Time series analysis for medical videos

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

Spring 2020

Time series analysis for medical videos

Henrik Løland Gjestang

Time series analysis for medical videos

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

Printed: Reprosentralen, University of Oslo

Abstract

Colorectal Cancer (CRC) is the third most common diagnosed cancer in both men and women. And a leading cause for CRC mortality is that patients in early stages have few and defuse symptoms which makes CRC hard to detect before later stages of the disease. To combat this the health care implemented regular screening for population most at risk. Yet for some the treatment come too late. We propose a system for automatically detection of diseases in the gastrointestal tract (GI-tract) using wireless capsule endoscopy and deep learning. A system like this requires a large amount of labeled data to train the prediction models. Although the health sector collects large amount of patient data, the most used deep learning systems need to have access to labeled data to properly train. This labeled data is difficult to create as doctors and other professionals need to manually inspect and annotate. Our proposal uses a model which only need a small amount of labeled data to operate, and an additional bulk of unlabeled data to advance.

Contents

List of Figures

List of Tables

Acknowledgements

This thesis is submitted as part of the master's degree in informatics: Computational Science: Imaging and Biomedical Computing. It has been very interesting to work with ..

I would especially like to thank my supervisor Pål for ..

Thanks to my internal supervisor Professor Anne H. Solberg and the rest of the DSB group, for....

Finally, I would like to thank my parents for their encouragement and everlasting love.

Chapter 1

Introduction

Explain the aims and rationale for the physics case and what you have done. At the end of the introduction you should give a brief summary of the structure of the report. Motivate the reader and give overarching ideas. Describe what has been done and the structure of the report (how is it organized).

1.1 Background and Motivation

**** This entire section is going to be rewritten ****

In this project we aim to design and develop a system for analyzing medical videos 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, Chron's disease, Colorectal cancer, etc. by using video object tracking, object detection, machine learning or other relevant tools.

Neural networks models that we would like to explore further for this purpose are Convolutional neural networks (CNN), Recurrent neural networks (RNN), Capsule neural networks, Long Short-Term memory networks and more.

The main idea is to go beyond image-based methods and also exploit the time factor of the data. The videos we will be using for this is delivered by Bærum Hospital, and is carefully labeled by using tools such as described in the paper "Expert Driven Semi-Supervised Elucidation Tool for Medical Endoscopic Videos." In this paper Albisser *et al.* presents a semi-supervised method to gather the annotations in a easy and time saving way [1].

Figure 1.1: Illustration of how such a camera pill could look like[1].

1.1.1 Nomenclature used in the field of capsule endoscopy

From a variety of literature searches we have found that there is five different terms in use:

- **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.

[1]Image Credit: Medtronic

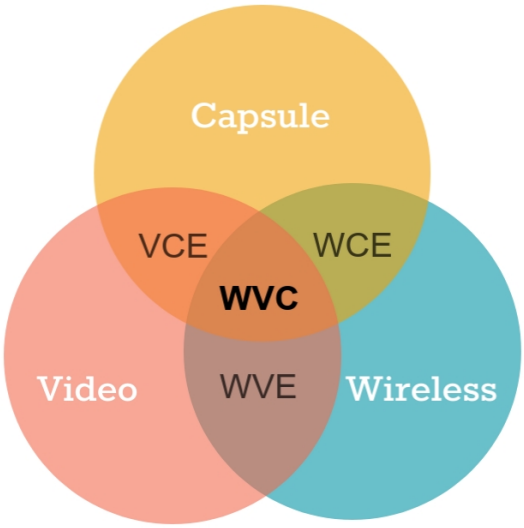


Figure 1.2: Distinction of the nomenclature relating to capsules, wirelessness and video.

The terms are often confused and used interchangeably. Consequently in this thesis they will all be referred to as WCE.

1.2 Problem statement

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.

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). 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 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.

1.3 Scope and limitations

Based on the described problem statement the scope of this thesis is to compare some prior selected automatic systems with focus on machine learning, for classification tasks in the medical domain. We will test the systems on video and still images taken from a

wireless capsule endoscopy system, provided by Bærum Hospital. This data is collected with the Olympus Endocapsule 10 System[2] and gives us a wide view of the GI-tract. A limitation for our research will therefore be that our dataset lack a great deal of diversity. While we might have a wide range of patients all the data contributions stem from one provider in one location. This might not be much of an issue due to GI-tract being quite homogeneous over large populations. But can presumably cause biasing issues in our deep learning models.

By using a tool provided by Augere Medical, we have sequentially classified each frame in XXX WCE videos, assisted by doctor Thomas de Lange, with many years of experience. To the best of our efforts we managed to divide the data into XXX classes which we later will train our networks on.

Considering the scope of this thesis, we will limit ourselves to use some of the more common deep convolutional neural networks and a more novel stacked hourglass network. There are many other more common network used in the field of object classification like NN, RNN, statistical methods, but these methods are widely tested and we wish to push the envelope of machine learning systems on medical videos and also have a wide variety of ways to score our model against previously tested algorithms.

1.4 Research methods

We have based this paper on Association for Computing Machinerys (ACMs) research methodology. 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 [3]. They introduced the phrase "discipline of computing" to embrace the two fields. The task force were given three objectives to complete.

1. Present a description of computer science that emphasizes fundamental questions and significant accomplishments. The definition should recognize that the field is constantly changing and that what is said is merely a snapshot of an ongoing process of growth.

2. Propose a teaching paradigm for computer science that conforms to traditional scientific standards, emphasizes the development of competence in the field, and harmoniously integrates theory, experimentation, and design.

3. Give a detailed example of an introductory course sequence in computer science based on the curriculum model and the disciplinary description.

To fully elaborate the core fundamentals of computer science and computer engineering they agreed upon no core fundamentals is different in the two fields, but the difference is manifested in the way the two fields elaborate the core; computer science focuses mainly on analysis and abstraction and computer engineering focuses mainly on abstraction and design.

Furthermore the task force appointed three major paradigms which represent different areas of competence in the field. Some will argue that the different paradigms

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

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. The three paradigms, or cultural styles, are listed below.

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 object. The paradigm consist of the following four steps.

1. Characterize objects of study (definition);
2. Hypothesize possible relations among them (theorem);
3. Determine whether the relationships are true (proof);
4. Interpret the results found.

When working in this paradigm it is expected to follow this reasoning and iterate the steps when errors or inconsistencies are discovered, until they can be explained and interpreted.

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 for which a scientist is expected to follow.

- Form a hypothesis;
- Construct a model and make a prediction;
- Design an experiment and collect data;
- Analyze the results.

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.

- State requirements;
- State specifications;
- Design and implement the system;
- Test the system.

An engineer is to iterate these steps until the results from the testing satisfy the requirements and specifications of the system.

1.5 Main Contributions

Write a section on how our work contributes to the field... TBW (to be written)

1.6 Thesis Outline

This thesis is split into five chapters. Chapter 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 the thesis is summarized below:

- In Chapter 2 we discuss the literature that focus on the topic of automated lesion detection in computer systems.
- In Chapter 3 we present the details of design, implementation of system and the processing and collection of data.
- In Chapter 4 we present the experiments we have conducted and..
- In Chapter 5 we provide a comprehensive overview of the results found and discus 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, deep learning and artificially intelligence to produce a better, and cheaper health care system. Many of the methods used are 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. The method which we propose 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 background and works 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 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 output. We begin with looking at a basic three layer neural network and build from there up to CNNs and some of the most advanced architectures most recently proposed. This will give a good understanding of how and why we use deep learning to classify medical images.

2.1 Medical scenario

To detect irregularities in the digestive system (Figure 2.1) is a difficult and time-consuming task. To classify irregularities correctly and precisely require expert knowledge. To fully understand the necessity of an automated system for detection lesions in the GI-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 9 meters. It can be so long because the small intestine is very twisty. The GI tract is controlled by the brain by nerves and hormones. The 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.

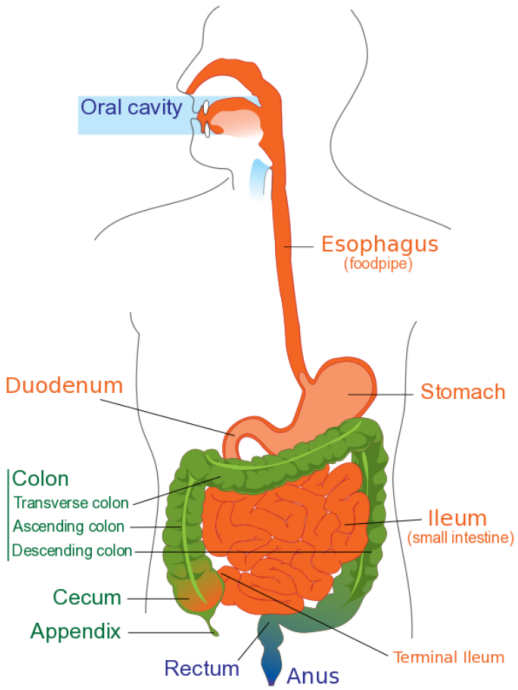


Figure 2.1: An overview of the terms used to describe the digestive system[1].

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 are going to focus on colorectal cancer (CRC). See section 3.1 for list of other diseases from the 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], [4]. And 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

[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`

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 microsimulation called MISCAN-COLON [5] to simulate the 2000 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 3 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 who perform the procedure can be more important than the most important health factors like age and gender [6]. Most screening occurs through endoscopy examinations and is uncomfortable for the patient and cost XXX nurse hours and XXX US dollars. This could be improved by the use of cheaper screening methods like WCE and Artificial Intelligence (A.I). A capsule could be picked up at the local pharmacist, swallowed and then the data sent back to the hospital to be analyzed.

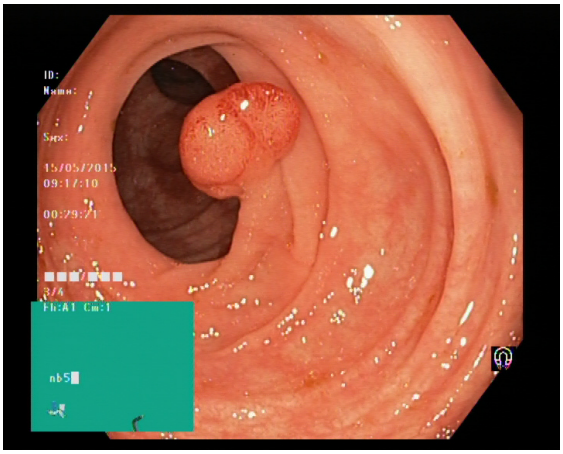


Figure 2.2: Image from Kvasir-V2 dataset of a polyp in the colon, taken with a fiber-optic endoscope (Section 2.5.2).

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 [7]. 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.

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 types. There also are 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 sedated. It 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.

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

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.

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 [8]. It could travel through the entire digestive system because it had no external wires, fiber-optic bundles or cables of any sort. 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 recorder, which is carried on a 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) 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.

Two example images taken by WCE 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

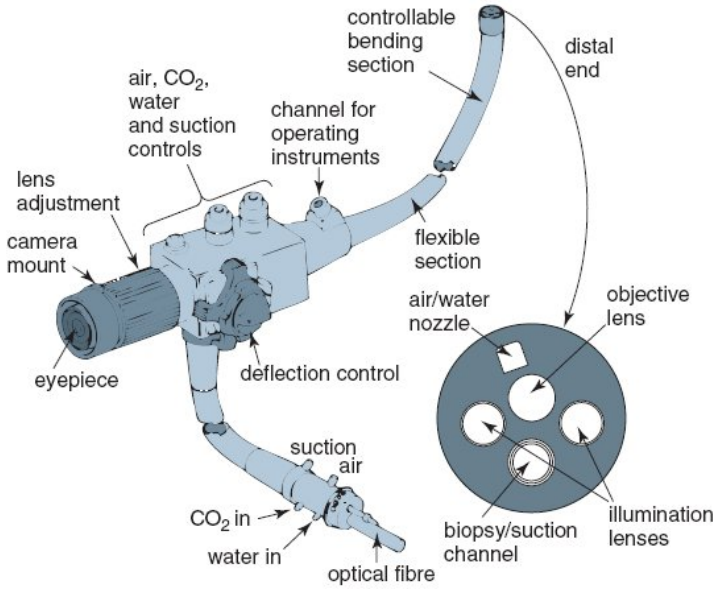


Figure 2.3: Image of a fiber optic endoscope with explanation of different parts of the tool[2].

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 to predict when the capsule exits the stomach through the pylorus.

There is ongoing research done in the field of map prediction (see section 2.7.3)which could be of great interest for WCE technology as it would allow us to better predict the location of a disease inside the patients gut, as well as enable the clinician processing the video to see the orientation of the capsule.

The WCE 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 256x256 to 512x512. They are specialized for different parts of the GI-tract and come from 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 2.4a) and the Olympus RE-10 endocapsule recorder (Figure 2.4b). 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) x 26 mm (length) and it weight 3.3 gram.

A typical video collected by WCE examination lasts a few hours. A clinician must

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

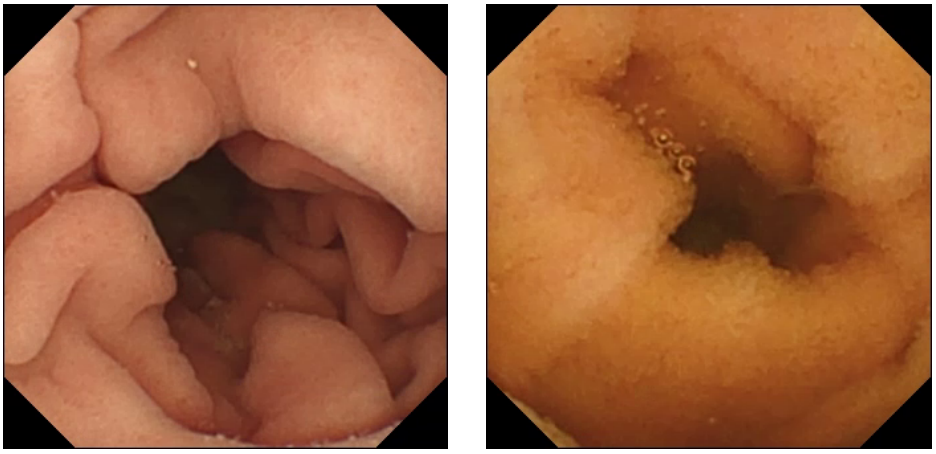
(a) Olympus EC-S10 endocapsule

(b) Olympus RE-10 endocapsule recorder

Figure 2.4: Used WCE equipment.

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 - moments of stillness, camera obscured by food debris and moments of chaotic movements and therefor rapid changes in the imaging area. As such the 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) Stomach **(b)** Small intestine

Figure 2.5: Images from Kvasir PillCam (See Section **??**) dataset taken with WCE.

2.2 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 [9]. The DCNN-based WCE 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 [9]. The main steps of this convolutional neural network are described in detail in section 2.2.2.

2.2.1 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.

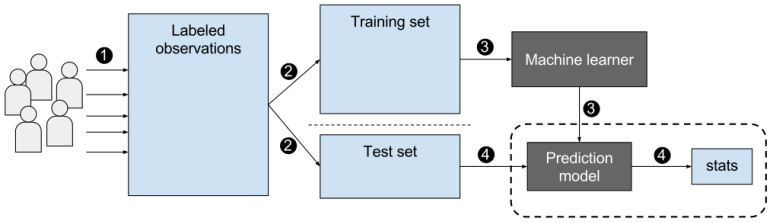


Figure 2.6: Workflow of supervised learning; 1. the dataset is labeled by observers; 2. the samples is split into training and test sets; 3. algorithm is learning on the training set; 4. checking the predictions on the 'unseen' test set to understand how the model performs.

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.

Unsupervised learning

Large, cleanly labeled datasets are not does 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.
- **Autoencoders**: Autoencoders 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 2.7. 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.

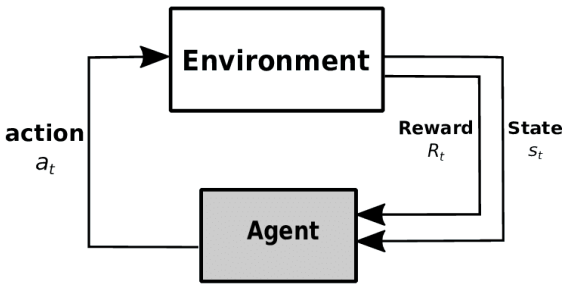


Figure 2.7: Reinforcement learning: Agent and environment.

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.

2.2.2 Convolutional Neural Network

One of the most used neural networks for image classification is the Convolutional Neural Network (CNN). The model was first proposed by Krizhevsky *et al.* in 2012 [10] where they trained a deep convolutional neural network and used it to classify 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes with top-1 and top-5 error rates of 37.5% and 17.0% which far surpassed all other models at the time. Next we will get into a bit of the details of a CNN.

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 [11]. 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.

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

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

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

2.2.3 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:

$$\theta = \theta - \eta \cdot \nabla J(\theta)$$

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:

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

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:

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

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 [12] fixes one of the main problems with normal SGD, which is that is will often get stuck in in saddle points [13] 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:

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

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) [14]. 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:

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

Adaptive gradient based learning algorithms

Adagrad (ADAptive GRADient algorithm) [15] 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:

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

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) [16] 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:

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

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

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

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 Network architectures

2.3.1 ResNet

Residual Netowrk (ResNet) [17] 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.

2.3.2 EfficientNet

EfficientNets [18] 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, width, and resolution. Depth is how many layers are in the model. 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. 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 uses compound scaling to uniformly scale network width, depth and resolution.

2.3.3 Student teacher model

Write about and discuss noisy-student paper.

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 we can not train on all of the 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%).

Figure 2.8: A visualization of the splits.

In Figure 2.8 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.

2.4.2 Performance metrics

In the upcoming few sections we will introduce the reader for some of the metrics used to evaluate a model.

Confusion matrix

2.4.3 Cross validation

2.4.4 Cross dataset validation

2.5 Datasets

The datasets used in our experiments are Hyper-PillCam, Kvasir and Hyper-Kvasir. This section will demonstrate the main differences between the three datasets and explain how they can be found and used for fact checking. All three datasets are collected using endoscopic equipment at Vestre Viken Health Trust in Norway. The VV 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 WCE videos. In the later part of this section we will discuss some of the difficulties of the aforementioned datasets.

2.5.1 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-PillCam 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.

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 [24]	Various	GI lesions & landmarks	8,000 images, 8 classes, 1,000 images per class	Available, public, free for research and educational purposes	See Section 2.5.2 for the description
Hyper-Kvasir [25]	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 2.5.3
Nerthus [26]	Colonoscopy	GI findings	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	XXX
Medico [27]	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	XXX

Table 2.2: Existing colonoscopy image and video datasets.

[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
KID [28]	VCE	Angiectasia, bleeding, inflammations, polyps	2,500+ images + 47 videos	Discontinued	Open academic
GIANA'17 [29]	VCE	Angiectasia	600 images	Available, by request	Includes ground truth segmentation masks
CAD-CAP [30]	VCE	Normal, Vascular Lesions and Inflammatory Lesions	25,000 images	Discontinued	By request
Kvasir-PillCam	VCE	GI lesions, landmarks and findings	XXX images with ROI, XX classes and XXX unlabeled images/videos	Available, public, free for research and educational purposes	Ours, See Section 3.1.2.

Table 2.3: An overview of existing VCE datasets from the GI tract.

Class number	Class name	Number of samples
0	normal	8000
1	polyp	8000
2	polyrus	8000

Table 2.4: Kvasir-V2 class names and corresponding class numbers.

2.5.2 Kvasir-V2

The Kvasir dataset [24] 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.

2.5.3 Hyper-Kvasir

The Hyper-Kvasir dataset [25] is one of the largest medical datasets containing 110.079 images and 373 videos where it captures anatomical landmarks and pathological and normal findings. 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 `https://datasets.simula.no/hyper-kvasir/`.

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.

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. This is important to highlight because it benefits other

Class name	Samples	Class name	Samples
barrets	41	normal-z-line	932
barretts-short-segment	53	polyps	1028
bbps-0-1	646	pylorus	999
bbps-2-3	1148	retroflex-rectum	391
cecum	1009	retroflex-stomach	764
dyed-lifted-polyps	1002	ulcerative-colitis-0-1	35
dyed-resection-margins	989	ulcerative-colitis-1-2	11
esophagitis-a	403	ulcerative-colitis-2-3	28
esophagitis-b-d	260	ulcerative-colitis-grade-1	201
hemorrhoids	6	ulcerative-colitis-grade-2	443
ileum	9	ulcerative-colitis-grade-3	133
impacted-stool	131		

Table 2.5: Hyper-Kvasir class names and corresponding amount of samples.

researchers who is in need of similar data can access the dataset easily.

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 [31], 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 you can see the 23 classes and how many images there is for each class.

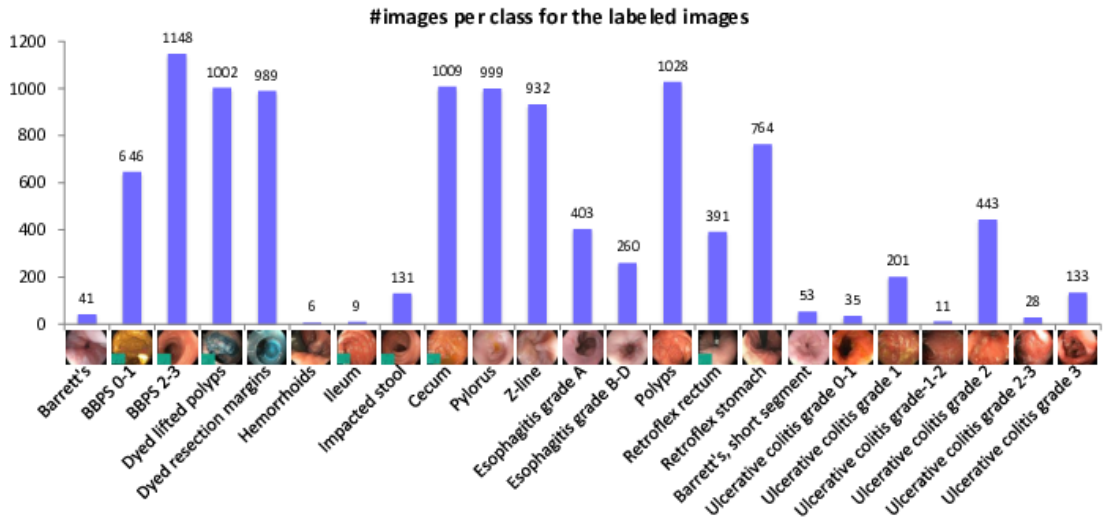


Figure 2.9: Number of samples for each of the 23 classes in Hyper-Kvasir dataset.

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.

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 Region of Interest (ROI) are represented by the white mask while the black does not contain polyp pixels. In Figure 2.10 we can see an example of the segmented Kvasir images.

Figure 2.10: Example of a segmented image from Hyper-Kvasir dataset.

Videos

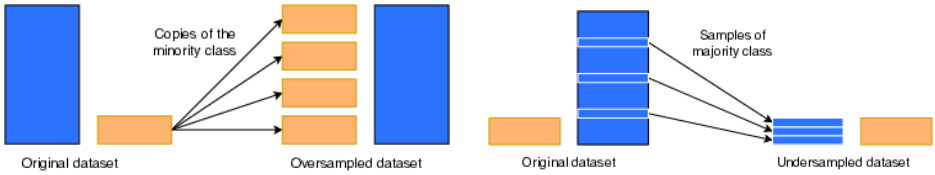
In total there are 373 videos provided in the dataset, corresponding to 11.62 hours of videos and about 1 million video frames that can be convertet to images. The file format for the videos are Audio Video Interleave (AVI). The video portion of the dataset is 38.6GB in file size. In addition to the video folder there is a CSV file provided containing the videos IDs and findings. The finding contains the description of the finding in the video, of which there is 171 of. Finding description is meant to describe the video as a whole. Some of the findings are related to the categories found in the image portion and some are unique for the videos.

2.5.4 Augere Medical AS

Write about Augere Medical, the tagging tool and export script used.

2.5.5 Imbalanced data

Class imbalance typically refers to a problem with classification problems where the classes are not represented equally. In the medical imaging domain this is very common



(a) Oversampling **(b)** Undersampling

Figure 2.11: Two popular resampling methods.

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 tissue than there are images of tissue with lesions. 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 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.

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

Resample

In Figure 2.11a is an example for how samples in a class are copied to generate a new, balanced dataset.

2.6 TensorFlow Framework

2.6.1 tf.data

2.7 Related work

Zhu *et al.* have made a computer-aided lesion[7] 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 [32]. 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.

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

Yuan *et al.* have accomplished an average overall recognition accuracy of 98.0% for detecting polyps in WCE 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 intensities from the learned low-dimensional features [33]. 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 WCE images. Gastrointestinal (GI) 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 [34]. 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.

We will go through some other methods not directly related to neural networks but which we think may come in handy for my thesis later on.

2.7.1 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" [35]. 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 WCE 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.

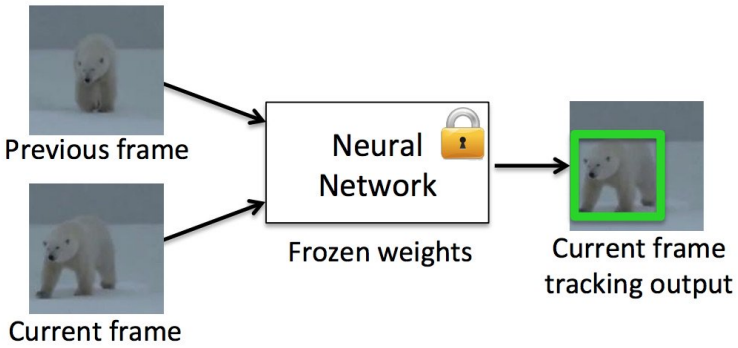


Figure 2.12: Illustration of how object in two frames is tracked with a bounding box[8].

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.

2.7.2 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" [36] 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" [37] such that it works with very few training images and yields more precise segmentations. 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 upsampled 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 personell 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

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

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**

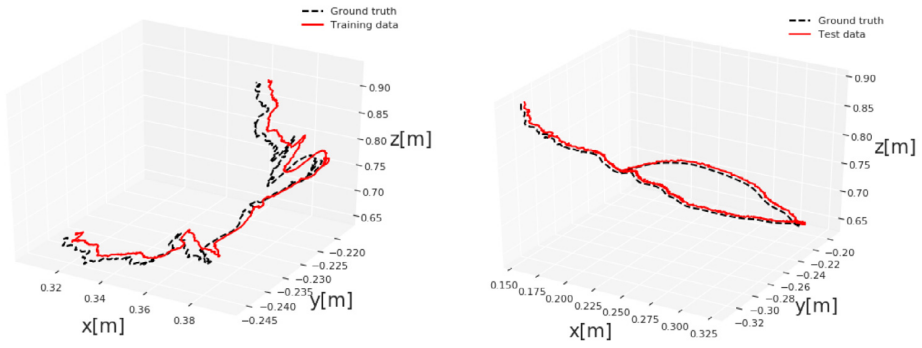
Table 2.6: Segmentation results on the ISBI cell tracking challenge in 2015.

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 [36]. By doing this Ronneberger *et al.* were able to achieve very good results (Table 2.6).

2.7.3 Mapping

As mentioned in section 2.1.4, a concern when processing the images taken with a WCE 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 RCNN's[9] [38]. See Figure 2.13 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 [39]. 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 2.13: An example of Deep EndoVO accuracy [38].

[9]Deep recurrent convolutional neural networks

2.8 Summary

Chapter 3

Methodology

In this chapter we will discuss how we created the dataset used in our research, what the dataset contains and the purpose for which we created it.

3.1 Data collection

As discussed in Section 2.5 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 collected for the purpose of deep learning. To assist the under-explored 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 for non-commercial, educational and research purposes.

3.1.1 Privacy, Legal and Ethics Issues

To obtain medical videos 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 **Pasientjournalloven??** legislation. A medical study conducted at 2 academic hospitals from May 2017 to September 2018 found that most patients are willing to share their data and bio-specimens for research purposes [40]. Regardless of the patient opting in to share their data and bio-specimens it is difficult for researchers to get their hands on it. We solved this problem by collaborating with a number of Norwegian hospitals and research teams working there. One of the research teams we collaborated with is Augere Medical (See Section 2.5.4 for more info), and through them we got in contact with Vestre Viken Hospital Trust, allowing our research team to download anonymous data from hospital systems and transfer it using secure media to our facility. Upon downloading the data we further stripped the metadata files for potential information regarding patients like time stamps, dates and camera equipment used.

3.1.2 Hyper-CasuleCam

The dataset we used in our experiments consist of endoscopic videos collected from Bærum Hospital, a hospital in Vestre Viken Hospital Trust. Unlike Kvasir V2 and Hyper-Kvasir datasets we have made the Hyper-CasuleCam dataset for the purpose of this thesis. Initially we received 44 VCE videos, which were first reviewed by a clinician, whom selected thumbnails of region of interests of both lesions and normal findings. The videos were then exported from Vestre Viken Hospital Trust and re-encoded to an open source file format (MP4?), from proprietary Sony technology format (whats the name?) Prior to being exported the videos were anonymized by removing all metadata and renaming the files with randomly generated file names. A few videos had to be shortened to cut out images taken by the capsule before 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 and labeled and marked findings with bounding boxes. When the student encountered images they were uncertain of the expert reviewed the case.

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 because in many medical datasets the minority class is the most important and therefore more sensitive for classification errors.

In addition to labeling the images the dataset also contain a JSON format file which stores coordinates for where in the frame the finding is located. The Kvasir-PillCam dataset will be an open-source dataset available for others scientists, and will later be grown to include more PillCam videos, both labeled and unlabeled samples.

Labeled images

There are a total of XX different findings in the dataset. The findings are split in two different types; anatomical landmarks and pathological findings. Anatomical landmarks consist of two categories:

- Pylorus valve; where the capsule enters the small intestine from the stomach.
- Ileocecal valve; the transition from small intestine to large intestine.

Pathological findings consist of:

- Protruding lesions; polyps, varices and tumors.
- Excavated lesions; scars, ulcers, erosions, angiectasia, and dieulafoy lesions.
- Mocusa; edematous, scalloping, hemorrhagic, erythematous, and ulcerated mucosa.
- Normal; no pathological finding.

We have also included one category for foreign bodies which includes images with things found inside the small intestine which does not fit in any of the previously mentioned categories. This includes thing like other VCE-devices and tables. The

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

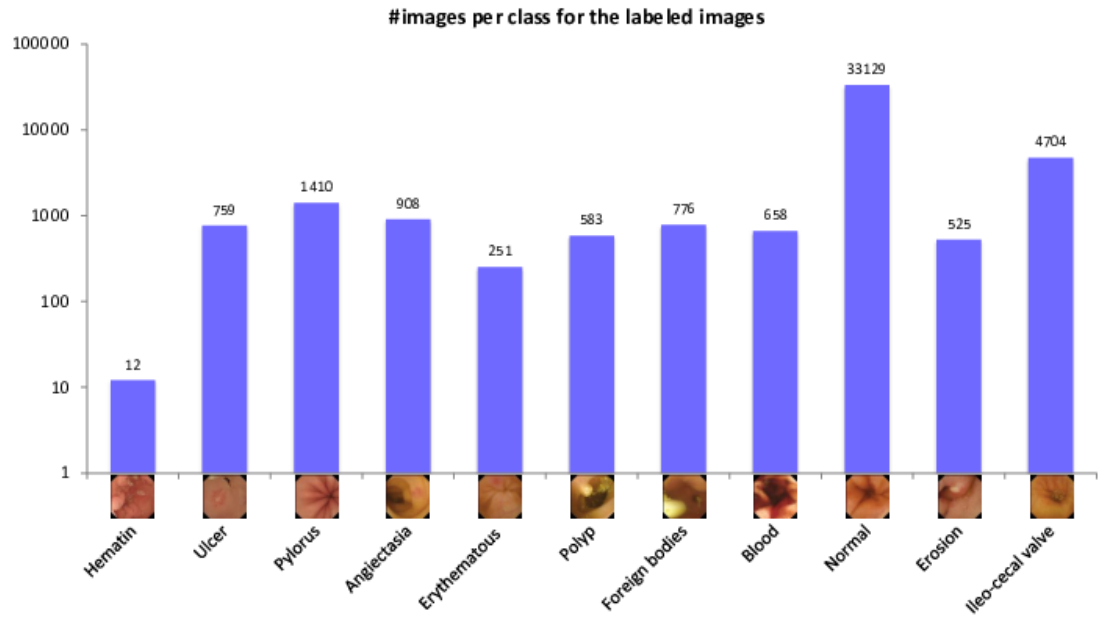


Figure 3.1: The distribution of samples per class in the Hyper-CapsuleCam dataset. NB, the Y-axis of this plot is in algorithmic scale.

images annotated with the *normal* tag is meant to be used for binary classification where all the others classes are combined into a single class for *finding* and one class for no findings. *Normal* images are hand picked from portions of video where there are no findings, and taken from each video to get a diverse category.

When all 44 videos have been precisely labeled the dataset is exported from Augere Medical tagging tool and split into folders for each class. The number of images per class are given in Table 3.1 and Figure 3.1 with an algorithmic scale. In total we have 43,905 labeled images in 12 classes. The sample distribution across the classes is skewed depending on how many findings there are in the videos. Some findings occur often and some very rarely.

Unlabeled images

Later, we received an additional XX videos. These videos were not annotated but used for unlabeled data. (and/or sample-videos?) These videos were processed exactly the same as previous videos, so that it will be compatible (**describe**) with the annotated dataset.

Videos

In addition to the labeled and unlabeled images all of the videos used for extracting them are included in the dataset in MP4 format. They are weakly labeled by a clinical expert.

Class name	# samples	Percentage
Angiectasia	908	xx,x
Blood	658	xx,x
Erosion	525	xx,x
Erythematous	251	xx,x
Foreign Bodies	776	xx,x
Hematin	12	xx,x
Ileo-cecal valve	4704	xx,x
Normal	33,129	xx,x
Polyp	583	xx,x
Pylorus	1410	xx,x
Ulcer	759	xx,x
Unknown	190	xx,x

Table 3.1: Hyper-CapsuleCam class names, corresponding amount of samples and the percentage of samples in each class.

3.2 Data pipeline

The preprocessing pipeline is implemented by using TensorFlows *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 GPU which enhance the load distribution of processing power. In Figure 3.2 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.

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 3.3.

3.2.1 Splitting and resize images

Although not strictly a part of the pipeline itself the 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]
```

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

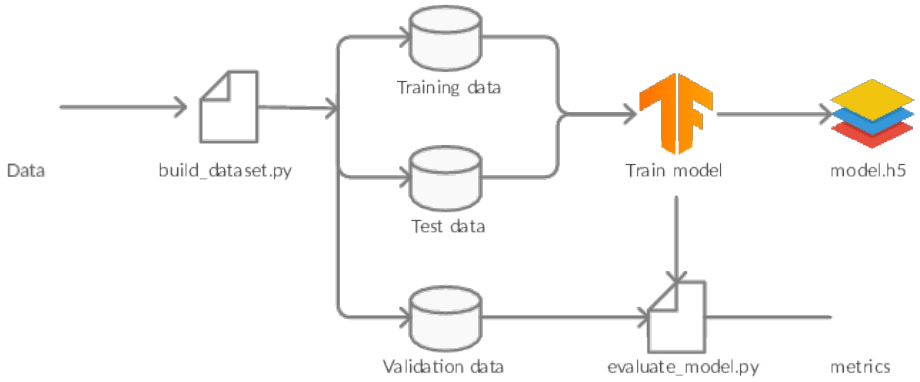


Figure 3.2: The pipeline used for training our teacher model.

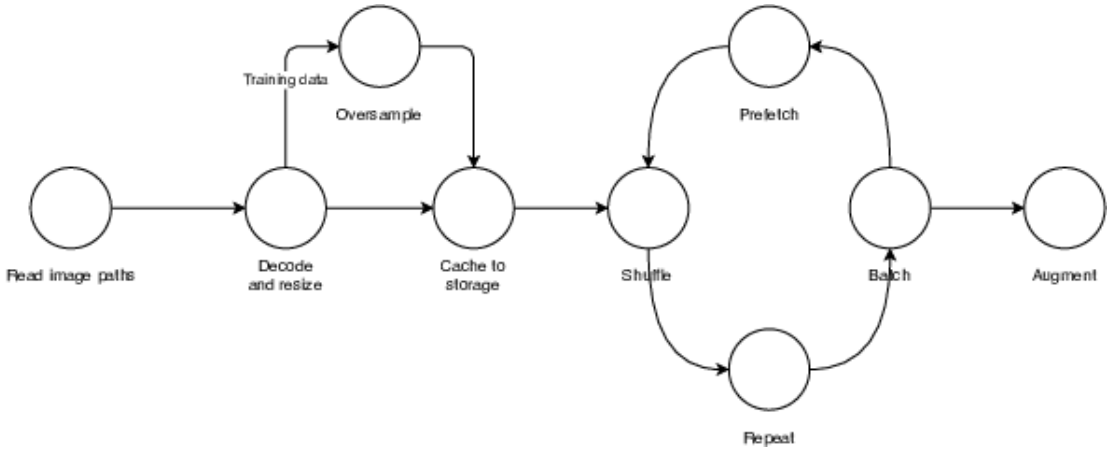


Figure 3.3: The input pipeline we use for feeding our network with image-label pairs during training.

often all end up in one of the dataset splits and not be represented in the other two. To mitigate this issue I 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
```

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 file names 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 by 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's, with a bilinear interpolation. We used openCV because for this particular task it was about 30% faster than Pillow, 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 by 576 pixels, which means we are reducing the dimensions with a factor of three. Area based interpolation [41] or Gaussian resampling, with a suitable chosen radius, may give better results. This is something I would like to have tested if I 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.

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 jpg 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

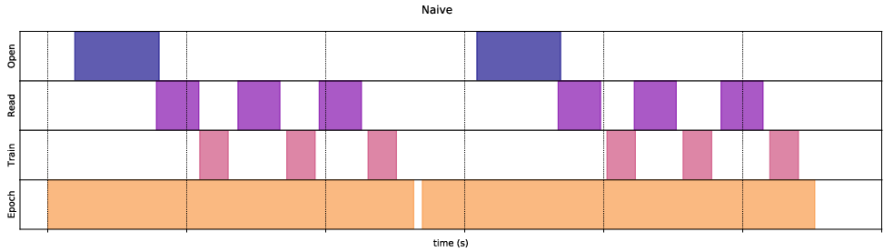


Figure 3.4: The time spent for reading, opening and training - the naive method.

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)
```

Next we apply a transformation function to each element of the dataset to create (image, label) pairs from the 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 a efficient pipeline it is required that the Graphical Processor Unit (GPU) is fed data at the right time. If the GPU is waiting to receive data the pipeline is not optimized and training 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 file, (2); reading the data from that file and (3); using the data for training. 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 3.4.

To mitigate this time-consuming issue the tf.data API have a couple of tools at disposal. 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 3.5. 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 CPU from the time when data is consumed by the GPU.

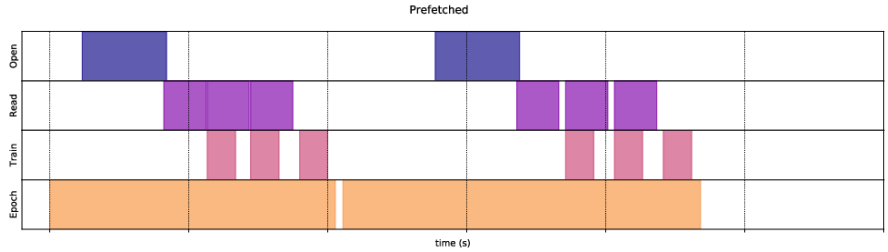


Figure 3.5: Time spent on reading, opening and training with pipeline prefetching enabled.

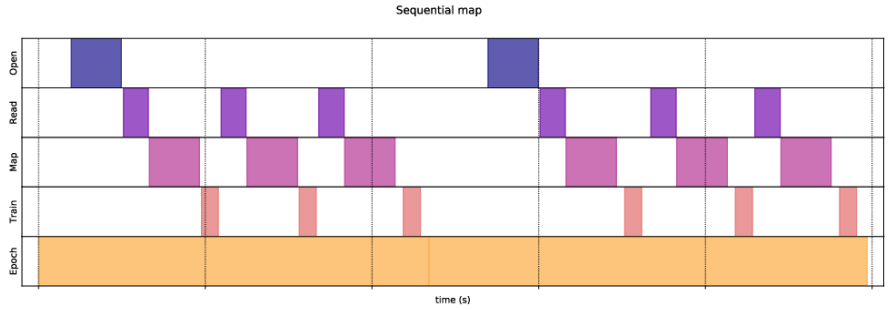


Figure 3.6: Naive pipeline, here the times spent on opening, reading, pre-processing and training steps sum together for a single iteration.

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 parallelized across multiple CPU cores. The tf.data API provides a num_paralell_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 3.6 for the naive approach, and Figure 3.7 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 SSD 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 3.8 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. 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.

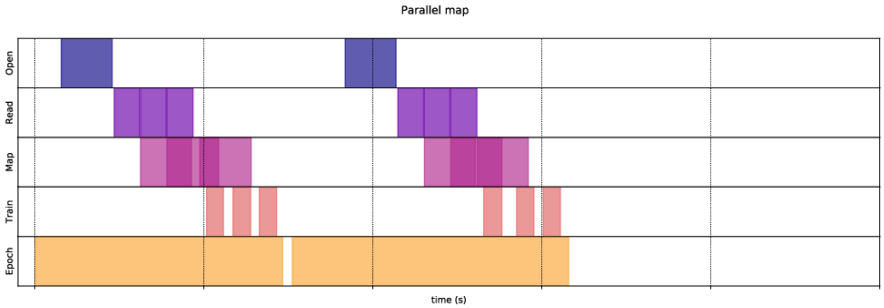


Figure 3.7: Here you can see pre-processing steps overlap, and the overall time for each iteration is reduced.

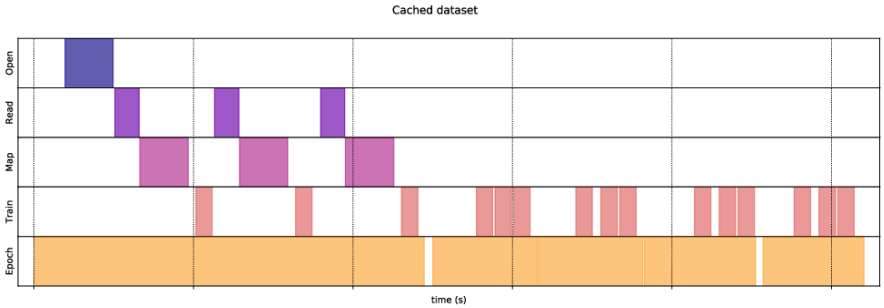


Figure 3.8: Cached pipeline, here the pre-processing is only executed during the first training step.

3.2.4 Shuffle the dataset

We shuffle the data to reduce variance and to make sure the model remain general and overfit less. This is especially important as our dataset is sorted by their classes. 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 important because we have the risk of creating batches that are not representative of the overall dataset, and therefore gradient estimate will be off.

TensorFlows 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 an 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, .....]
```

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.

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

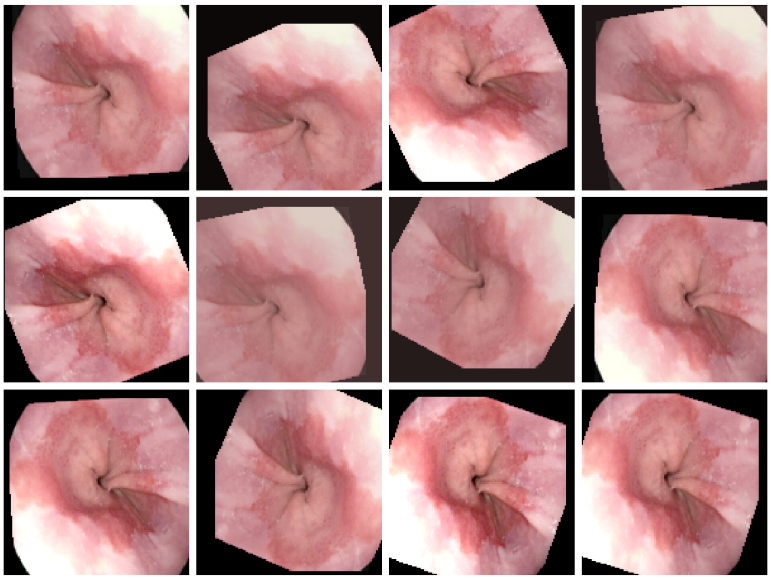


Figure 3.9: The effect of data augmentation on an image from Hyper-Kvasir. Image taken from barrets-short-segment class. In this example the augmentations performed are a bit excessive to underline the effects.

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 3.9 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

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

  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 by 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 3.10 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. 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 [42], [43]. 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

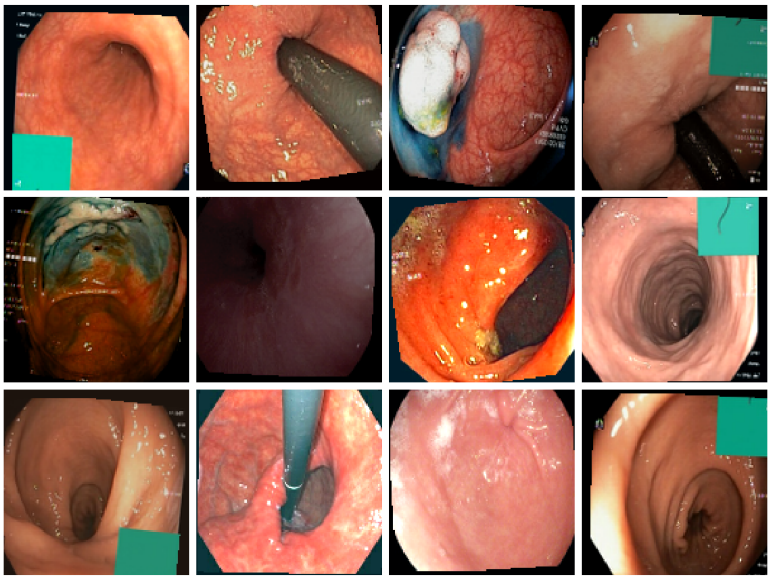


Figure 3.10: The effect of reduced data augmentation on images from Hyper-Kvasir dataset. Here the effect of cropping is reduced by padding the image with 10% of its original dimensions, and rotation is reduced to max 10 degrees.

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 3.1: 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 [44]. Masters

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 3.2: A table showing the system specifications for the machine used for all training and evaluation sessions.

Model	128x128		256x256	
	Batch Size	Time [s]	Batch Size	Time [s]
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 used for each EfficientNet model with image dimensions 128 by 128 and 256 by 256 respectively.

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 [45], [46]. 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 by 256 being trained on EfficientNetB0 which has 5.3 million parameters.

3.2.8 Sampling the data

3.3 Training our networks

Due to memory limitations we are forced to reduce the batch size when training with larger models. If we run with a too large batch size the model will not fit in memory and we get a Out of Memory error (OOM Error). In Table 3.3 is a overview of what batch sizes we are using for which models during training.

3.3.1 Weight initializing

3.4 Binary classification

As previously mentioned one of the main concerns when dealing with medical image data is that there are a lot easier to come by images with no medical findings, than there is images with findings. While labeling images from the Hyper-CapsuleCam dataset at the beginning of this study, we quickly noticed this was an issue. After spending weeks labeling data, the total corpus of images with findings were in the thousands, while the images we had tagged as normal, healthy, images were in the tens of thousands.

After some testing with using ResNet50 model initialized with ImageNet pretrained weights, we confirmed that the model did not generalize well, and would classify all images as the majority class which greatly outnumbered the other classes.

Next, we merged all classes with labeled findings into one big class, while leaving the class with healthy images as is. This gave us a binary dataset of forty thousand images. This class-label merge was done in the input pipeline by creating a list of all class names with findings, and upon reading an image from the dataset directory, we checked if the image paths contained one of the class names from the list. If so, we labeled the image as 1, which represents 'positive' and if not the image was labeled with 0, representing 'negative'.

The dataset was split into training, testing and validation datasets with the split of 60%, 15% and 15% respectively.

3.4.1 Learning rate

Many models train better if you gradually reduce the learning rate during training. We are using learning rate which follows a inverse time decay which means that the learning rate is decreased to 1/2 of the base rate at x epochs, 1/3 at $2x$ epochs and so on. We find that by doing this we get a model which has a less fluctuating loss during training. In Figure 3.11 we have trained two models, identical except in Figure 3.11a the learning rate is set to 0.001 during all 30 epochs, while in Figure 3.11b the learning rate starts at 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. When comparing the accuracy of the trained model on the test data we see they both perform equally well after 30 epochs, but the model trained with the use of inverse time decay learns slightly slower than the model trained without.

For further studies we would like to run grid search to find a more optimal value for learning rate.

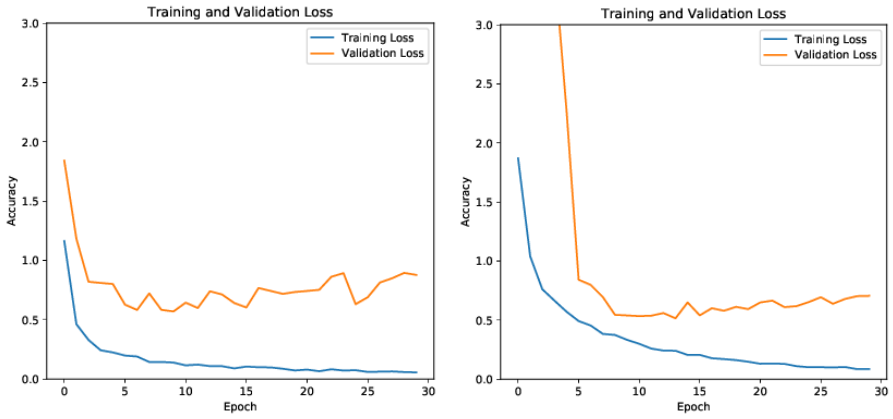
3.4.2 Optimizer for gradient descent

3.4.3 Hyper-parameter tuning

3.4.4 Evaluation methods and metrics

3.5 Sample the dataset vs weighting the classes

To combat the class imbalance in our dataset we have experimented with two tactics. The first thing we tested, which was also the easiest to implement with TensorFlow was



(a) Learning rate set to 0.001. **(b)** With inverse time decay.

Figure 3.11: Effect of learning rate inverse time decay on the computed loss during training. Here both models are trained by using Adam, batch size of 128, with data augmentation and oversampling and 20% dropout.

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.

Oversample

Oversampling the training dataset is handled by a oversample() 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 one dataset object for each class, containing as many samples as there were originally. The next step is to chache 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 infinitely 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, label pairs when iterated over.

```
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"]
)
```

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 0.00054 samples, while the class bbps-2-3 have a ratio of 0.10773, that is on a order of two magnitudes more samples. The output from after the oversampling shows that all classes have, almost, the same distribution of $\frac{1}{23} = 0.04348$.

```
---- Ratios before resampling ----
[0.00496378 0.07163939 0.00321975 0.01247652 0.02441642 0.09283606
 0.00053662 0.08746982 0.03783204 0.00093909 0.00375637 0.10772739
 0.00080494 0.06063858 0.01220821 0.09471425 0.04158841 0.00254897
 0.09377515 0.03662463 0.01878186 0.09645828 0.09404346]

---- Ratios after resampling ----
[0.04111328 0.04501953 0.04589844 0.04414063 0.04443359 0.04189453
 0.04033203 0.03916015 0.04453125 0.04619141 0.04091797 0.04628906
 0.04345703 0.04160156 0.04453125 0.04296875 0.04580078 0.040625
 0.04560547 0.04404297 0.04238281 0.04257812 0.04648437]
```

Class weights

The way we calculate the class weights for our data is by the following formula.

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

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

In tf.Keras we can then give the class weights as a dictionary to the model.fit function like this:

```
class_weights = {"class A": 0.5,
         "class B": 0.25}

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

What we are doing here is to tell Keras that class A should hold 50% of the weight for the loss function since it is more important than class B which we accordingly set to 25%.

In the case of Hyper-Kvasir dataset the class weights we use for our experiments are given in Table 3.4.

```
---- Class weights ----
{0: 8.759107, 1: 1.0, 2: 13.503623, 3: 3.484806, 4: 1.7806976, 5: 1.0, 6:
    81.021736, 7: 1.0, 8: 1.1492445, 9: 46.298138, 10: 11.574534, 11:
   1.0, 12: 54.014492, 13: 1.0, 14: 3.5613952, 15: 1.0, 16: 1.0454417,
   17: 17.057209, 18: 1.0, 19: 1.1871318, 20: 2.3149068, 21: 1.0, 22:
   1.0}
```

Notice how the minimum range for class weights are set to 1.0. This is done to stabilize the loss. **find reference and reason** Found that when weighting the classes the model learns slower. Which means decay rate of learning rate must be lowered for it not to drop too soon. Also increase the buffer for early stopping of training.

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

Table 3.4: Hyper-Kvasir class weights. All weigts below 1.0 are set to a minimum of 1.

3.6 Teacher-student architecture

The specific implementation we use is EfficientNets release v1.1.0 by qubvel[2]. This tf.keras implementation comes with pre-trained weights for both ImageNet and Noisy-Student.

3.6.1 Training the teacher model

For every iteration of putting the student back as the teacher, we tune the parameters for training step size, the level of augmentation and model dropout.

3.6.2 Generating new pseudo labels

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 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.

After subtracting samples from the unlabeled dataset through predicting and saving images with a probability score higher than the threshold, we sort the images based on the probability scores. This is done so for majority classes with a large amount of samples can be squeezed to fit the minority classes and we can select the samples with the highest probability scores.

The sorted samples from the unlabeled dataset is then resampled based on the original distribution of samples in the training data. Since the original dataset is resampled that means we will pick out X samples for each class to keep the distribution even between classes during training.

[2]github.com/qubvel/efficientnet

Inspecting the pseudo labels

Since the unlabeled dataset don't contain labels, it is difficult to fully comprehend the results of the extracted data.

3.7 Summary

Chapter 4

Experiments and results

In this section, we will first describe our experiment details. We will then present our ..? Lastly, we demonstrate the surprising improvements of our models on domain specific image classification tasks with Hyper-Kvasir and Hyper-CapsuleCam datasets.

4.1 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 hyperparameters, as well as differentiate teacher models from student models, the number of 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.

4.2 Evaluation method and metrics

4.3 Training details

For labeled images we re-scale the images to 128 by 128 pixel 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 the model into the memory. 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

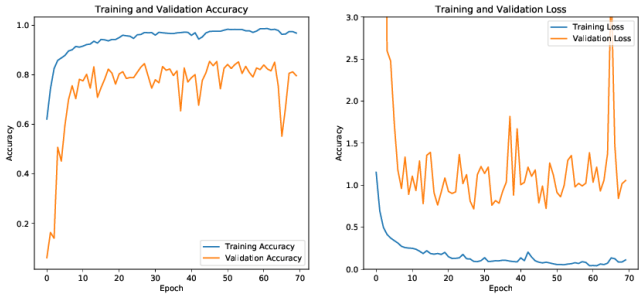


Figure 4.1: Example where the model accuracy for training data greatly outperforms the accuracy for testing data.

batch size for labeled images. The training steps used are calculated by

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

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 4.1 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 don't 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.

The optimizer we use is for our experiments are Adam [16], 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 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 manage to get slightly better results by starting the learning rate at 0.01, and then using inverse time decay (another Learning Rate Scheduler), with a steep learning rate drop-off in the first epochs and then the learning rate plateau after about 10 epochs.

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. This model suits us good for two reasons (1), it comes with pretrained ImageNet weights, and (2) it is easy to test different model sizes. The EfficientNets (B0-B7) are used for all our experiments. This base model is then connected with a pooling layer, a dropout layer, a fully connected layer, another dropout layer and finally a output layer. The dropout is set to 10% for the case of training the teacher model, and 30% for the training the student model on the combined labeled data and generated pseudo labels.

We find that by using dropout and image augmentation the model generalizes much better to 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 after switching the teacher with the student we increase the image augmentation multiplier to 100%.

4.3.1 Labeled and labeled 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 by 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.

4.3.2 Architecture

We use EfficientNets at the base of our model since it handles large datasets exceptionally well.

4.4 Experiments

4.5 Binary vs multiclass

4.5.1 Weight initialization

Found that training on Hyper-Kvasir datasets yield best results when using EfficientNet with is initialized with weights from ImageNet. Early in our study the only pre-trained weights readily available were trained without the use of AutoAugment, and performed slightly worse than the ImageNet weights trained with AutoAugment, released at a later time. Especially we observed that the model would initialize the training with lower loss when trained with ImageNet-AutoAugment weigths.

We tested with initializing the teacher model with original noisy-student weights but found that loss were fluxing more early in training. Also 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.

As part of this experiment we also tested with initializing the teacher model with no weights and train from scratch. Not surprisingly, this gave slower training, and after 20 epochs we got a 15% lower accuracy than when training with ImageNet weights.

4.5.2 Class Weigting vs Resample

Found that by re-sampling the dataset the model generelized much better than by calculating the loss from weighted class distribution.

Chapter 5

Conclusions

5.1 Results

5.2 Summary and contributions

We have looked on some highly relevant papers written about automatic detection systems for medical videos from the last few years. From back when feature extraction methods consisted of selecting color and intensities thresholds, to newer and more sophisticated algorithms like CNN's have become mainstream. The newer methods may be more complex and harder to implement but we have found that these automatic feature extraction methods have a far higher accuracy and produce less false positives. We have also looked at the importance of having a big and varied dataset with labeled data. If the dataset is not large enough we can use several data augmentation methods to increase it, like the ones used in U-Net.

5.3 Discussion

5.4 Further work

Bibliography

[1] Z. Albisser, M. Riegler, P. Halvorsen, J. Zhou, C. Griwodz, I. Balasingham, and C. Gurrin,
"Expert driven semi-supervised elucidation tool for medical endoscopic videos,"
in *Proceedings of the 6th ACM Multimedia Systems Conference*, 2015, pp. 73–76 (cit. on p. 1).

[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 (cit. on pp. 3, 9).

[3] 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. 1989 (cit. on p. 4).

[4] 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 (cit. on p. 9).

[5] 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 (cit. on p. 10).

[6] 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 (cit. on p. 10).

[7] A. C. S. Van Heel, "A New Method of transporting Optical Images without Aberrations," *Nature*, vol. 173, no. 4392, pp. 39–39, Jan. 1954 (cit. on p. 11).

[8] G. Iddan, G. Meron, A. Glukhovsky, and P. Swain, "Wireless capsule endoscopy,"
Nature, vol. 405, no. 6785, p. 417, May 2000 (cit. on p. 12).

[9] 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. 15).

[10] 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 (cit. on p. 18).

[11] P. Sadowski, "Notes on backpropagation," 2016 (cit. on p. 19).

[12] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, Jan. 1999 (cit. on p. 22).

[13] 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," *arXiv:1406.2572 [cs, math, stat]*, Jun. 2014. arXiv: `1406.2572 [cs, math, stat]` (cit. on p. 22).

[14] 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 (cit. on p. 22).

[15] 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 (cit. on p. 22).

[16] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017. arXiv: `1412.6980 [cs]` (cit. on pp. 23, 56).

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778 (cit. on p. 23).

[18] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," *arXiv:1905.11946 [cs, stat]*, Nov. 2019. arXiv: `1905.11946 [cs, stat]` (cit. on p. 24).

[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 (cit. on p. 26).

[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. 26).

[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. 2012 (cit. on p. 26).

[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. 2014 (cit. on p. 26).

[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. 26).

[24] 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 (cit. on pp. 26, 27).

[25] 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. 2019 (cit. on pp. 26, 27).

[26] 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. 2017, pp. 170–174 (cit. on p. 26).

[27] 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. 2018 (cit. on p. 26).

[28] 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 (cit. on p. 27).

[29] Bernal, J and Aymeric, H, *Gastrointestinal Image ANAlysis (GIANA) Angiodysplasia D\&L challenge*, https://endovissub2017-giana.grand-challenge.org/home/, 2017 (cit. on p. 27).

[30] 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 (cit. on p. 27).

[31] 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. 28).

[32] 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. 30).

[33] 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 (cit. on p. 31).

[34] 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. 31).

[35] D. Zhang, H. Maei, X. Wang, and Y.-F. Wang, "Deep Reinforcement Learning for Visual Object Tracking in Videos," *arXiv:1701.08936 [cs]*, Jan. 2017. arXiv: `1701.08936 [cs]` (cit. on p. 31).

[36] 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. 32, 33).

[37] J. Long, E. Shelhamer, and T. Darrell,
"Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440 (cit. on p. 32).

[38] 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. 2018 (cit. on p. 33).

[39] T. Ping-Sing and M. Shah, "Shape from shading using linear approximation,"
Image and Vision Computing, vol. 12, no. 8, pp. 487–498, Oct. 1994 (cit. on p. 33).

[40] 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. 2019 (cit. on p. 35).

[41] P. W. Wong and C. Herley, "Area based interpolation for image scaling,"
US5889895A, Mar. 1999 (cit. on p. 40).

[42] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le,
"AutoAugment: Learning Augmentation Strategies From Data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 113–123 (cit. on p. 46).

[43] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "RandAugment: Practical automated data augmentation with a reduced search space,"
arXiv:1909.13719 [cs], Nov. 2019. arXiv: `1909.13719 [cs]` (cit. on p. 46).

[44] D. Masters and C. Luschi, "Revisiting Small Batch Training for Deep Neural Networks," *arXiv:1804.07612 [cs, stat]*, Apr. 2018.
arXiv: `1804.07612 [cs, stat]` (cit. on p. 47).

[45] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *arXiv:1206.5533 [cs]*, Sep. 2012. arXiv: 1206.5533 [cs] (cit. on p. 48).

[46] 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. 2003 (cit. on p. 48).