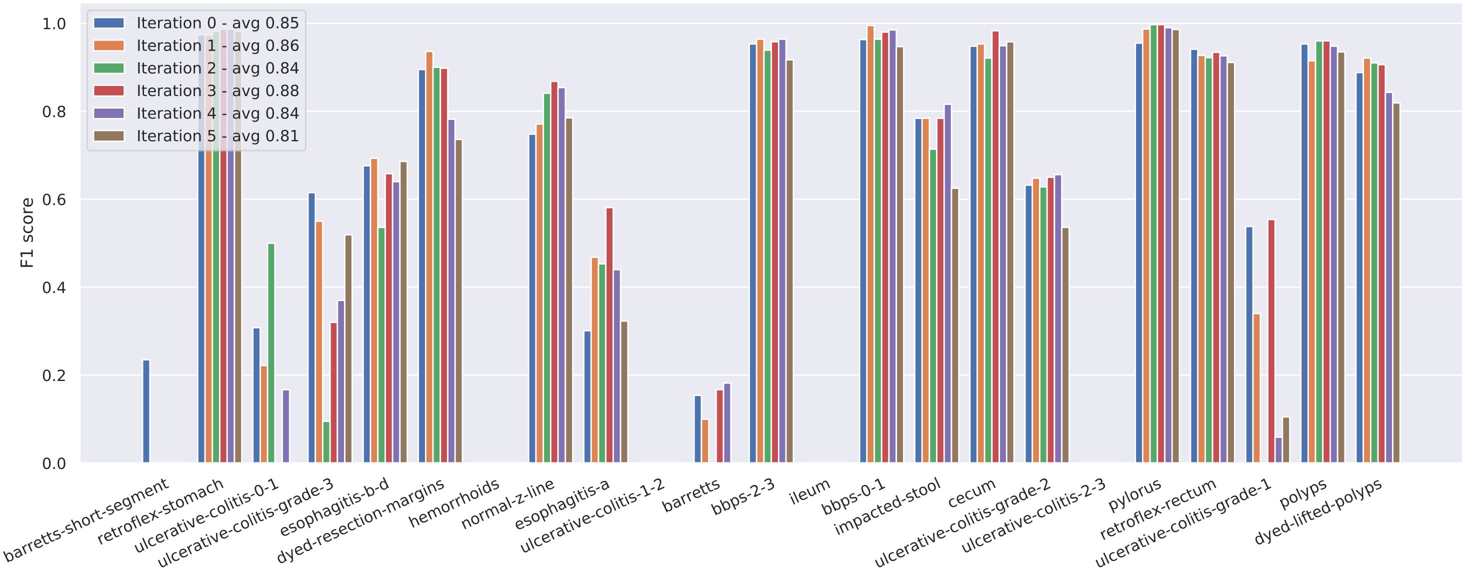


Figure 4.16: F_1-score measured per class in teacher-student model after 6 iterations. In the legend is the weighted average F_1-score for each iteration.

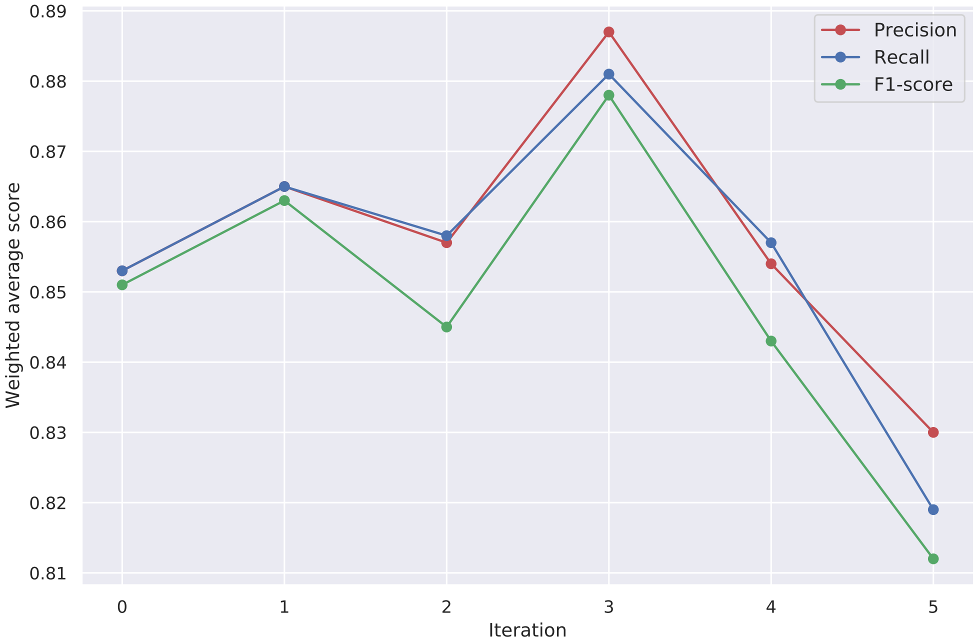


Figure 4.17: Precision, recall, and F_1-score for teacher-student model trained with equal teacher and student. Network used is EfficientNetB0. The datapoints are a weighted average of every class in dataset.

multiplier 0.5, weights initialized with ImageNet and Pseudo label threshold 0.90. The four experiments run are with the following networks;

- EfficientNetB0 with 4.7 million parameters total,
- EfficientNetB2 with 8.5 million parameters total,
- EfficientNetB4 with 18.6 million parameters total and,
- EfficientNetB6 with 42.2 million parameters total.

For each of the four runs, we collect the classification report generated by measuring the model on the test dataset (See Table 4.1 for an example). From this report we can read the change in model accuracy, precision, recall and F_1-score. The F_1-score is a good measure for how our model handles class imbalance. If the system improves from injecting generated pseudo labels into the training data we should see an increase in F_1-score. Because of feature drifting (mentioned in Section 3.4.3) we anticipate the F_1-score will drop given enough iterations of swapping the teacher and student models.

In Figure 4.18, are the collected data points for each experiment. Had the different network dimensions effected the teacher/student system we would expect to see a clear separation of the data points. However, we see that the data points are close together and in no particular order with respect to network width, depth and resolution. Another observation is that in general the student, albeit being equal to the teacher, seem to perform better on average. EfficientNetB2 and EfficientNetB4 does not reach a point where the inferred pseudo labels have clouded the models ability to perform well on the test data, while EfficientNetB0 and EfficientNetB6 see a noticeable reduction in F_1-score on the last couple iterations (down 8.0%).

4.3.3 Noising the student

As discovered by Xie, Luong, Hovy, *et al.*, it is better to use noise when training the student model [64]. In the following experiment, we compare how an equal teacher and student perform against a noisy-student. We use data from a previous experiment (Section 4.3.2) for teacher-student models trained without noise added to the student. In this data, both the teacher models and student models are equal.

We inject noise into the student during training by adding part model noise, and part input noise. The model noise is from using the larger EfficientNetB6 with more 30% dropout on the final dense layers, and the input noise is by augmenting the input image by 50% (as described in Section 3.2.6) during training. For the teacher, we used the smaller EfficientNetB0 with 10% dropout and 10% image augmentation.

From Figure 4.19, we see that the experiment with a noised student performs better than the averaged results for similar teachers and students models with different network dimensions. From the initial teacher model to the best performing model at the third iteration we have increased the weighted F_1-score by 3.2% by doing iterative training with a noised student. In Table 4.6 are the results fro accuracy, weighted F_1-score and macro F_1-score for each iteration of the noisy student experiment.

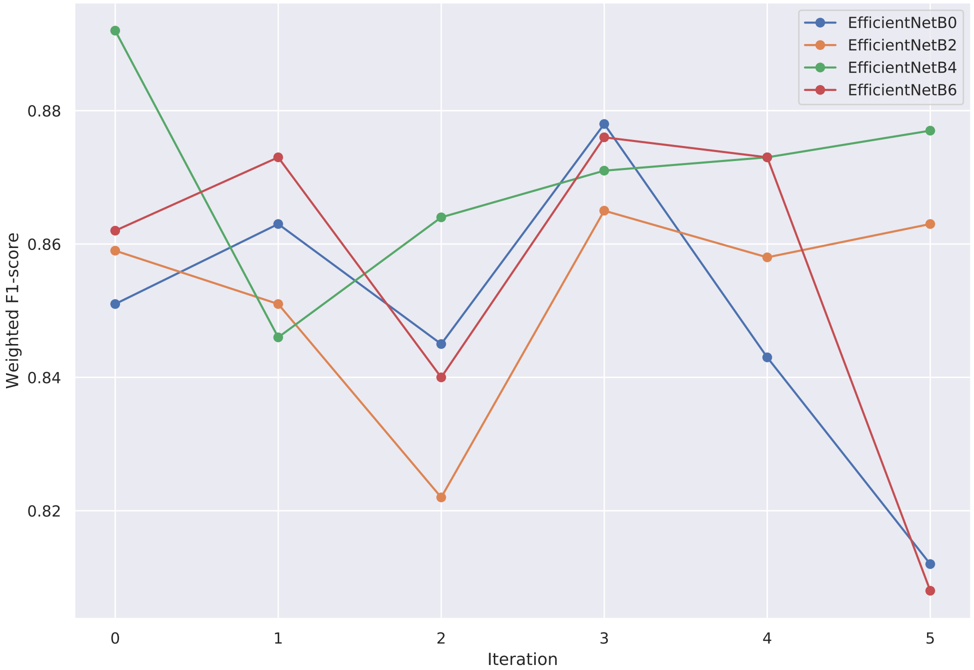


Figure 4.18: Average weighted/micro F_1-Score per class, for different model dimensions of EfficientNetB0, EfficientNetB2, EfficientNetB4 and finally, EfficientNetB6.

Iteration	**Accuracy**	**Weighted F_1**	**Macro F_1**
0	0.855	0.854	0.567
1	0.850	0.846	0.567
2	0.865	0.867	0.592
3	0.871	0.860	0.549
4	**0.893**	**0.886**	**0.605**
5	0.840	0.840	0.556

Table 4.6: Accuracy, weighted F_1-score and macro F_1-score produced by iterative training with a noised student model on Hyper-Kvasir dataset. See Figure 4.19 for a visualization of weighted F_1-score metric.

4.4 Teacher-student model on Kvasir-Capsule

In this section, we take a look at the new Kvasir-Capsule dataset, and test if a semi-supervised teacher-student model is able to take advantage of highly skewed and sparse labeled and unlabeled data collected from the VCE videos. First, do this we will have to introduce a new method of splitting the data because we are dealing with images exported from video and not independent endoscopy images like in Hyper-Kvasir. Due

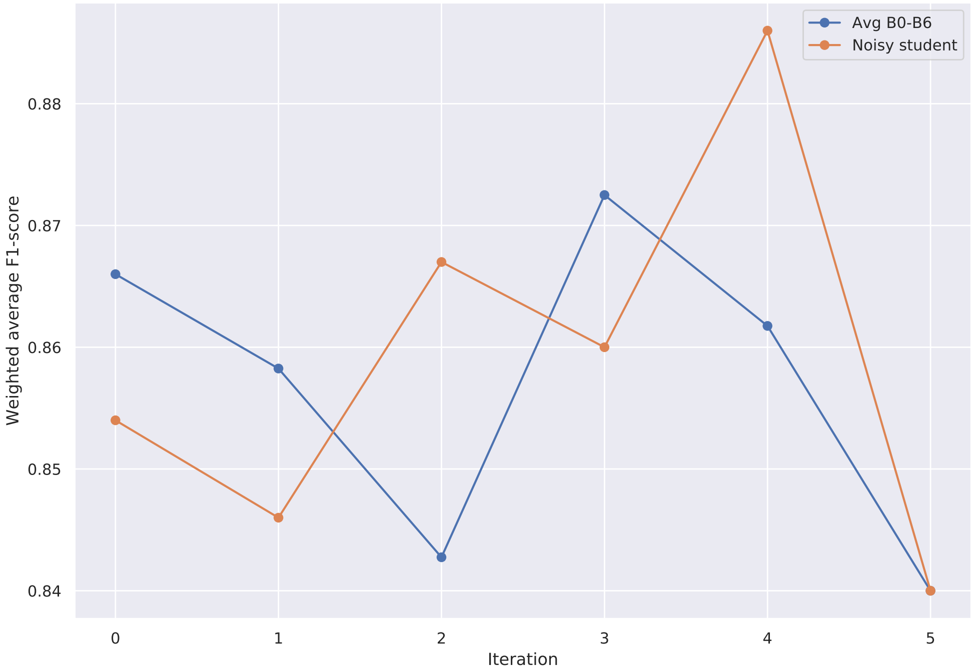


Figure 4.19: Comparing how the model perform when noise is injected into the student. The blue line represent an average of an earlier experiment run with same teacher and student models with EfficientNetB0, B2, B4 and B6. Their respective results are found in Figure 4.18. The other line show the results after 3 iterations of our teacher-student framework with noise added to the student.

to the change in data split, we also need a new evaluation method to fully understand how the framework perform. We will present and discuss our experiments, and in the next chapter we will answer our main research question.

4.4.1 Importance of good dataset split

Hyper-Kvasir dataset, unlike Kvasir-Capsule, is a collection of colonoscopy images taken from a wide range of examinations and therefore the images within this dataset have little connection other than being annotated into its 23 classes. Kvasir-Capsule however, contain images which are annotated in video form and exported to single frame images. And because the VCE device will sometimes travel quickly, and sometimes slowly, many images will have little variation in them. As an example the output from the tagging tool can have a large number of frames, but of those frames only a few represents unique cases of findings or anatomical landmarks.

In our first experiments with the Kvasir-Capsule we split the dataset into train, test and validation data the same way we did with Hyper-Kvasir - which was to pick 60% of samples from every class and use for training data, then 15% for validation data and

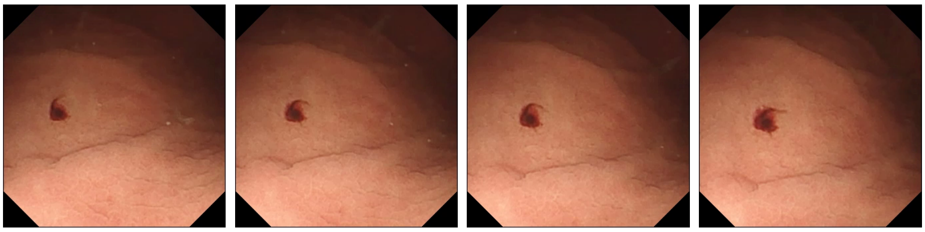


Figure 4.20: Trailing frames taken from Angiectasia class of Kvasir-Capsule dataset. In these frames the video capsule is barely moving and therefore produces very similar images of the same finding.

the remaining samples were used for testing. We then trained a test model and quickly discovered that the model (EfficientNetB0) performed much better than expected. In Figure 4.21 are the accuracy and loss for both the training and the test split. Here we see that after only a few epochs of training the model reaches 90% accuracy on the unseen test data.

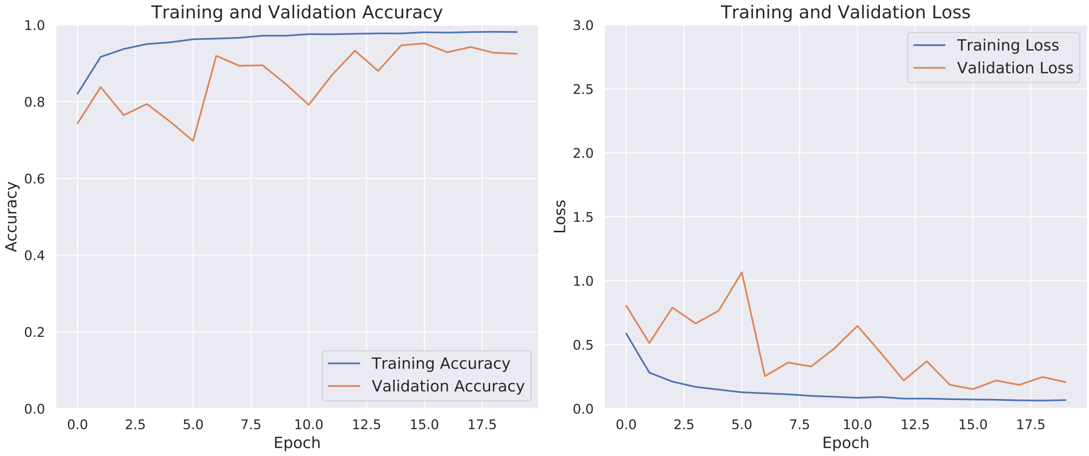


Figure 4.21: Accuracy and loss from training and validation dataset after 18 epoch of training on Kvasir-Capsule dataset. In this figure, the images are split at random and because of this the model sees the validation and test data.

The reason this model over-perform is due to the before-mentioned data splitting. Many samples contain near identical information as they are gathered from the same finding in the same video and because the dataset is split at random for every class the near identical images are represented in all 3 splits of the data, which means that in a sense the model sees the test data while training. In Figure 4.20 we see one such example. In this figure, there are four samples taken from the *angiectasia* class were the VCE device is sitting idle and therefore produces very similar images.

To fix this issue we split the data again, this time we made sure samples from any given finding were only included in one split. By doing this another issue arose; some classes have few unique findings. This means if a class contain 100 images, but all are from the same finding and therefore can't be split, it is not much better than having a class with a single image. We therefore had to drop *hematin* and *polyp* from the dataset before running our experiments. In Table 4.7 we have listed the two splits with the number of samples for each class, with the *hematin* and *polyp* classes which were excluded. In Figure 4.22 are the initial results when training on the data using this new method of splitting. In this figure, the model do not perform nearly as good on the test data as the previous run, even when trained using a larger network (EfficientNetB6), but this is expected because of the change in data splitting method. We see that the accuracy and loss for training data is quite similar for the different data split methods and this is likely due to the model 'seeing' the same data despite the change in the training dataset. Because of scarce sample size we also dropped the testing dataset split and use only two equal size splits, one for training and another for validation. Then we swap the splits used for training and validation, train the model again so it learns from all samples and average the results from the two models.

Class	**split_0**	**split_1**
Lymphoid Hyperplasia	224	368
Foreign Bodies	753	23
Ulcer	709	145
Erosion	207	232
Pylorus	592	937
Ileo-cecal valve	2,343	1,846
Blood	22	424
Angiectasia	795	71
Hematin	12	0
Erythematous	127	132
Polyp	55	10
Reduced Mucosal View	915	1,484
Normal mucosa	16,618	17,988

Table 4.7: Two fold dataset split of Kvasir-Capsule dataset. *Hematin* and *polyp* were not used in our training because of few samples in those classes.

As a last step, we ran an iterative teacher-student experiment two times on the data and monitored the accuracy and loss computed on the verification dataset. The experiment is run two times because we need to separate for which split we use for training and which is used for validation. In the first run we train on split_0 and validate on split_1, and in the second run we switch the data and train on split_0 and validate on split_1. The results are seen in Figure 4.23. In this figure, we can see that after training a total of 12 models, 6 on each each split and using the remaining split as validation, the models perform better when being trained on split_1. The accuracy is higher and the loss is lower for this split. This is not by any means a significant finding, but is the reason to why it is important to train on both splits and then average the results.

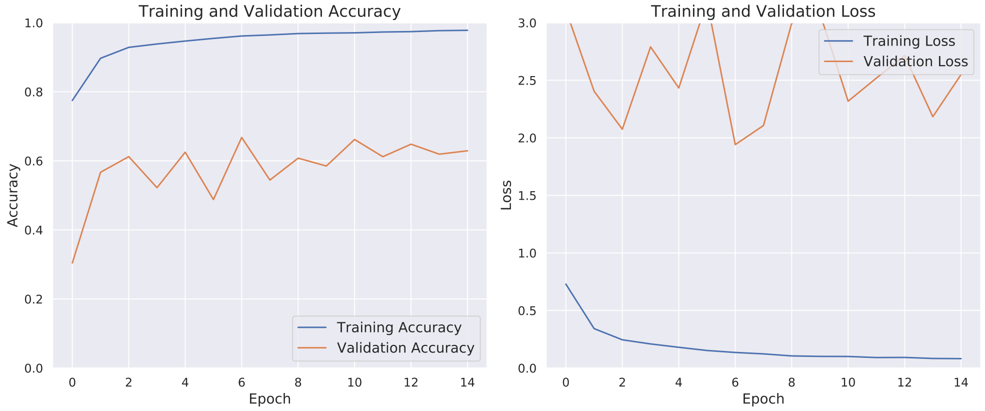


Figure 4.22: Accuracy and loss from training and validation dataset after 15 epoch of training on Kvasir-Capsule dataset. In this figure, the data have been carefully split to not include samples from same finding in more than one split.

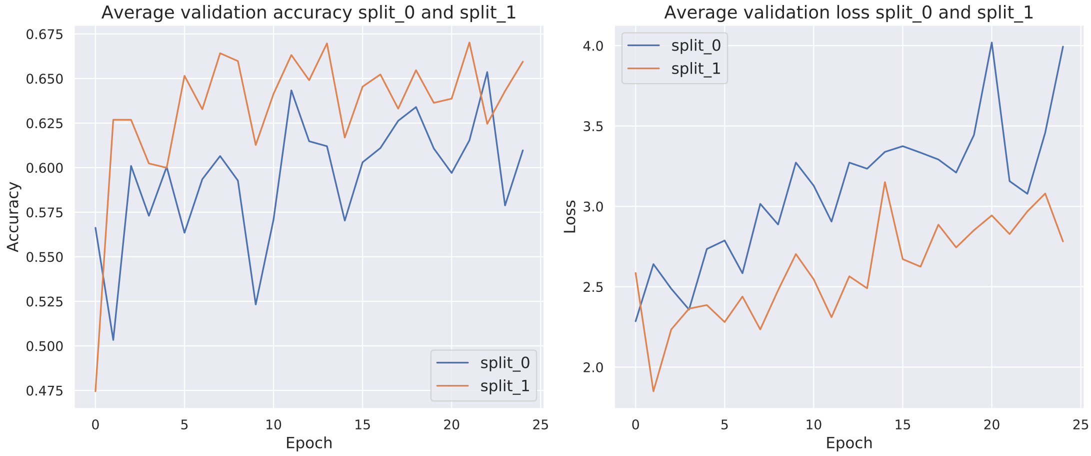


Figure 4.23: Accuracy and loss computed on the split_0 and split_1 during training of 3 iterations of teacher and student models for 25 epochs. The results are then averaged for the models trained on each respective split. Split_0 represents the average results of models trained on split_0 and validated on split_1 and vice versa.

4.4.2 Unlabeled dataset

In Kvasir-Capsule, there is a total of 2.6 million unlabeled images. Having a large amount of unlabeled data is necessary for better performance when it comes to semi-supervised iterative training [64], however with limited time and resources we have opted to use a subset of the unlabeled dataset. The subset was selected from the 2.6 million images by picking 200,000 images in the input pipeline. We pick images at random to make sure we create a new corpus which represents the available data in the 74 videos not used for annotation. To make our findings reproducible we seed the random number generator, which makes the input pipeline produce the same subset of unlabeled images for both splits of the dataset. More details of the unlabeled data are found in Section 3.1.2 where we have discussed how the data were collected.

By doing this we, effectively double the amount of unlabeled data in comparison with Hyper-Kvasir dataset, at the expense of higher memory usage and time consumption during training. With the smaller teacher model it is estimated to take 28 hours to fully iterate the unlabeled dataset on our system, with 24-26 images per second, and the larger student model would take approximately 42 hours to run through all 2.6 million images, with 16-18 images per second. By limiting ourselves to use 200,000 unlabeled images it takes 2 hours for the teacher model and 3 hours for the student model.

4.4.3 Evaluate Kvasir-Capsule results

Due to the new splits introduced with Kvasir-Capsule dataset, we essentially run the full experiments on both splits, using one split for training and the other for validation and vice versa. This has both advantages and disadvantages over using a three way split of training, validation and testing data. Our model now learn from all of the data, and no information is hidden within the test data. However, we no longer leave a subset of the data out of the loop to be used for testing how well the model perform on unseen data, making system verification more difficult.

To evaluate our system trained on two data splits we duplicate the split used for model verification and feed it to the evaluation pipeline (See Figure 3.3 for visualization of the training pipeline) and compare the classification report produced by the evaluation script and the history of model metrics generated by TensorFlow during training. We will then present the results for both splits and the average of the two.

System verification is a vital stepping stone to implement machine learning in the medical sector, but due to time limitations we have deemed this out scope for this project and instead we leave the reader with some comments on how we would go about further testing our system for the Kvasir-Capsule dataset. We propose to use cross-dataset validation to increase confidence in the system. Other VCE datasets like GIANA [27] includes findings of angiectasias from the small intestine and could be used for testing of the systems ability to correctly detect that specific finding. Because of the lack of other available VCE datasets we also propose to use colonoscopy datasets like Kvasir-v2 [17] and Hyper-Kvasir [6] for verification of pylorus detection.

4.4.4 Noisy student experiments

An issue we encountered when running our teacher-student framework on the Kvasir-Capsule dataset was that due to the larger number of samples in the dataset we could no longer run our system on the largest EfficientNetB7 model. From earlier experiments in Section 4.2.4, we have found that model performance do not vary much in the largest networks and we chose to train the student model with EfficientNetB4. We then ran our program on both splits of the Kvasir-Capsule dataset and created an average of the results. Next we will discuss two experiments we conducted on the Kvasir-Capsule dataset.

Growing classes to 500 samples

In our first experiment, we chose to set the parameter "class_limit" to 500. This choice was made because we have experienced from earlier that adding too many pseudo labels to the training data increase the memory usage of the system. The image resolution were set to 335 by 336 pixels and model training are set to 25 epochs for both teacher and student model. We ran the experiments for three iterations. Because we wanted to analyze the model history of the accuracy and loss during training we disabled early stopping, and ran the teacher and student models for 25 epochs each.

In Figure 4.24, we have included the history of training and loss validation generated by all the models during training. We notice that the training accuracy varies between 50% and 70%. This is lower than the results which were obtained from running experiments on Hyper-Kvasir dataset. We suspect the reason for this is that Kvasir-Capsule dataset has even higher class imbalance than Hyper-Kvasir. Another factor is that although having higher number of samples, the quality per sample is lower due to the fact that the dataset is exported from video rather than various examination images like Hyper-Kvasir. This is discussed in more detail in Section 4.4.1. We also see in Figure 4.24 that the loss calculated on the validation data is increasing on average for all models and the models are overfitting the training data. In an effort to eliminate the increasing loss, we introduce early stopping again, and add more pseudo labels in later experiments.

We monitor F_1-scores and accuracy for all models during execution of our program, and if these values are averaged after training on both splits of the data we can create a plot of the system development during the iterations of switching out the teacher with a student and see the effect of injecting pseudo labels into the training data. Such a plot is seen in Figure 4.25. In this figure, we can see that the performance gains are poor. The first teacher model achieve a rather impressive accuracy and weighted F_1-score of around 70%, but in the following iterations the combined performance of the system is declining. The second student model improve upon the second teacher model, but still worse than the initial datapoint taken before any pseudo labels are introduced to the training data.

Growing classes to 1500 samples

In this experiment, we configured the model to terminate training after 10 epochs if the validation loss has not improved. At the termination of training the model then load the weights from the epoch with the lowest computed loss. This help reduce overfitting and increase the models ability to generalize on the data. Another setting we changed in this experiment is the variable which determine the threshold for maximum pseudo labels to combine with each class. In this experiment, we raise the value from 500 maximum total

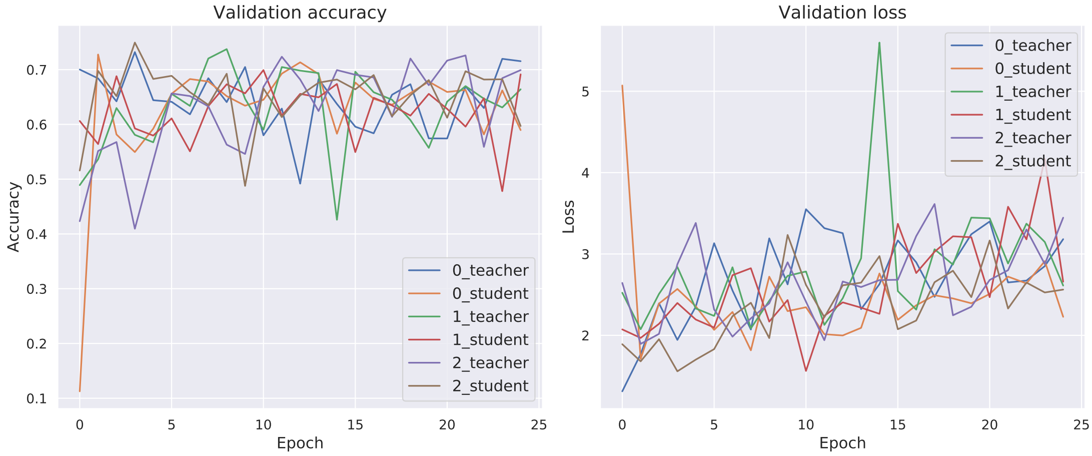


Figure 4.24: Accuracy and loss computed on the validation data during training of 3 iterations of teacher and student models for 25 epochs.

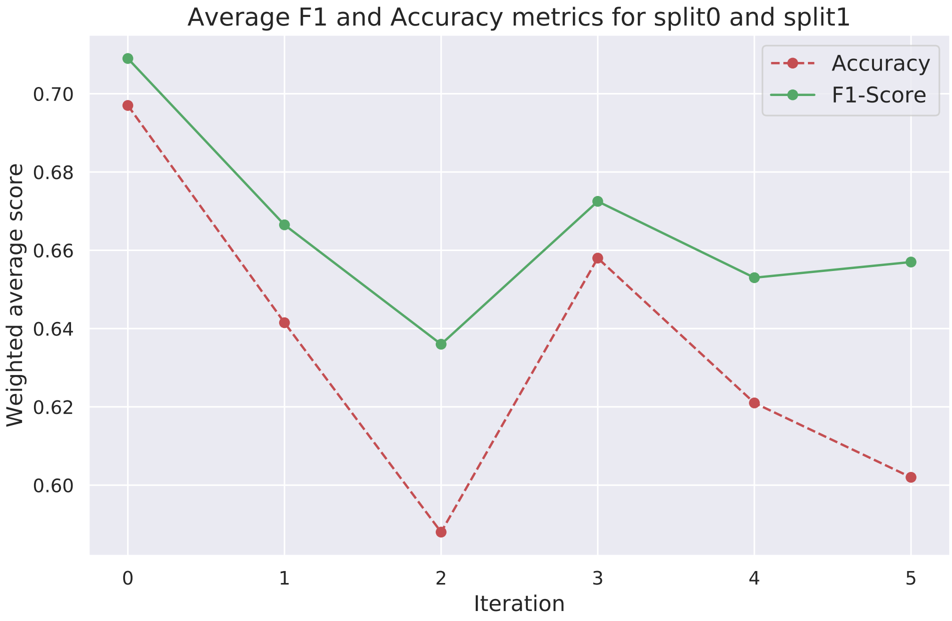


Figure 4.25: Averaged accuracy and F_1-score for both splits after 3 iterations of switching out the teacher with the student.

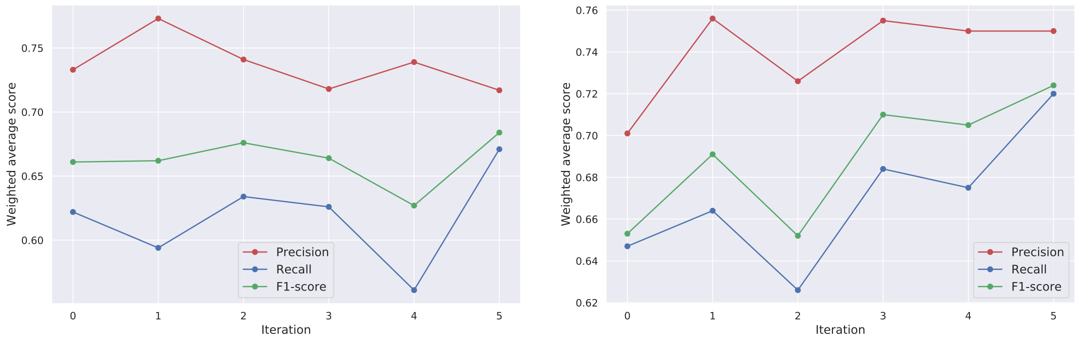
samples per class to 1500 samples. This mean that while the program is running the classes will continue to grow as the models find more pseudo labels from the unlabeled images, until the combined amount of samples in each class reach 1500. At that point

Iteration	Accuracy	Weighted $\mathbf{F_1}$
0	**0.697**	**0.709**
1	0.641	0.666
2	0.588	0.636
3	0.658	0.673
4	0.621	0.653
5	0.602	0.657

Table 4.8: Accuracy and weighted F_1-score produced by iterative training with a noised student model on Kvasir-Capsule dataset. See Figure 4.25 for a visualization of the metrics.

no more pseudo labels are added.

In Figure 4.26, we see an visualization of Recall, Precision and F_1-score averaged for three iterations of our teacher-student system. We can see for both splits, the precision metric are always above the recall metric. In the case were the models are trained on split_1 (Figure 4.26b), the F_1-score increase with the student models, and mainly because the rise in recall. The precision metric is less effected by the injection of pseudo labels. This also strengthen our theory introduced in Section 4.4.1 that the system perform better when trained on split_1 and validated on split_0. This is perhaps not a significant discovery but further confirm our suspicion of an uneven dataset split.



(a) Trained on split_0 and validated on split_1

(b) Trained on split_1 and validated on split_0

Figure 4.26: Weighted average score for recall, precision and F_1-score for teacher-student models run for 3 iterations on both splits of Kvasir-Capsule dataset.

In the Figure 4.27, created from the same experiment, we averaged the accuracy and F_1-score for both splits. In Table 4.9, we listed the corresponding metric values. Here we see that the model accuracy and average weighted F_1-score increase as pseudo labels are injected into the training data. When the experiment terminated, the weighted F_1-score had increased by 4.7%. We observe that the larger student models appear to perform better than the smaller teacher models on average. And at the last student model in the last iteration both averaged model accuracy and averaged weighted F_1-score are up from the initial point.

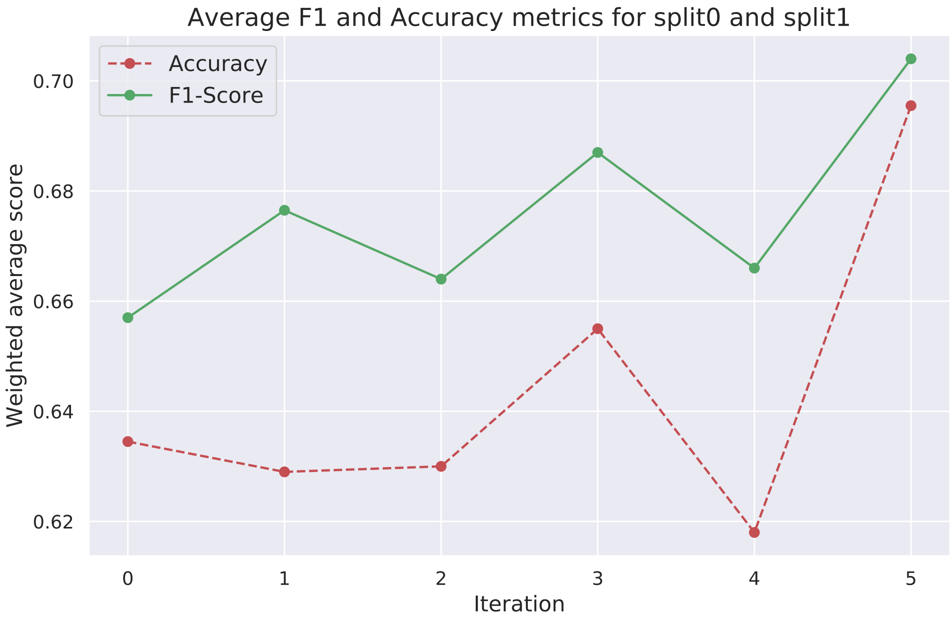


Figure 4.27: Averaged accuracy and F_1-score for both splits after 3 iterations of switching out the teacher with the student.

Iteration	Accuracy	Weighted $\mathbf{F_1}$
0	0.634	0.657
1	0.629	0.676
2	0.630	0.664
3	0.655	0.687
4	0.618	0.666
5	**0.695**	**0.704**

Table 4.9: Accuracy and weighted F_1-score produced by iterative training with a noised student model on Kvasir-Capsule dataset. See Figure 4.27 for a visualization of the metrics.

These results appear very promising but it is important to point out the shortcomings of our system as well. In Figure 4.28, we include a confusion matrix produced by the final student model which were trained on split_1. In the displayed confusion matrix we can see that there are a lot of false positives for *normal mucosa* and *pylorus* classes by the darker column for these two classes especially. These two classes in particular hold a majority of the VCE images. The pylorus class contain 1,529 images, and *normal mucosa* contain 34,606 images. When these classes are randomly sampled in the input pipeline over many epochs the models learn to generalize the classes better than the minority classes and thereby predicts labels of majority classes.

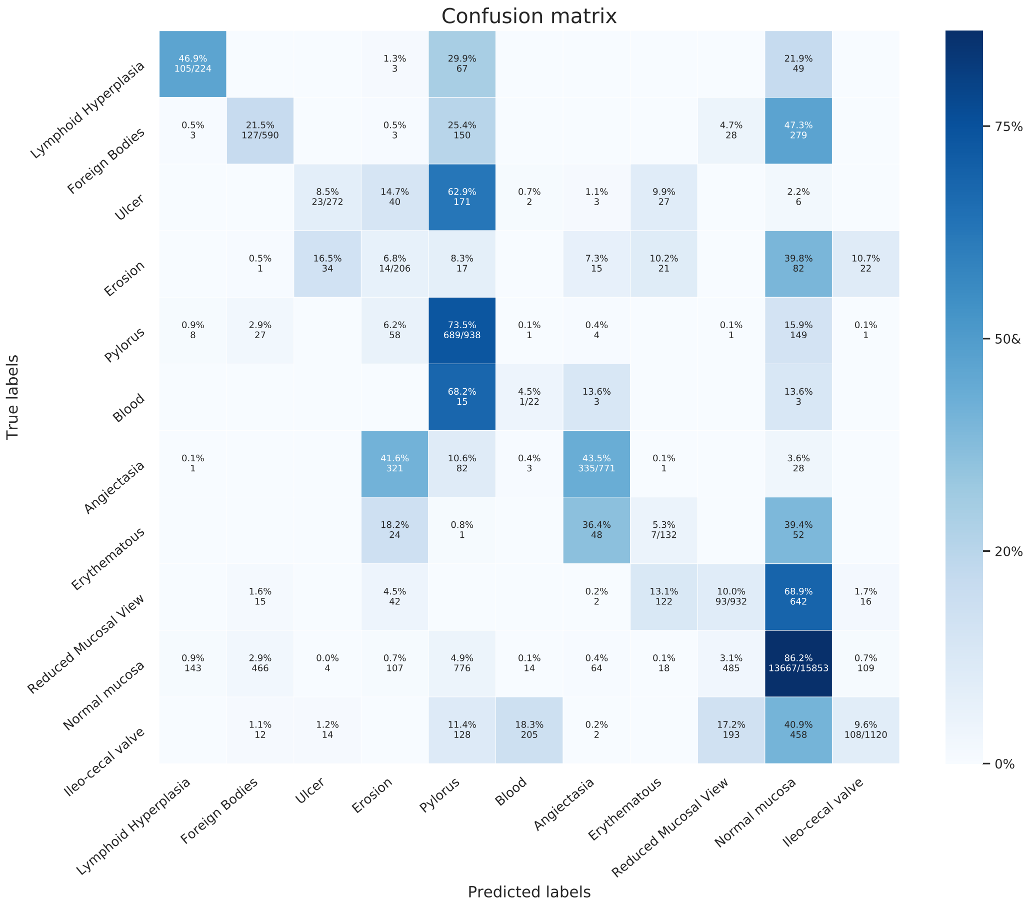


Figure 4.28: Confusion matrix created by testing the last student model on split_1 of Kvasir-Capsule dataset.

In Figure 4.29, we include a generated report of top six pseudo labels from each class in Kvasir-Capsule unlabeled dataset, generated by the initial teacher model. In this preview of inferred pseudo labels we can analyze the quality of predicted labels, which is useful to diagnose the teacher-student framework. An expert endoscopist reviewed the pseudo label report, and the results are given in in Table 4.10. Here we see the majority of the generated pseudo labels are correctly classified, but for some classes the teacher model have a hard time. This could be linked to having a sparse and skewed dataset. Of the 66 shown pseudo labels, 69.69% are correctly classified. The endoscopist who reviewed the labels noted that for *lymphoid hyperplasia* class the hyperplasia is floating in the lumen and not on the mucosa and therefore not miss-classified. The labels from *foreign bodies* are off food left-overs, so it is a question of class definition. Images from *blood* labels are deep red in color, but could potentially be from other factors. And finally, pseudo labels from *ileo-cecal valve* are difficult to verify without having access to the video from with the images are originating from.

Class	**Wrong classification**	**Correct Classification**
Lymphoid Hyperplasia	1,2,3,4,5,6	
Foreign Bodies		1,2,3,4,5,6
Ulcer		1,2,3,4,5,6
Erosion	1,4,5,6	2,3
Pylorus	4,5,6	1,2,3
Blood		1,2,3,4,5,6
Angiectasia	1,2,3,4,5	6
Erythematous		1,2,3,4,5,6
Reduced mucosal view		1,2,3,4,5,6
Normal mucosa		1,2,3,4,5,6
Ileo-cecal valve	2,3	1,4,5,6
# Total samples	20	46

Table 4.10: Validation report for Figure 4.29 made by an endoscopist expert. ”1” represent the left pseudo label from the corresponding class, ”2” represent the second pseudo label etc.

Each row holds images with the six highest model confidences, in descending order from left to right. Quality of the displayed pseudo labels vary from class to class and in some cases, image to image. The inferred pseudo labels of *ulcer* class have very similar images, likely from the same video and finding. When combined with the original training data, these six images only really count as a single image because of the overlap in image information, as discussed in Section 4.4.1. The images of the class *erythematous* are potentially also from the same video, but in these images we see more variation in the form of capsule rotation and translation between frames. These images provide better pseudo labels for the next student model due to the image feature variation.

4.5 Summary

In this chapter, we presented our experiments leading up to defining a suitable set of hyper-parameters for our teacher and student models, and then the experiments we performed on the teacher-student framework itself. For our initial experiments conducted to find a suitable network configuration we primarily focused on Hyper-Kvasir dataset because it allowed us to compare our results along the way with previous works. Then when our framework was completed we ran some initial tests with the same dataset before moving on to our VCE dataset, Kvasir-Capsule. Initial experiments conducted on Hyper-Kvasir were useful for comparing dataset balancing methods such as resample and class weighting, model weight initialization, input image resolution, and network

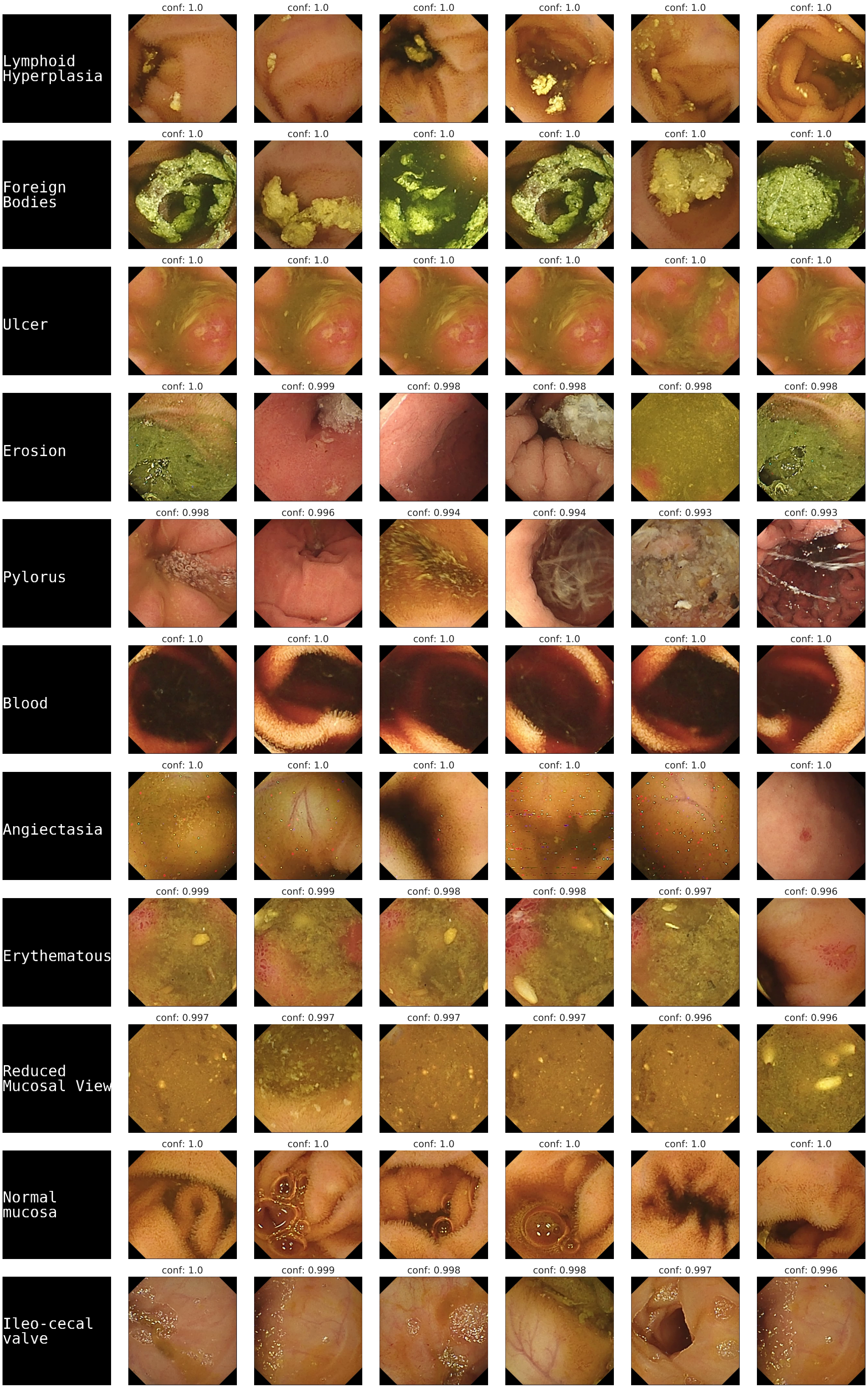


Figure 4.29: A preview of pseudo labels from Kvasir-Capsule unlabeled dataset, generated by the first teacher model. Each row is pseudo labels of the respective class, as stated by the left column. The model confidence is given above each image, rounded to four digits.

dimensions. When the framework was complete we experimented with parameters such as teacher-student model dimensions during iterative training, adding noise to the student model, evaluation methods, and the amount of pseudo labels to combine with the classes. The experiments conducted in this chapter relate back to our last research objective in Section 1.2, where we wanted to experiment with various model hyper-parameters and framework settings to get a better overview of their effects on performance, and thereby, obtain a better understanding of the self-learning process of such a framework which could result in knowledge to better improve performance.

After conducting our initial experiments, we found that using pre-trained weights greatly outperform training from scratch when it comes to both model accuracy and training time, re-sampling the dataset had enormous benefits over calculating class weights, small reduction in image resolution have little impact on model performance but allow for much larger models to be trained, and finally, small networks train faster but are worse at generalizing. Our main findings from conducting experiments on the teacher-student framework on Hyper-Kvasir dataset were that for four different network dimensions for both teacher and student model, the second student achieved the best results on average and all four model dimensions performed very similar. Adding noise in the form of input image augmentation and larger model with more dropout, increased the performance further. When testing our framework on Kvasir-Capsule, we experimented with different thresholds for combining pseudo labels with the training data. We found that adding more pseudo labels increased the performance.

Chapter 5

Conclusions and Further Work

5.1 Summary

Semi-supervised learning is of great interest in machine learning and data mining because it can use readily available unlabeled data to improve supervised learning tasks when the labeled data are scarce or expensive. This is often the case for the medical domain where image annotation require trained specialists. The total world population is increasing every year, and as a consequence, we perform more colonoscopies than ever before. The demand for better systems for medical diagnosis will follow this trend. Previous work in development of medical multimedia system show that providing real-time assistance during colonoscopy improves the chance of successful treatment, if the initial observation of disease indicators can be made visually at a early stage, preferably before patients notice any symptoms. But, for these systems to reach its full potential there is still a need for larger collection of multimedia, including the likes of VCE videos, and better high-level deep learning approaches for visual feature inspection [100].

We have during this thesis presented a teacher-student framework that take advantage of the vast amounts of unlabeled data by first training the teacher model on labeled images, use the teacher model to generate pseudo labels on unlabeled images, and train the student model on the combination of labeled images and pseudo labeled images. We iterate this algorithm a few times by treating student as a teacher to relabel the unlabeled data. By adding noise to the student we force the student to learn beyond the teacher to make prediction with more difficult images.

To be able to test and evaluate our semi-supervised teacher-student framework, we created a VCE dataset called Kvasir-Capsule [5], consisting of 118 examination videos provided by Bærum Hospital in Norway. 44 of these videos were carefully annotated into 13 classes of pathological findings, anatomical landmarks and quality of mucosal view resulting in a labeled dataset of 44,260 images. The remaining videos were exported to 2.6 million unlabeled images as part of the dataset.

Our main research question asked in Section 1.2 was as follows:

> *Can a semi-supervised teacher-student framework improve on traditional supervised models by incorporating inferred pseudo labels into the labeled training data in the field of gastrointestinal tract endoscopy?*

After conducting our experiments, we can say with good confidence that with proper tuning and enough labeled data it is very much possible. Our results showed an overall increase in F_1-score of 3.2% for the Hyper-Kvasir dataset, and 4.7% for Kvasir-Capsule VCE dataset. However, we were not able to conclusively validate performance increase on Kvasir-Capsule dataset, seeing as small changes in framework settings caused unexpected changes in the iterative training. With more testing and validation we believe our framework has potential to be a useful addition to existing medical multimedia systems for automatic disease detection.

5.2 Contributions

As discussed in the problem statement in Section 1.2, we derived three research objectives. Below, we restate each objective together with a description of how our work solves the stated problems.

Objective 1 *Collect data for a video capsule endoscopy dataset of both labeled and unlabeled images from the gastrointestinal tract, with a skewed balance of class samples to represent real world scenario. This dataset should be used for testing our framework.*

This objective is supported by our creation of a gastrointestinal dataset, Kvasir-Capsule [5], containing a total of 44,260 manually labeled images with bounding boxes around the respective finding, split into 13 classes for pathological findings, anatomical landmarks and quality of mucosal view. The annotation was performed by three MSc student, supervised by an expert endoscopist with many years in the field. Whenever the MSc student encountered an issue, the endoscopist reviewed the case. We also include the 44 videos used for extraction of labeled images, as well as 72 videos which are not labeled and thus useful for unsupervised and semi-supervised machine learning system or to generate more labeled images by other qualified personnel in the future. All videos are taken by video capsule endoscopy during a number of examinations at Bærum Hospital in Norway, between the year 2016 and 2018. The Kvasir-Capsule dataset is available from the Open Science Framework (OSF) accessible via the link `https://osf.io/dv2ag/`.

Objective 2 *Provide a implementation of a teacher-student framework for multiclass image classification based on the novel EfficientNet architecture, with a suite of evaluation tools to help with further analysis.*

This objective is supported by our proposed teacher-student framework presented in Section 3.4 which use a teacher model, based on Googles EfficientNet architecture, to create pseudo labels from unlabeled endoscopy images, which are then combined with original labeled training data. Next, we create a larger student model with more stochastic noise, and inject the input pipeline with noise transformations, like image translation, rotation and variances in brightness, saturation etc, to create a larger and more noisy student model. This noised student model learn more features from the combined dataset of labeled images and inferred pseudo labels, than the smaller teacher model. This process is then iterated a couple of times to further increase model performance.

Objective 3 *Use various model hyper-parameters and framework setting to get a better understanding of the effect which is caused by combining pseudo labels with original training images, and map performance gains by using various network dimensions.*

This last objective is supported by our research made in Chapter 4, where we present a detailed analysis and ablations of various design choices, such as architecture, hyper-parameters, class imbalance equalizing methods, image input resolution and more. When performing such experiments we measure cross-entropy loss, model accuracy, recall, precision and F_1-score during training and create easy-to-diagnose plots and reports for every model. The data is split in suitable folds and used for training and validation to ensure good validity. Based on this performed analysis, we derive a configuration of our teacher-student framework which improve on the baseline performance of our initial EfficientNet models by 3.2% for Hyper-Kvasir and 4.7% for Kvasir-Capsule.

Through the work produced in this thesis, and by answering the stated research objectives, we have learned the value of using pre-trained network weights to greatly reducing training time, importance of sampling a imbalanced dataset to help the model generalize better during training, how changes in image resolution speed up training at the cost of model performance, and the various effects of changing network dimensions. With this knowledge we then designed and developed a self-learning teacher-student framework. This semi-supervised teacher-student framework, trained on sparsely and skewed labeled video capsule endoscopy images and traditional endoscopy images, has shown the ability to improve on traditional supervised models in our conducted experiments with varying results. With more tuning of the framework settings and more data for both training and validation, this self-learning paradigm of machine learning can have profound effects on the future of computer-assisted diagnose in the medical domain.

5.3 Further work

The work done in this thesis shows that self-learning semi-supervised approaches like our teacher-student model are promising, but also that there is potential for improvements and that there is still much work that can be done. Some of the improvements which would show immediate progress is to gather larger VCE datasets, and to do experiments to gain a better understanding of how to take full advantage of sparsely labeled datasets when combined with vast amounts of unlabeled data. Below, we propose some further work:

Better image preprocessing and cleanup Kirkerød, Borgli, Thambawita, *et al.* proposed a method which use a Generative Adverserial Network (GAN) to remove artifacts like scope guide and black borders from the training data and as a result allows networks to generalize to a greater extent and achieve higher accuracy [101]. Kvasir-Capsule [5] has less artifacts than the medical data used for evaluating the preprocessing tool developed in the thesis, but nonetheless, it would be interesting to test how the removal of black borders around the VCE images would effect the performance of our system. This is something we would have tested if we had more time. We also noticed the trained models would sometimes select pseudo labels from the unlabeled dataset which contained mostly static noise, such as from periods of bad connection between the capsule and the receiving unit. This leads us to believe that some data filtering beforehand which would remove such noisy images would be beneficial for our models.

Sample from generated pseudo labels Because the unlabeled images from Kvasir-Capsule are extracted from videos there are large amounts of similar images containing large overlap in image features. This causes the teacher-student model to include images which are very similar, and can in some cases be many images from the same video and the same finding. In short, this means while 100 new pseudo labels are combined with the respective class, it actually only contain several similar images of the same finding, sometimes also from the same angle, see Figure 4.20 for an example. This could be further improved by including a pseudo label evaluation method before introducing the sample to the training data. We imagine such an evaluation method would hinder many similar images to be included and instead pick images from the next in line finding from the unlabeled dataset.

Human-in-the-loop From some of our experiments with Kvasir-Capsule dataset, we noticed that the model struggles to learn the correct features of the classes with fewest samples, and therefore often will produce out of domain pseudo labels. In the case of extreme sparse data, or extremely skewed, it could be a human-in-the-loop, a specialized physician, which would be presented some of the generated pseudo labels with the highest confidence, and manually discard the worst labels.

Better validation methods and metrics Because of the immense class imbalance which we face with medical datasets, we need to use a large amount of the dataset for training, and in some cases, only keeping a couple of samples hidden away for cross validation can harm the features learned by the model. Cross dataset validation would have a great impact on this because more of the data is used for training. Time constraints did not allow us to implement this but is something we would like to test in the future. In the case of medical domain image classification and the class imbalances which follows it is especially important to ensure the model have learned the classes well. In our studies, we rely heavily on F_1-score computed on the test dataset after training. We believe our models would benefit from computing macro F_1-score during training on the test dataset and select the epoch with best F_1-score to use for its final weights instead of lowest loss. For some of the classes in our datasets, we noticed that particular models had difficulties in learning some classes with overlap in image features, and in other classes, we suspect even though it correctly classified the images it might have focused on unimportant features. Therefore, we suggest that in further studies the system would benefit from implementing class activation maps (CAM) to verify the model is learning relevant features.

Fine tune on labeled data In Section 2.6, we presented some of the findings made by Yalniz, Jégou, Chen, *et al.* They pre-trained the student model and later fine-tuned on the initial labeled data to circumvent potential labeling errors introduced by the unlabeled data [75]. We believe that fine-tuning the student model could improve our method as we have found that in some cases where the labeled data is sparse, and the teacher model therefore struggle to learn the class features which then means the pseudo labels generated by the model is of lower quality.

Bibliography

[1] B. Bilbao-Osorio, S. Dutta, and B. Lanvin, "The Global Information Technology Report 2014", p. 369, 2014 (cit. on p. i).

[2] A. Jemal, R. Siegel, J. Xu, and E. Ward, "Cancer Statistics", *CA: A Cancer Journal for Clinicians*, vol. 60, no. 5, pp. 277–300, 2010. [Online]. Available: `https://onlinelibrary.wiley.com/doi/abs/10.3322/caac.20073` (visited on 05/23/2019) (cit. on pp. 3, 14).

[3] E. Rosenthal, "The $2.7 Trillion Medical Bill", *The New York TimesHealth*, Jun. 1, 2013. [Online]. Available: `https://www.nytimes.com/2013/06/02/health/colonoscopies-explain-why-us-leads-the-world-in-health-expenditures.html` (visited on 08/03/2020) (cit. on pp. 3, 14).

[4] D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, P. R. Young, and P. J. Denning, "Computing as a discipline", *Communications of the ACM*, vol. 32, no. 1, pp. 9–23, Jan. 1, 1989. [Online]. Available: `https://doi.org/10.1145/63238.63239` (visited on 03/16/2020) (cit. on p. 6).

[5] P. H. Smedsrud, H. Gjestang, O. O. Nedrejord, E. Næss, V. Thambawita, S. Hicks, H. Borgli, D. Jha, T. J. D. Berstad, S. L. Eskeland, M. Lux, H. Espeland, A. Petlund, D. T. D. Nguyen, E. Garcia-Ceja, D. Johansen, P. T. Schmidt, M. A. de Lange Thomas Riegler, and P. Halvorsen, "Kvasir-Capsule, a video capsule endoscopy dataset", Aug. 2020. [Online]. Available: `https://osf.io/gr7bn/` (cit. on pp. 7, 8, 21, 24, 57, 114, 131, 132, 134).

[6] H. Borgli, V. Thambawita, P. H. Smedsrud, S. Hicks, D. Jha, S. L. Eskeland, K. R. Randel, K. Pogorelov, M. Lux, D. T. D. Nguyen, D. Johansen, C. Griwodz, H. K. Stensland, E. G. Ceja, P. T. Schmidt, H. L. Hammer, M. Riegler, P. Halvorsen, and T. de Lange, "Hyper-Kvasir: A Comprehensive Multi-Class Image and Video Dataset for Gastrointestinal Endoscopy", Open Science Framework, preprint, Dec. 20, 2019. [Online]. Available: `https://osf.io/mkzcq` (visited on 02/19/2020) (cit. on pp. 7, 21, 23, 24, 112, 114, 122).

[7] S. A. Hicks, “Mimir: An Automatic Reporting and Reasoning System for Screening of the Gastrointestinal Tract Using Deep Neural Networks”, 2018. [Online]. Available: `https://www.duo.uio.no/handle/10852/65179` (visited on 08/22/2019) (cit. on p. 11).

[8] R. Jensen, “Polyp Detection using Neural Networks - Data Enhancement and Training Optimization”, 2017 (cit. on p. 11).

[9] J. Wei, A. Suriawinata, L. Vaickus, B. Ren, X. Liu, J. Wei, and S. Hassanpour, “Generative Image Translation for Data Augmentation in Colorectal Histopathology Images”, p. 17, 2019 (cit. on p. 11).

[10] B. J. Erickson, P. Korfiatis, Z. Akkus, and T. L. Kline, “Machine Learning for Medical Imaging”, *RadioGraphics*, vol. 37, no. 2, pp. 505–515, Feb. 17, 2017. [Online]. Available: `https://pubs.rsna.org/doi/full/10.1148/rg.2017160130` (visited on 07/02/2020) (cit. on p. 11).

[11] R. Siegel, C. DeSantis, and A. Jemal, “Colorectal cancer statistics, 2014”, *CA: A Cancer Journal for Clinicians*, vol. 64, no. 2, pp. 104–117, 2014. [Online]. Available: `https://acsjournals.onlinelibrary.wiley.com/doi/abs/10.3322/caac.21220` (visited on 03/23/2020) (cit. on p. 14).

[12] I. Vogelaar, M. van Ballegooijen, D. Schrag, R. Boer, S. J. Winawer, J. D. F. Habbema, and A. G. Zauber, “How much can current interventions reduce colorectal cancer mortality in the U.S.?”, *Cancer*, vol. 107, no. 7, pp. 1624–1633, 2006. [Online]. Available: `https://acsjournals.onlinelibrary.wiley.com/doi/abs/10.1002/cncr.22115` (visited on 03/30/2020) (cit. on p. 14).

[13] S. Chen and D. Rex, “Endoscopist Can Be More Powerful than Age and Male Gender in Predicting Adenoma Detection at Colonoscopy”, *American Journal of Gastroenterology*, vol. 102, no. 4, pp. 856–861, Apr. 2007. pmid: `17222317`. [Online]. Available: `insights.ovid.com` (visited on 03/30/2020) (cit. on p. 14).

[14] A. C. S. Van Heel, “A New Method of transporting Optical Images without Aberrations”, *Nature*, vol. 173, no. 4392, pp. 39–39, Jan. 1954. [Online]. Available: `http://www.nature.com/articles/173039a0` (visited on 05/25/2019) (cit. on p. 14).

[15] G. Iddan, G. Meron, A. Glukhovsky, and P. Swain, “Wireless capsule endoscopy”, *Nature*, vol. 405, no. 6785, p. 417, May 2000. [Online]. Available: `https://www.nature.com/articles/35013140` (visited on 05/25/2019) (cit. on p. 16).

[16] G. Costamagna, S. K. Shah, M. E. Riccioni, F. Foschia, M. Mutignani, V. Perri, A. Vecchioli, M. G. Brizi, A. Picciocchi, and P. Marano, "A prospective trial comparing small bowel radiographs and video capsule endoscopy for suspected small bowel disease", *Gastroenterology*, vol. 123, no. 4, pp. 999–1005, Oct. 1, 2002. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0016508502002032` (visited on 08/03/2020) (cit. on p. 16).

[17] K. Pogorelov, P. T. Schmidt, M. Riegler, P. Halvorsen, K. R. Randel, C. Griwodz, S. L. Eskeland, T. de Lange, D. Johansen, C. Spampinato, D.-T. Dang-Nguyen, and M. Lux, "KVASIR: A Multi-Class Image Dataset for Computer Aided Gastrointestinal Disease Detection", in *Proceedings of the 8th ACM on Multimedia Systems Conference - MMSys'17*, 2017, pp. 164–169. [Online]. Available: `http://dl.acm.org/citation.cfm?doid=3083187.3083212` (visited on 02/19/2020) (cit. on pp. 21–23, 79, 122).

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255 (cit. on p. 21).

[19] G. Fernández-Esparrach, J. Bernal, M. López-Cerón, H. Córdova, C. Sánchez-Montes, C. R. de Miguel, and F. J. Sánchez, "Exploring the clinical potential of an automatic colonic polyp detection method based on the creation of energy maps", *Endoscopy*, vol. 48, no. 9, pp. 837–842, Sep. 2016. [Online]. Available: `http://www.thieme-connect.de/DOI/DOI?10.1055/s-0042-108434` (visited on 04/20/2020) (cit. on p. 23).

[20] N. Tajbakhsh, S. R. Gurudu, and J. Liang, "Automated Polyp Detection in Colonoscopy Videos Using Shape and Context Information", *IEEE Transactions on Medical Imaging*, vol. 35, no. 2, pp. 630–644, Feb. 2016 (cit. on p. 23).

[21] J. Bernal, J. Sánchez, and F. Vilariño, "Towards automatic polyp detection with a polyp appearance model", *Pattern Recognition*, vol. 45, no. 9, pp. 3166–3182, Sep. 1, 2012. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0031320312001185` (visited on 04/20/2020) (cit. on p. 23).

[22] J. Silva, A. Histace, O. Romain, X. Dray, and B. Granado, "Toward embedded detection of polyps in WCE images for early diagnosis of colorectal cancer", *International Journal of Computer Assisted Radiology and Surgery*, vol. 9, no. 2, pp. 283–293, Mar. 1, 2014. [Online]. Available: `https://doi.org/10.1007/s11548-013-0926-3` (visited on 04/20/2020) (cit. on p. 23).

[23] P. Mesejo, D. Pizarro, A. Abergel, O. Rouquette, S. Beorchia, L. Poincloux, and A. Bartoli, “Computer-Aided Classification of Gastrointestinal Lesions in Regular Colonoscopy”, *IEEE Transactions on Medical Imaging*, vol. 35, no. 9, pp. 2051–2063, Sep. 2016 (cit. on p. 23).

[24] K. Pogorelov, K. R. Randel, T. de Lange, S. L. Eskeland, C. Griwodz, D. Johansen, C. Spampinato, M. Taschwer, M. Lux, P. T. Schmidt, M. Riegler, and P. Halvorsen, “Nerthus: A Bowel Preparation Quality Video Dataset”, in *Proceedings of the 8th ACM on Multimedia Systems Conference*, Jun. 20, 2017, pp. 170–174. [Online]. Available: `https://doi.org/10.1145/3083187.3083216` (visited on 04/20/2020) (cit. on p. 23).

[25] K. Pogorelov, M. Riegler, P. Halvorsen, S. Hicks, K. R. Randel, D. T. Dang Nguyen, M. Lux, O. Ostroukhova, and T. de Lange, “Medico multimedia task at MediaEval 2018”, Dec. 22, 2018. [Online]. Available: `https://bora.uib.no/handle/1956/20930` (visited on 04/20/2020) (cit. on p. 23).

[26] A. Koulaouzidis, D. K. Iakovidis, D. E. Yung, E. Rondonotti, U. Kopylov, J. N. Plevris, E. Toth, A. Eliakim, G. W. Johansson, W. Marlicz, G. Mavrogenis, A. Nemeth, H. Thorlacius, and G. E. Tontini, “KID Project: An internet-based digital video atlas of capsule endoscopy for research purposes”, *Endoscopy International Open*, vol. 05, no. 6, E477–E483, Jun. 2017. [Online]. Available: `http://www.thieme-connect.de/DOI/DOI?10.1055/s-0043-105488` (visited on 04/20/2020) (cit. on p. 24).

[27] H. Bernal J and Aymeric. (2017). Gastrointestinal Image ANAlysis (GIANA) Angiodysplasia D\&L challenge, [Online]. Available: `https://endovissub2017-giana.grand-challenge.org/home/` (visited on 04/20/2020) (cit. on pp. 24, 122).

[28] R. Leenhardt, C. Li, J.-P. L. Mouel, G. Rahmi, J. C. Saurin, F. Cholet, A. Boureille, X. Amiot, M. Delvaux, C. Duburque, C. Leandri, R. Gérard, S. Lecleire, F. Mesli, I. Nion-Larmurier, O. Romain, S. Sacher-Huvelin, C. Simon-Shane, G. Vanbiervliet, P. Marteau, A. Histace, and X. Dray, “CAD-CAP: A 25,000-image database serving the development of artificial intelligence for capsule endoscopy”, *Endoscopy International Open*, vol. 08, no. 3, E415–E420, Mar. 2020. [Online]. Available: `http://www.thieme-connect.de/DOI/DOI?10.1055/a-1035-9088` (visited on 04/20/2020) (cit. on p. 24).

[29] L. R. Lundell, J. Dent, J. R. Bennett, A. L. Blum, D. Armstrong, J. P. Galmiche, F. Johnson, M. Hongo, J. E. Richter, S. J. Spechler, G. N. J. Tytgat, and L. Wallin, “Endoscopic assessment of oesophagitis: Clinical and functional correlates and further validation of the Los Angeles

classification", *Gut*, vol. 45, no. 2, pp. 172–180, Aug. 1, 1999. pmid: 10403727. [Online]. Available: https://gut.bmj.com/content/45/2/172 (visited on 07/05/2020) (cit. on p. 26).

[30] P. Sharma, J. Dent, D. Armstrong, J. J. G. H. M. Bergman, L. Gossner, Y. Hoshihara, J. A. Jankowski, O. Junghard, L. Lundell, G. N. J. Tytgat, and M. Vieth, "The Development and Validation of an Endoscopic Grading System for Barrett's Esophagus: The Prague C & M Criteria", *Gastroenterology*, vol. 131, no. 5, pp. 1392–1399, Nov. 1, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0016508506017914 (visited on 07/05/2020) (cit. on p. 26).

[31] E. J. Lai, A. H. Calderwood, G. Doros, O. K. Fix, and B. C. Jacobson, "The Boston bowel preparation scale: A valid and reliable instrument for colonoscopy-oriented research", *Gastrointestinal Endoscopy*, vol. 69, pp. 620–625, 3, Part 2 Mar. 1, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0016510708019883 (visited on 07/05/2020) (cit. on p. 26).

[32] K. W. Schroeder, W. J. Tremaine, and D. M. Ilstrup, "Coated Oral 5-Aminosalicylic Acid Therapy for Mildly to Moderately Active Ulcerative Colitis", *New England Journal of Medicine*, vol. 317, no. 26, pp. 1625–1629, Dec. 24, 1987. pmid: 3317057. [Online]. Available: https://doi.org/10.1056/NEJM198712243172603 (visited on 07/05/2020) (cit. on p. 26).

[33] H. Inoue, H. Kashida, S. Kudo, M. Sasako, T. Shimoda, H. Watanabe, S. Yoshida, M. Guelrud, C. J. Lightdale, K. Wang, R. H. Riddell, M. D. Diebold, R. Lambert, J. F. Rey, M. Jung, H. Neuhaus, A. T. Axon, R. M. Genta, and J. J. Gonvers, "The Paris endoscopic classification of superficial neoplastic lesions : Esophagus, stomach and colon.", *Gastrointest. Endoscopy*, vol. 58, S3–S43, 2003. [Online]. Available: https://serval.unil.ch/notice/serval:BIB_28728 (visited on 07/05/2020) (cit. on p. 26).

[34] G. Wallace, "The JPEG still picture compression standard", *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, Feb. 1992 (cit. on p. 25).

[35] H. He and E. A. Garcia, "Learning from Imbalanced Data", *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009 (cit. on p. 29).

[36] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique", *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 1, 2002. [Online]. Available: https://www.jair.org/index.php/jair/article/view/10302 (visited on 06/17/2020) (cit. on p. 29).

[37] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks", *Neural Networks*, vol. 106, pp. 249–259, Oct. 1, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608018302107 (visited on 06/17/2020) (cit. on p. 29).

[38] Y. Zou, L. Li, Y. Wang, J. Yu, Y. Li, and W. J. Deng, "Classifying digestive organs in wireless capsule endoscopy images based on deep convolutional neural network", in *Proceedings of the 2015 IEEE International Conference on Digital Signal Processing (DSP)*, Jul. 2015, pp. 1274–1278 (cit. on p. 30).

[39] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998 (cit. on p. 33).

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (visited on 05/31/2019) (cit. on p. 33).

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", presented at the Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034. [Online]. Available: https://www.cv-foundation.org/openaccess/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html (visited on 07/02/2020) (cit. on p. 33).

[42] L. Chen, S. Wang, W. Fan, J. Sun, and S. Naoi, "Beyond human recognition: A CNN-based framework for handwritten character recognition", in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, Nov. 2015, pp. 695–699 (cit. on p. 33).

[43] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification", in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2012, pp. 3642–3649 (cit. on p. 33).

[44] P. Sadowski, "Notes on backpropagation", 2016. [Online]. Available: https://www.%20ics.%20uci.%20edu/pjsadows/notes.%20pdf%20(online) (cit. on p. 35).

[45] N. Qian, "On the momentum term in gradient descent learning algorithms", *Neural Networks*, vol. 12, no. 1, pp. 145–151, Jan. 1, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608098001166 (visited on 05/29/2020) (cit. on p. 38).

[46] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", Jun. 10, 2014. arXiv: 1406.2572 [cs, math, stat]. [Online]. Available: http://arxiv.org/abs/1406.2572 (visited on 05/29/2020) (cit. on p. 38).

[47] Y. NESTEROV, "A method for unconstrained convex minimization problem with the rate of convergence o(1/k$\hat{2}$)", *Doklady AN USSR*, vol. 269, pp. 543–547, 1983. [Online]. Available: https://ci.nii.ac.jp/naid/20001173129/ (visited on 05/29/2020) (cit. on p. 38).

[48] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011. [Online]. Available: http://jmlr.org/papers/v12/duchi11a.html (visited on 05/29/2020) (cit. on p. 38).

[49] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", version 8, Jan. 29, 2017. arXiv: 1412.6980 [cs]. [Online]. Available: http://arxiv.org/abs/1412.6980 (visited on 05/25/2020) (cit. on pp. 39, 100).

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html (visited on 05/29/2020) (cit. on p. 39).

[51] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training Very Deep Networks", in *Advances in Neural Information Processing Systems 28*, 2015, pp. 2377–2385. [Online]. Available: http://papers.nips.cc/paper/5850-training-very-deep-networks.pdf (visited on 06/04/2020) (cit. on p. 40).

[52] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, “Very Deep Convolutional Networks for Text Classification”, Jan. 27, 2017. arXiv: `1606.01781 [cs]`. [Online]. Available: `http://arxiv.org/abs/1606.01781` (visited on 06/04/2020) (cit. on p. 40).

[53] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition”, Dec. 10, 2015. arXiv: `1512.03385 [cs]`. [Online]. Available: `http://arxiv.org/abs/1512.03385` (visited on 06/04/2020) (cit. on p. 40).

[54] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”, Nov. 22, 2019. arXiv: `1905.11946 [cs, stat]`. [Online]. Available: `http://arxiv.org/abs/1905.11946` (visited on 03/05/2020) (cit. on p. 40).

[55] S. Zagoruyko and N. Komodakis, “Wide Residual Networks”, Jun. 14, 2017. arXiv: `1605.07146 [cs]`. [Online]. Available: `http://arxiv.org/abs/1605.07146` (visited on 06/04/2020) (cit. on p. 40).

[56] S. Kannojia and G. Jaiswal, “Effects of Varying Resolution on Performance of CNN based Image Classification An Experimental Study”, *International Journal of Computer Sciences and Engineering*, vol. 6, pp. 451–456, Sep. 30, 2018 (cit. on p. 40).

[57] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “MnasNet: Platform-Aware Neural Architecture Search for Mobile”, May 28, 2019. arXiv: `1807.11626 [cs]`. [Online]. Available: `http://arxiv.org/abs/1807.11626` (visited on 06/05/2020) (cit. on p. 41).

[58] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks”, Mar. 21, 2019. arXiv: `1801.04381 [cs]`. [Online]. Available: `http://arxiv.org/abs/1801.04381` (visited on 06/05/2020) (cit. on p. 41).

[59] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “MnasNet: Platform-Aware Neural Architecture Search for Mobile”, May 28, 2019. arXiv: `1807.11626 [cs]`. [Online]. Available: `http://arxiv.org/abs/1807.11626` (visited on 06/05/2020) (cit. on p. 41).

[60] H. Wu and S. Prasad, “Semi-Supervised Deep Learning Using Pseudo Labels for Hyperspectral Image Classification”, *IEEE Transactions on Image Processing*, vol. 27, no. 3, pp. 1259–1270, Mar. 2018 (cit. on pp. 42, 49).

[61] W. Bai, O. Oktay, M. Sinclair, H. Suzuki, M. Rajchl, G. Tarroni, B. Glocker, A. King, P. M. Matthews, and D. Rueckert, “Semi-supervised Learning for Network-Based Cardiac MR Image Segmentation”, in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2017*, 2017, pp. 253–260 (cit. on p. 42).

[62] A. Madani, M. Moradi, A. Karargyris, and T. Syeda-Mahmood, "Semi-supervised learning with generative adversarial networks for chest X-ray classification with ability of data domain adaptation", in *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, Apr. 2018, pp. 1038–1042 (cit. on p. 42).

[63] Q. Liu, L. Yu, L. Luo, Q. Dou, P. A. Heng, and P. A. Heng, "Semi-supervised Medical Image Classification with Relation-driven Self-ensembling Model", *IEEE Transactions on Medical Imaging*, pp. 1–1, 2020 (cit. on p. 42).

[64] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with Noisy Student improves ImageNet classification", Jan. 7, 2020. arXiv: `1911.04252 [cs, stat]`. [Online]. Available: `http://arxiv.org/abs/1911.04252` (visited on 02/14/2020) (cit. on pp. 42, 43, 98, 116, 121).

[65] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "RandAugment: Practical automated data augmentation with a reduced search space", Nov. 13, 2019. arXiv: `1909.13719 [cs]`. [Online]. Available: `http://arxiv.org/abs/1909.13719` (visited on 04/22/2020) (cit. on pp. 42, 69).

[66] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", p. 30, (cit. on p. 42).

[67] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep Networks with Stochastic Depth", in *Computer Vision – ECCV 2016*, 2016, pp. 646–661 (cit. on p. 42).

[68] P. J. Donnelly and J. W. Sheppard, "Cross-Dataset Validation of Feature Sets in Musical Instrument Classification", in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, Nov. 2015, pp. 94–101 (cit. on p. 47).

[69] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", Mar. 16, 2016. arXiv: `1603.04467 [cs]`. [Online]. Available: `http://arxiv.org/abs/1603.04467` (visited on 05/31/2020) (cit. on pp. 47, 60).

[70] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual.* 2009 (cit. on p. 47).

[71] F. Chollet *et al.* (2015). Keras, [Online]. Available: `https://github.com/fchollet/keras` (cit. on p. 47).

[72] R. Zhu, R. Zhang, and D. Xue, "Lesion detection of endoscopy images based on convolutional neural network features", in *Proceedings of the 2015 8th International Congress on Image and Signal Processing (CISP)*, Oct. 2015, pp. 372–376 (cit. on p. 48).

[73] Y. Yuan and M. Q.-H. Meng, "Deep learning for polyp recognition in wireless capsule endoscopy images", *Medical Physics*, vol. 44, no. 4, pp. 1379–1389, Apr. 2017. [Online]. Available: `http://doi.wiley.com/10.1002/mp.12147` (visited on 05/26/2019) (cit. on pp. 48, 49).

[74] X. Jia and M. Q. Meng, "A deep convolutional neural network for bleeding detection in Wireless Capsule Endoscopy images", in *Proceedings of the 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Aug. 2016, pp. 639–642 (cit. on p. 49).

[75] I. Z. Yalniz, H. Jégou, K. Chen, M. Paluri, and D. Mahajan, "Billion-scale semi-supervised learning for image classification", May 1, 2019. arXiv: `1905.00546 [cs]`. [Online]. Available: `http://arxiv.org/abs/1905.00546` (visited on 06/06/2020) (cit. on pp. 49, 135).

[76] D. Zhang, H. Maei, X. Wang, and Y.-F. Wang, "Deep Reinforcement Learning for Visual Object Tracking in Videos", Jan. 31, 2017. arXiv: `1701.08936 [cs]`. [Online]. Available: `http://arxiv.org/abs/1701.08936` (visited on 05/31/2019) (cit. on p. 50).

[77] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", in *Proceedings of the Medical Image Computing and Computer-Assisted Intervention*, 2015, pp. 234–241 (cit. on pp. 50, 51).

[78] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation", presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440. [Online]. Available: `https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Long_Fully_Convolutional_Networks_2015_CVPR_paper.html` (visited on 06/01/2019) (cit. on p. 50).

[79] M. Turan, Y. Almalioglu, H. Araujo, E. Konukoglu, and M. Sitti, "Deep EndoVO: A recurrent convolutional neural network (RCNN) based visual odometry approach for endoscopic capsule robots", *Neurocomputing*, vol. 275, pp. 1861–1870, Jan. 31, 2018. [Online]. Available:

http://www.sciencedirect.com/science/article/pii/S092523121731665X (visited on 05/26/2019) (cit. on pp. 51, 52).

[80] T. Ping-Sing and M. Shah, “Shape from shading using linear approximation”, *Image and Vision Computing*, vol. 12, no. 8, pp. 487–498, Oct. 1, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0262885694900027 (visited on 06/01/2019) (cit. on pp. 51, 52).

[81] J. Kim, H. Kim, E. Bell, T. Bath, P. Paul, A. Pham, X. Jiang, K. Zheng, and L. Ohno-Machado, “Patient Perspectives About Decisions to Share Medical Data and Biospecimens for Research”, *JAMA Network Open*, vol. 2, no. 8, e199550–e199550, Aug. 2, 2019. [Online]. Available: https://jamanetwork.com/journals/jamanetworkopen/fullarticle/2748592 (visited on 04/21/2020) (cit. on p. 56).

[82] (Apr. 27, 2016). EU General Data Protection Regulation, [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679 (visited on 07/26/2020) (cit. on p. 56).

[83] L. Aabakken, A. N. Barkun, P. B. Cotton, E. Fedorov, M. A. Fujino, E. Ivanova, S.-e. Kudo, K. Kuznetzov, T. de Lange, K. Matsuda, O. Moine, B. Rembacken, J.-F. Rey, J. Romagnuolo, T. Rösch, M. Sawhney, K. Yao, and J. D. Waye, “Standardized endoscopic reporting”, *Journal of Gastroenterology and Hepatology*, vol. 29, no. 2, pp. 234–240, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/jgh.12489 (visited on 07/29/2020) (cit. on p. 59).

[84] S. C. Zammit, A. Koulaouzidis, D. S. Sanders, M. E. McAlindon, E. Rondonotti, D. E. Yung, and R. Sidhu, “Overview of small bowel angioectasias: Clinical presentation and treatment options”, *Expert Review of Gastroenterology & Hepatology*, vol. 12, no. 2, pp. 125–139, 2018. eprint: https://doi.org/10.1080/17474124.2018.1390429. [Online]. Available: https://doi.org/10.1080/17474124.2018.1390429 (cit. on p. 59).

[85] F. Gomollón, A. Dignass, V. Annese, H. Tilg, G. Van Assche, J. O. Lindsay, L. Peyrin-Biroulet, G. J. Cullen, M. Daperno, T. Kucharzik, F. Rieder, S. Almer, A. Armuzzi, M. Harbord, J. Langhorst, M. Sans, Y. Chowers, G. Fiorino, P. Juillerat, G. J. Mantzaris, F. Rizzello, S. Vavricka, P. Gionchetti, and o. behalf of ECCO, “3rd european evidence-based consensus on the diagnosis and management of crohn’s disease 2016: Part 1: Diagnosis and medical management”, *Journal of Crohn’s and Colitis*, vol. 11, no. 1, pp. 3–25, Sep. 2016. eprint: https://academic.oup.com/ecco-jcc/article-

pdf/11/1/3/26359570/jjw168.pdf. [Online]. Available: https://doi.org/10.1093/ecco-jcc/jjw168 (cit. on p. 59).

[86] G. Bradski, “The OpenCV library”, *Dr. Dobb’s Journal of Software Tools*, 2000 (cit. on p. 62).

[87] A. Clark, “Pillow (PIL fork) documentation”, 2015. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf (cit. on p. 62).

[88] P. W. Wong and C. Herley, “Area based interpolation for image scaling”, U.S. Patent 5889895A, Mar. 30, 1999. [Online]. Available: https://patents.google.com/patent/US5889895A/en (visited on 05/19/2020) (cit. on p. 62).

[89] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, “AutoAugment: Learning Augmentation Strategies From Data”, presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 113–123. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Cubuk_AutoAugment_Learning_Augmentation_Strategies_From_Data_CVPR_2019_paper.html (visited on 05/21/2020) (cit. on pp. 69, 75).

[90] D. Masters and C. Luschi, “Revisiting Small Batch Training for Deep Neural Networks”, Apr. 20, 2018. arXiv: 1804.07612 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1804.07612 (visited on 05/22/2020) (cit. on pp. 70, 75).

[91] Y. Bengio, “Practical recommendations for gradient-based training of deep architectures”, Sep. 16, 2012. arXiv: 1206.5533 [cs]. [Online]. Available: http://arxiv.org/abs/1206.5533 (visited on 05/22/2020) (cit. on p. 70).

[92] D. R. Wilson and T. R. Martinez, “The general inefficiency of batch training for gradient descent learning”, *Neural Networks*, vol. 16, no. 10, pp. 1429–1451, Dec. 1, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608003001382 (visited on 05/22/2020) (cit. on p. 70).

[93] Y. Zhang, H. Qu, C. Chen, and D. Metaxas, “Taming the noisy gradient: Train deep neural networks with small batch sizes”, in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, Jul. 2019, pp. 4348–4354. [Online]. Available: https://doi.org/10.24963/ijcai.2019/604 (cit. on p. 75).

[94] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011 (cit. on p. 91).

[95] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves", in *Proceedings of the 23rd International Conference on Machine Learning*, Jun. 25, 2006, pp. 233–240. [Online]. Available: `https://doi.org/10.1145/1143844.1143874` (visited on 07/20/2020) (cit. on p. 92).

[96] V. Cheplygina, M. de Bruijne, and J. P. W. Pluim, "Not-so-supervised: A survey of semi-supervised, multi-instance, and transfer learning in medical image analysis", *Medical Image Analysis*, vol. 54, pp. 280–296, May 1, 2019. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S1361841518307588` (visited on 07/30/2020) (cit. on p. 93).

[97] I. Misra and L. van der Maaten, "Self-Supervised Learning of Pretext-Invariant Representations", presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 6707–6717. [Online]. Available: `https://openaccess.thecvf.com/content_CVPR_2020/html/Misra_Self-Supervised_Learning_of_Pretext-Invariant_Representations_CVPR_2020_paper.html` (visited on 07/30/2020) (cit. on p. 93).

[98] O. J. Hénaff, A. Srinivas, J. De Fauw, A. Razavi, C. Doersch, S. M. A. Eslami, and A. van den Oord, "Data-Efficient Image Recognition with Contrastive Predictive Coding", Jul. 1, 2020. arXiv: `1905.09272 [cs]`. [Online]. Available: `http://arxiv.org/abs/1905.09272` (visited on 07/30/2020) (cit. on p. 93).

[99] A. Karpathy. (Apr. 25, 2019). A Recipe for Training Neural Networks, [Online]. Available: `http://karpathy.github.io/2019/04/25/recipe/#2-set-up-the-end-to-end-trainingevaluation-skeleton--get-dumb-baselines` (visited on 07/31/2020) (cit. on p. 101).

[100] M. Riegler, C. Gurrin, D. Johansen, H. Johansen, P. Halvorsen, M. Lux, C. Gridwodz, C. Spampinato, T. de Lange, S. L. Eskeland, K. Pogorelov, W. Tavanapong, and P. T. Schmidt, "Multimedia and Medicine: Teammates for Better Disease Detection and Survival", in *Proceedings of the 2016 ACM on Multimedia Conference - MM '16*, 2016, pp. 968–977. [Online]. Available: `http://dl.acm.org/citation.cfm?doid=2964284.2976760` (visited on 08/03/2020) (cit. on p. 131).

[101] M. Kirkerød, R. J. Borgli, V. Thambawita, S. Hicks, M. A. Riegler, and P. Halvorsen, "Unsupervised preprocessing to improve generalisation for medical image classification", in *2019 13th International Symposium on Medical Information and Communication Technology (ISMICT)*, May 2019, pp. 1–6 (cit. on p. 134).