how about that unsupervised learning?

Mathias Kirkerød

Thesis submitted for the degree of
Master of science in Informatics: Technical and Scientific Applications
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2019

how about that unsupervised learning?

Mathias Kirkerød

how about that unsupervised learning?

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

Printed: Reprosentralen, University of Oslo

Abstract

Acknowledgments

my cat, if i had one

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Cancer is today, the second leading cause of death in the world, only behind cardiovascular diseases. It is one of the leading causes of mortality worldwide, with approximately 14 million new cases in 2012 [1]. Contrary to normal cells, cancer cells are often invasive, and it will spread if not treated. In contrast to many other diseases, cancer does not need to start from a foreign entity such as a bacteria or virus, but it is often from a malfunctioning cell that starts dividing rapidly. This cell division can happen when a cell is damaged, by for instance radiation or other factors like specific proteins, or other chemicals. The result is that the cell either has damage in the DNA which contributes to abnormal cell division or the cell division itself malfunctions. In both cases the damage causes the cell to divide uncontrollably. Cancer can in some cases form without any external forces. The cell division is not always perfect, and dysfunctional cells might start a rapid division after being created. In most cases, this is not a problem, as most cells self destruct when they cannot operate. cite
The risk of getting cancer is also increased by age. As we grow older, our body gets more prone to defective cell division, and for each imperfect division, the chance of getting cancerous cells increases. Our own body is designed to detect and remove cells that are prone to divide uncontrollably. Unfortunately, this system is not perfect, and the immune system can in some cases overlook cancerous cells. In either external or internal cases, cancer is by definition this uncontrollable multiplication.
Because cancer can hit anyone, at any age, without any predispositions, it is a heavily researched area, both in Norway and the rest of the world. Despite being such a researched area, it is still one of the top causes of human death. Some types of cancer, like breast cancer, is one of the simpler forms of cancer to treat, and at this point, those kind of cancers are non-fatal in 78% of the cases in the United Kingdom [2].
Humans can get cancer in every major organ, but some types of cancer are more common than others. For instance cancer in the gastrointestinal (GI) tract is such a place, with approximately 40,000 cases each year in the UK [3]. There are around 16,000 bowel cancer deaths in the United Kingdom every year, and it is the 2nd most common

cause of cancer-related death, accounting for 10% of all cancer mortalities.
Given the global focus on cancer, research into detection and treatment is highly relevant in modern western society. Especially with detection of cancerous areas in the body, the advancement of computer-aided diagnosis (CAD) has helped significantly when it comes to early detection and localisation. In addition to the boom in computing power, machine learning has become prevalent in the past few years, and specifically, deep learning has become a tool in image and video classification both within and outside the medical domain [4]–[7]. With machine learning and CAD, researchers have now the ability to help doctors with the vital task of detecting and classifying anomalies found in medical images and videos.
Earlier projects regarding CAD have shown promising results, giving doctors new tools when looking for cancer in the GI tract.
EIR - A Medical Multimedia System for Efficient Computer Aided Diagnosis by Riegler

Michael: What is the best source to quote and cite here?

The paper on Mimir by Hicks et al. presents a system to both improve the "black box" understanding and assist in the administrative duties of writing an examination report, and in summary helping medical staff with CAD [8].

more

The work done by Hicks et al. shows that deep learning has great applications when it comes to computer aided diagnosis, but there is little work into generalising the methods to work on new data. Another problem with deep learning is that the model overfits to the dataset, meaning it learns the internal bias associated with the data provided.
In this thesis, we explore these topics. We will look at methods into how to counteract overfitting of medical data, as well as methods in to help to generalise models to better adapt to new unseen datasets.

Pa slutten her, fra EIR og ned, bor du ha med en del mer detaljer og eksempler pa hva som fortsatt er problemer og utfordringer og da sarlig peke pa det som gir motivasjon for din problemstilling - mao du må si litt mer om datasets, hva er problemer osv.

1.2 Problem statement

Based on the motivation presented in the previous section, we believe that we still have room for improvement when making CAD. Based on the previous work done in the research area [9] [8], we present the following two hypothesises as a basis for the thesis:

Hypothesis H_0: When classifying images, we will get the best result when we have images with the least amount of sparse information[1]. Hence, by removing areas with sparse information, we will see an increase in classification performance compared to not removing the areas.

[1]Sparse information in the setting of this thesis is images where there are no relevant pixels for the classification, and the area has little to no entropy. A specific example for us is the area around images with RGB values of 0

Hypothesis H_0 talks about how black or white areas in pictures might create unwanted classification errors, and that by removing those areas might improve the results of the classification. We also mention low entropy areas as part of the hypothesis, though this needs to be tested individually.
and

Hypothesis H_1: When training a classifier, we will get a higher **chance** of generalisation of our results when removing the dataset-specific artefacts[2] compared to not removing artefacts.

Hypothesis H_1 talks about pixels that are not originating from the original image, or pixels that do not represent the real sample. We believe that the removal of these dataset specific artefacts, the machine learning algorithm does not learn to take these areas into account when classifying images, and subsequently learns the real features for the dataset, instead of the artificial features created by the artefacts.
Our objective in this thesis is to explore the following two hypothesises to show their validity.
The hypotheses raise the following questions which we will address:

1. Can the process of redrawing an area with a new more relevant information, we define it as inpainting, of sparse areas in datasets help with training and classification performed by machine learning? If so, how detailed should the inpainting be, and how much should be inpainted?

2. Can inpainting of dataset-specific artefacts help with classification of previously unseen data done by machine learning? If so, how detailed should the inpainting be, and how much should be inpainted?

1.3 Scope and Limitations

Based on the hypothesises in section 1.2, the scope of the thesis is to check their validity, both each on their own and their validity together with each other. Both our scope and the problem statement is based on medical images taken from the GI tract, and the goal is to see if the hypothesises can, in the end, help with medical image classification. We want to look at the problem statements on three different datasets, all with different attributes, and three forms of inpainting. For each of the datasets, we test all three combinations of inpainting with two different inpainting algorithms. For the six created datasets plus the base dataset, we run, two different pretrained transfer learning networks to see the success of the newly created dataset. In addition to doing this at the size 256×256 px we also do all the tests above at double resolution to the validity at larger image sizes.

[2]Artefacts in the setting of this thesis is parts of images where there are components of the image not containing ´´true pixels" from the real world. A specific example for us is any overlay put on the medical images, or for instance oversaturated pixels or lens flares

In total, we make fourteen datasets, and we test the first seven a total of 72 times, and the last seven 36 times. In addition to the base case, we do 109 total tests to check the validity of inpainting.

1.4 Research Method

For this thesis, we have decided to use the Association for Computing Machinery's (ACM) methodology for our research. The article "Computing as a discipline" presents the discipline of computing into three main categories [10].

1.4.1 Theory

The ´´theory" part of the article is rooted in mathematics and describes the development of a theory. The article describes the four steps of the theory phase as (1) characterise objects of study (definition), (2) hypothesise possible relationships among them (theorem), (3) determine whether the relationships are true (proof), and (4) interpret results.

In this thesis, we touch upon the theory behind machine learning, more specifically deep learning and convolutional neural networks. We identify the problems regarding overfitting and the lack of generalisability.

1.4.2 Abstraction

The "abstraction" part of the article is rooted in the experimental scientific method and relates to the investigation of the hypothesis. The four stages the investigation is: (1) form a hypothesis, (2) construct a model and make a prediction, (3) design an experiment and collect data, (4) analyse results.

The experiments done in this thesis falls under this category. Also, we have the hypothesises (H_0 & H_1) and methodology as part of the abstraction. Based on the hypothesises presented, we created tests to check their validity, of which we were able to either verify or refute the theory presented.

1.4.3 Design

The third part, "design", is rooted in engineering and consists of four steps followed in the construction of a system to solve the given problem: (1) state requirements, (2) state specifications, (3) design and implement the system, (4) test the system.

This category was supported by the finished system able to inpaint images to improve classification accuracy. This system was extensively used throughout the thesis to conduct a plethora of experiments.

1.5 Main contributors

Throughout this thesis we have developed a system for preprocessing data as a tool to help classification. The system can take any mask and recreate the masked area, and can be used both in the removal of dataset spessific artefacts as well as a general tool for inpainting.
In section 1.2 we outlined two main questions based on H_0 & H_1, that we are able to answer:

1. *Can the process of redrawing an area with a new more relevant information, inpainting, of sparse areas in datasets help with training and classification performed by machine learning? If so, how detailed should the inpainting be, and how much should be inpainted?*

   Maybe, but probably not. From our tests we see no noticeable difference when we train and test on the same dataset. We do see improvement for both models when testing on a new dataset,
   TODO, i do not what to write here or at the above spot..
   4.33
   4.31
   4.32
   This are the visiualised results.. I guess i need to use this to say something about the general success of things??? Would like some comments on what I need to write here.

2. *Can inpainting of dataset-specific artefacts help with classification of previously unseen data done by machine learning? If so, how detailed should the inpainting be, and how much should be inpainted?*

   Yes, the experiments show that, by inpainting
   TODO, i do not what to write here or at the above spot..
   4.33
   4.31
   4.32
   This are the visiualised results.. I guess i need to use this to say something about the general success of things??? Would like some comments on what I need to write here.

1.6 Outline

The thesis is organised as follows:

Chapter 2: Background We give more background information about the medical practice and machine learning. We talk about how modern hospitals administer

colonoscopies and give insight into how we find polyps and remove them. Here, we also present how these hospitals perform digital diagnosis. We give an introduction to machine learning and its uses, both the history and present-day applications. We will look at the most successful type of machine learning, and give a brief tour into how it works, and how it can be applied to medical data. We round off this chapter by looking at how machine learning and medical colonoscopy can work together to help with the detection of anomalies in the GI tract.

Chapter 3: Methodology We describe the methodology by presenting the work we want to do to test the hypothesises we use in the thesis. We first look into how we can solve our problems by using inpainting and go into detail into the areas we want to remove to test our problem statements. After this, we describe a system to the validity of our models, followed by technical details on the programming languages and packages used. We end the chapter by looking at the two programs we end up with to test our theories.

Chapter 4: Experiments We start by giving a review of the datasets we use to train and evaluate our model, followed by the metrics we use to describe our rate of success. We go more in detail into the six datasets we make, and the 78 total runs we take to ensure reliable results. We end this chapter by presenting the inpainting datasets and then presenting the evaluation of the datasets.

Chapter 5: Conclusion Finally, we summarise and conclude this thesis. We also present ideas and suggestions for further studies surrounding the findings in this thesis and present final remarks about the research.

Chapter 2

Background

In this chapter, we will present the background and motivation of our thesis. We start with our background in medical procedures, looking at how doctors perform colonoscopies, mainly from a gastrointestinal perspective. After this, we then look at what the objective is for the medical staff, with different anomalies in the GI tract. Then, we shift our focus to how doctors use computer-aided diagnosis (CAD) today to help with the screening.
After the discussion from a medical point of view, we shift our focus to machine learning and give a brief introduction to different machine learning methods. We will look at how machines can "learn" and discuss different areas we can use and the areas we are using machine learning today. We will then go in depth into the examples and look at the most common machine learning models. We will look at the most state of the art form of machine learning, namely neural networks, and look into the most frequent use case of this type of algorithm. With this in mind, we will look at neural networks, especially convolutional neural networks, and how they work. Lastly, we will combine the need for computer-aided diagnosis with machine learning, looking at where previous models fall short and why that is the case.

2.1 The Medical Background

As we recall from the introduction, our motivation for this thesis is in the improvement of medical diagnosis with the help of machine learning. We talked about existing methods for screening developed to work with datasets retried from the medical professionals. Before we look into the CAD procedures that are in use today, we need to go more in-depth into the capture of the medical images. Medical images can be classified in to multiple categories,

2.1.1 Endoscopy and Colonoscopy

Gastrointestinal endoscopies are one of the most common medical examinations where the mucosa of the patient is visualised via a camera through the whole of the GI

tract [11]. Today the medical staff working with the visual screening of the intestinal tract use primarily two different methods: colonoscopy and gastroscopy. Colonoscopy is the practice of inserting a colonoscope into the rectum and moving through the large intestine towards the small intestine. Gastroscopy is the practice of inserting a camera via the mouth to get a visualisation through the stomach.
The endoscopic tool used for this visualisation is made out of a flexible tube with a charged coupled device (CCD) working as a camera at the end. In addition to the light sensing chip, there is also an optical fibre to transport light to the camera. At the other end, the colonoscope is connected to a device that records the video, and a light source for the optical fibre. The video from the CCD is shown live for the medical staff for the doctors to analyse [12]. As Figure 2.1[1] shows, the colonoscope can either be inserted into the anus, and traverse up the colon, or it can be inserted through the stomach, and traverse through the small intestine.

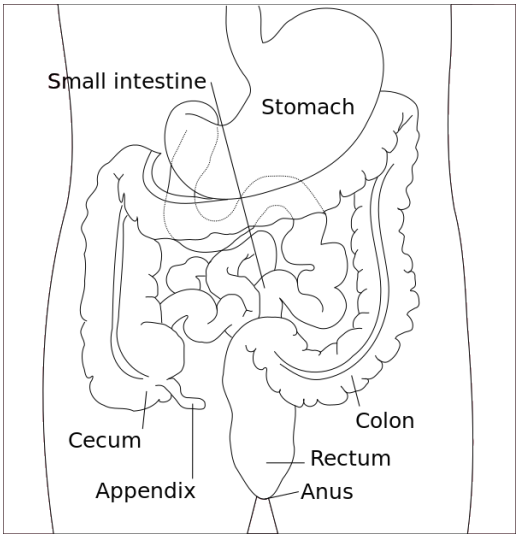


Figure 2.1: Diagram of the human GI tract

2.1.2 Summary

We have now looked at the medical perspective of colonoscopy. We see that the practice has not changed since the introduction of the colonoscope and that the best chance of detection is manually looking at images by professional medical staff. We have also taken a look into the automatic detection methods being experimented with today.

2.2 CAD - Computer aided diagnosis

During the colonoscopy, the medical staff uses digital imaging for the inspection.

[1]From `https://commons.wikimedia.org/wiki/File:Stomach_colon_rectum_diagram-en.svg`.

EIR

- Already used methods
- medico and other work
- overfitting and artefacts

2.3 Machine Learning

We have looked at the challenges that the medical staff has when it comes to detecting polyps, and how it is solved today. However, to truly understand how automated systems like Mirmir [13] works, we need to look at how Machine learning helps with the detection of the anomalies the medical staff are searching for.
Machine learning is a broad term, but we summarise it with the quote from Tom M. Mitchell in his machine learning book from 1997 [14]:

is this going to become talked about at this point

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with the experience E.

A few things of note from this quote is the variables mentioned in the quote. Experience E is the stored knowledge the program has gotten. It is in most cases just numbers used to approximate a solution given an input, to try to get it as close to the right answer. This approximation is made for every task T until we are happy with the result. Lastly, to tell how well our program performs we need a measure P that tells us how far away from the desired output we got.
From this, we see that the goal of machine learning is to improve some performance P with experience. This behaviour is a mimicry of human learning, where we as humans need to practice on a task to improve on it. As the amount of experience increase, both for us and the machine, the performance of the task becomes better and better. We have seen that machine learning algorithms have become superior at solving some human tasks [6], [15], [16], given enough time and computing power. Projects like Aplha GO and OpenAI Five shows that, given the right amount and type of data, our machine learning algorithms can solve the same problems humans solve, given both the right training environment and enough time.
Until now, we have talked about machine learning in broad terms at this point. We have drawn a parallel between how humans learn, and how machines gather experience. Now, we will look into the most popular machine learning techniques, and show the machine learning algorithms store the experience gathered.

2.3.1 Machine learning types

With a basis in the quote from the machine learning book from Tom Mitchell [14], we have a broad definition of what machine learning can be. As long as we have a model trying to complete a task based on previous experience, it can be called machine learning. Though just like for humans, machine learning has multiple ways to gather and retain information. Figure 2.2 shows a chart over the three most common categories within the field of machine learning.

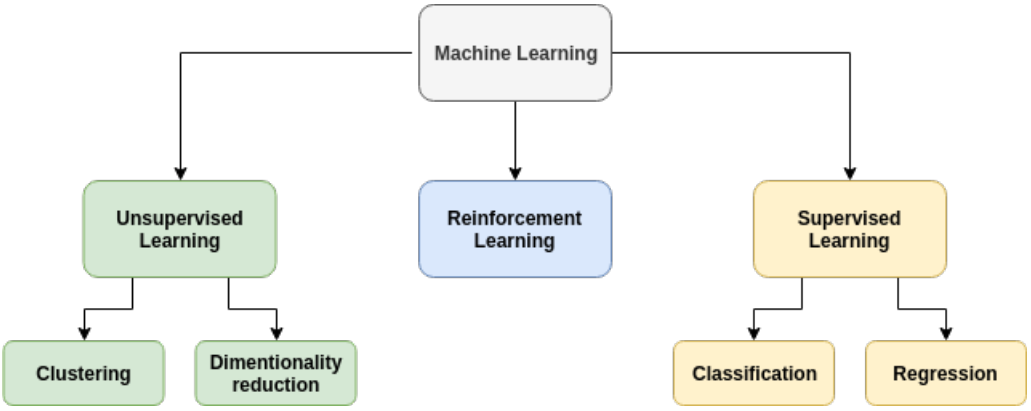


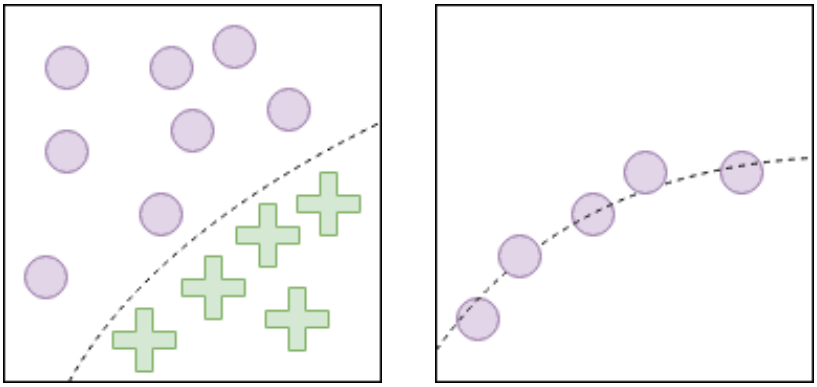
Figure 2.2: The three main types of machine learning and their most common subtypes

We have three subcategories of machine learning: Supervised, Unsupervised, and Reinforcement learning. We will now present the three methods briefly. Then we will look at famous examples that helped shape machine learning types and algorithms we are using today.

Supervised machine learning

Supervised machine learning concerns the iterative process of labelling data based on previously labelled data. Supervised machine learning functions have the objective of, given an input-output pair, approximate the input to be as close to the output. [17] Alternatively, in simplistic terms, given an input x, produce an answer as close as possible to the output y. A supervised algorithm analyses the training data and produces an inferred function, which can be used to map new data entries. The two most common types of Supervised learning is shown in Figure 2.3.
Examples of supervised tasks are to recognise handwritten numbers, or differentiate between different car models. We consider a task supervised if the images come with the correct label in the data set. A more straightforward classification assignment is binary classification, where the target is (often) yes or no. Examples for binary classification is if an email is spam or not, is a car Norwegian or International. In the last example, the classification changes from binary to multi-class if we sort the cars on every nationality, and not just Norwegian/non-Norwegian. Another type concerning

Her må det vere et storre skille mellom trening og evaluering. Tenk paa om det er smart aa omskrive dette



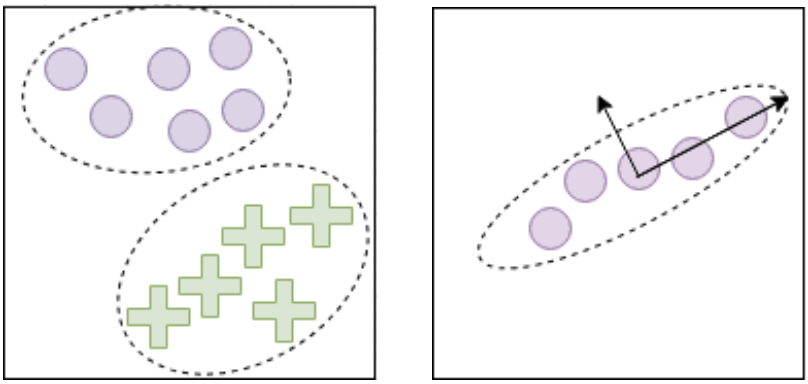
(a) Supervised classification (b) Supervised regression

Figure 2.3: Examples of the two most common use cases for supervised learning

supervised learning tasks is regression. Regression is the act of prediction given prior data. Examples of regression are the prediction of stock prices, to estimating house prices, to predicting the weather.

Unsupervised machine learning

Unsupervised learning is the act of training without any supervision, in the sense that we do not give the algorithm the next output to the given input as we do in supervised learning. Figure 2.4 shows a simplified concept of unsupervised learning. Since we do not have categorised data in unsupervised learning, we often want the algorithms to find some underlying structure of the data, rather than classifying it. Types of unsupervised learning can, for instance, be clustering, the act of sorting data based on similarity or it can be dimensionality reduction, the act of simplifying or compress the data.
An example of this can be if we want to sort plants based on similarity, or we are detecting anomalies in a dataset. We often use unsupervised learning for principal component analysis (PCA) [18] or other dimensionality reduction methods. A third method to used unsupervised learning is the adversarial route, where we use machine learning to make similar looking data to the original data set.



(a) Unsupervised clustering

(b) Unsupervised dimensionality reduction

Figure 2.4: Examples of the two most common use cases for unsupervised learning

Reinforcement learning

Reinforcement learning is the area in machine learning that is concerned with how a software agent should take actions. The agent bases its actions on the environment, and it is influenced by the objective to get the maximum reward, as illustrated in Figure 2.5. It is closely influenced by behaviourism, with the fact that the software agent wants to maximise the reward obtained continually.
Successful types of reinforcement learning alogrithms are, for instance, Deep Recurrent Q-Learning [19], State–action–reward–state–action (SARSA) [20].

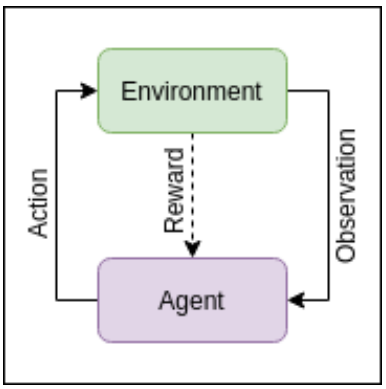


Figure 2.5: The basic structure of reinforcement learning

Well known machine learning algorithms

Now that we have a basis on the three types of machine learning we can go into more detail on the most successful types of machine learning used both now and in the past.

Machine learning was coined as a term as early as in the 1950s by Alan Turing. The first concept was related to the Turing machine and is now considered a foreshadowing of genetic algorithms. [21]

Forward to 1967, the Nearest Neighbour algorithm [22] was created, which is considered the start of basic pattern recognition. The Nearest neighbour algorithm is a type of machine learning that requires no prior training, making it fast and deterministic.

Another early adoption of machine learning was in the form of regression. Regression is the statistical concept of estimating the relationships among variables. It is in heavy use today, and one of the core concept we use machine learning for today. Legendre first used regression in 1805 with his method of least squares. The least square methods were initially being done by hand, and it was at the time one best models, backed by math, to estimate the relationship between an input and a subsequent output. Today, regression analysis is widely used in statistics and informatics, and there is a significant overlap between the two research fields. While often we can make analytical models when working with a dataset with few variables, machine learning has the possibility of making much more complex models.

cite

find out how much is needed to write here

A newer and applied form of supervised machine learning is the support vector machine. The original support vector machine was invented by Vladimir N. Vapnik and Alexey Ya Chervonenkis in 1963. In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested ways to make the support vector machine work in multiclass examples by using kernels. A support vector machine is a form of classifier where the goal is to divide two datasets by a line that is just between the data.

cite

Here is an example of a famous reinforcement learning thing.

Summary

We have now discussed the general structure of the three types of machine learning. For each of the three methods, we have looked at designs that utilise their form of learning, showing their real-world applications. We have also looked into some successful algorithms through the ages, highlighting innovations that helped form our vast library of methods we can use to tackle statistical problems we meet. We will now first go more in-depth into how a general machine learning algorithm works, giving a rundown on how a simple algorithm works from start to end. After this, we look into more advanced examples of modern machine learning algothroms that forms a basis in this thesis.

2.3.2 The basic concept of machine learning

One of the easiest to understand tasks in machine learning is the process of regression. As stated earlier, regression is a process of approximation given prior input. We start with one of the simplest forms of approximation, namely linear approximation. In linear approximation, we are interested in finding the function that best defines our

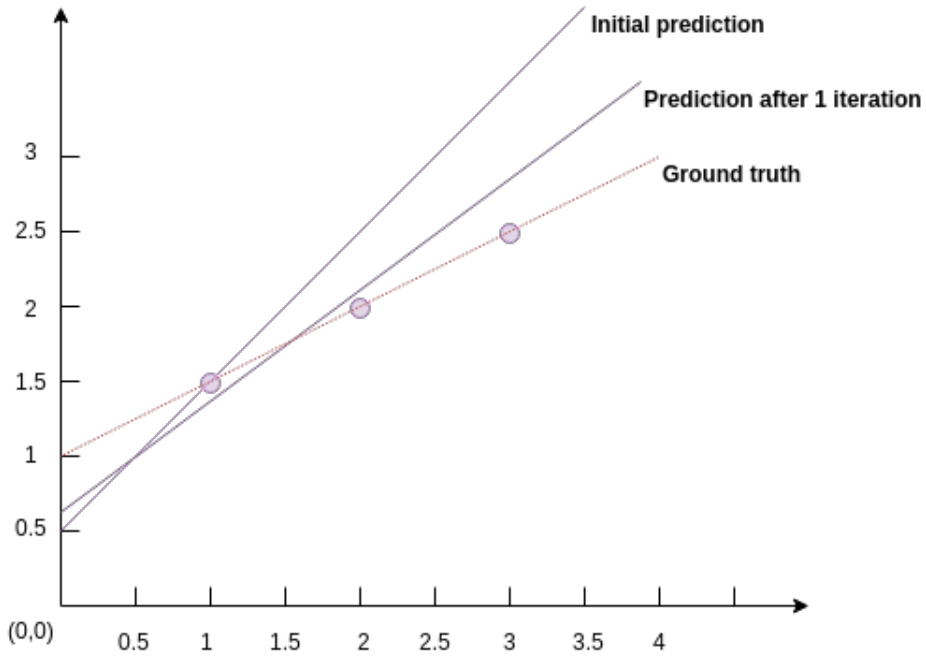


Figure 2.6: Example of linear regression. Here the red line is the best approximation of a y value, given an x value.

data using only a polynomial of the first degree. First, we recall that a first-order function is always on the form

$$y = ax + b \tag{2.1}$$

Where x is input, y is output and the constants a and b defines the function.
Figure 2.6 shows an ideal example of linear regression with the model we are solving. Here we approximate the values of our model with the straight line defined by choosing the right slope (a) and the right constant (b). With the knowledge of math, we look into how to do it computationally with the help of machine learning.
We can recall from the quote 2.3 by [14] that we gain experience E by doing a task T. In our example we choose to store our experience as its done in equation 2.2.

$$y = W^{(0)}x + W^{(1)} \tag{2.2}$$

Here, like before, our output is y, and our input is x. We have replaced "a" and "b" with new placeholders $W^{(0)}$ and $W^{(1)}$. In this example $W^{(0)}$ and $W^{(1)}$ are constants, but in more complex examples, W would be matrices. Now our goal is to, given a task T, gain experience E and store it in $W^{(0)}$ and $W^{(1)}$. With our values for $W^{(0)}$ and $W^{(1)}$ we want the best performance P. The best performance here is defined as getting the smallest difference between the predicted output data and the actual output data.
The most prominent way of calculating this error is to use the mean square error

between the predicted and actual output of the data.

$$MSE = \frac{1}{2m}\sum_i (\hat{y} - y)_i^2 \tag{2.3}$$

Where m is the number of samples, y is the real output, and $\hat{y}$ is the predicted output. The 2 in the denominator is just a constant to make the derivation of the formula easier. From this, we can intuitively see that the error tends towards 0 when $\hat{y}$=y. We can also note, because of the squaring in the formula, that the error is only based on L2[2] distance between $\hat{y}$ and y.
Now that we have an error, we need a way to improve it. At this point, we have a way to store experience E (in $W^{(0)}$ and $W^{(1)}$), measure performance P (in the MSE), and we have tasks T (in the form of input-output pairs). Given an input-output pair, we will now look at how to use machine learning to better approximate the next input-output pair.

Lets start with:

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, y = \begin{bmatrix} 1.5 \\ 2 \\ 2.5 \end{bmatrix} \tag{2.4}$$

As we can discern from this formula, and by looking at the Figure 2.6, our ideal model would lie at $y = 0.5x + 1$ as marked with the dotted red line. This means that our ideal weights would be $W^{(0)} = 0.5$ and $W^{(1)} = 1$. In our initial formula, we set the the weights $W^{(0)} = 1$ and $W^{(1)} = 0.5$. To get the ideal formula, we would like $W^{(0)}$ decrease by a half and $W^{(1)}$ to increase by a half. Using the formula 2.2 with the input x values we can calculate $\hat{y}$, given our weights, to be:

$$\hat{y} = \begin{bmatrix} 1.5 \\ 2.5 \\ 3.5 \end{bmatrix} \tag{2.5}$$

We can now calculate the performance by applying an error function. Using the MSE formula 2.3, the loss $\mathcal{L}$ is:

$$\mathcal{L} = \frac{1}{2*3}\left((1.5-1.5)^2 + (2.5-2)^2 + (3.5-2.5)^2\right) = 0.20 \tag{2.6}$$

With our new found error, we need a way to use this to update our weights $W^{(0)}$ and $W^{(1)}$ to get a better estimate.
The most common way to update our weights is to use gradient descent. Gradient descent is a first order iterative optimisation algorithm for finding the minimum of a function [23]. In our case, we are looking for the minimum value of the MSE function. Gradient descent is defined as (simplified for our example):

$$a_{n+1} = a_n - \gamma \nabla F(a_n) \tag{2.7}$$

[2]The L2 distance is the Euclidean distance between two points in a plane. L1 distance, often called Taxicab distance, taking the absolute value instead of the square root.

Where ∇F is the derivative of the function in question, a is the input at step n, and γ is a learning rate set to a small number. The learning rate is an essential part of the calculation, as without it we would often calculate the new weights too extreme for our problem. By introducing a learning rate, we take small, more controlled steps in the right direction.
Derivating 2.3 and using a learning rate of 0.2 we get the following.

$$\begin{aligned}\nabla F_{W^{(0)}} &= \frac{d}{d_{W^{(0)}}} \frac{1}{2m} \sum_i (\hat{y} - y)_i^2 \\ &= \frac{1}{m} \sum_i (\hat{y} - y)_i \cdot x_i \\ &= \frac{1}{m} \sum_i (w^{(0)} \cdot x + w^{(1)} - y)_i \cdot x_i\end{aligned} \tag{2.8}$$

$$\begin{aligned}\nabla F_{W^{(1)}} &= \frac{d}{d_{W^{(1)}}} \frac{1}{2m} \sum_i (\hat{y} - y)_i^2 \\ &= \frac{1}{m} \sum_i (\hat{y} - y)_i \\ &= \frac{1}{m} \sum_i (w^{(0)} \cdot x + w^{(1)} - y)_i\end{aligned} \tag{2.9}$$

Inserting 2.8 and 2.9 in to 2.7 gives us the two following formulas for $W^{(0)}$ and $W^{(1)}$

$$\begin{aligned}W^{(0)} &= W^{(0)} - \gamma \frac{1}{m} \sum_i (w^{(0)} \cdot x_i + w^{(1)} - y)_i \cdot x_i \\ &= 1 - 0.2 \cdot \frac{1}{3} \sum (0 + 1 + 3) \\ &= 0.733\end{aligned} \tag{2.10}$$

$$\begin{aligned}W^{(1)} &= W^{(1)} - \gamma \frac{1}{m} \sum_i (w^{(0)} \cdot x_i + w^{(1)} - y)_i \\ &= 0.5 - 0.2 \cdot \frac{1}{3} \sum (0 - 0.5 - 1) \\ &= 0.6\end{aligned} \tag{2.11}$$

The new weights gives us $\hat{y}$ to be:

$$\hat{y} = \begin{bmatrix} 1.25 \\ 1.85 \\ 2.45 \end{bmatrix} \tag{2.12}$$

This gives us the loss:

$$\mathcal{L} = \frac{1}{2*3}\left((1.33-1.5)^2 + (2.06-2)^2 + (2.79-2.5)^2\right) = 0.019 \tag{2.13}$$

After one iteration of gradient descent, we see the weights becoming closer to the desired result. With more iterations the closer the weights will get to the point that gives the smallest error, as long as the learning rate is small enough. We looked at an example using the formula for a linear approximation. In the real world, there are only a handful of problems that we solve by making a linear approximation. We will now look into more advanced types of approximations made with machine learning.

2.4 Neural Networks

We have looked at different types of machine learning, and we have gone in depth into how a linear regression model works. In this section, we want to get further insight into how we can make more complex models, and we will look into the most popular method for machine learning, namely neural networks [24]. After the rundown on how neural networks are built up and how they operate, we will look into convolutional neural networks. In the end, we will look at successful networks, mainly made for image generation and classification which is our target challenge in the medical domain.

2.4.1 The perceptron

To explain how a neural net operates, we first need to look at the most fundamental structure present in every type of neural network, namely the perceptron. Frank Rosenblatt introduced the first perceptron in 1957 as an attempt to mimic the human neuron [25].
Figure 2.7[3] shows how a human neuron looks like, and in which direction the signal goes. Each neuron is connected to multiple other neurons by connecting the dendrites to other neighbouring neurons forming a pathway for the electrical signals to flow. When a signal is sent, the dendrites register the signal and sends it through the axon out to the axon terminal. At the axon terminal, other neurons pick up the electrical signal and pass it through their axon. This flow of electricity is the fundamental way different part of our brain communicates, and the different pathways the signals can take represent how we learn. The original idea behind the perceptron and this branch of machine learning is to mimic this process of making pathways throughout the network as a way to learn from experience.
With the biological neuron as a reference point, we can now look more in-depth into the mathematical equivalent. Figure 2.8 shows the equivalent in the realm of computer science, having the same flow from the input to the output. The perceptron does,

[3]From `https://commons.wikimedia.org/wiki/File:Neuron.svg`

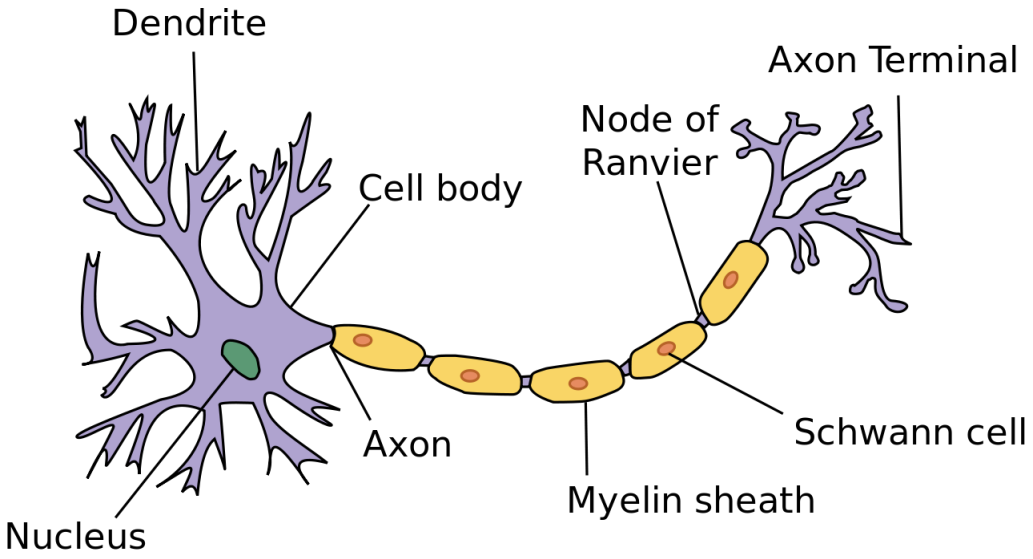


Figure 2.7: Image of a simplified neuron

however, not work with electrical signals. Instead, it works with numbers representing this signal. This abstraction gives the perceptron the ability to have arbitrary high values, as well as negative ones. We multiply the input signals to the perceptron with an associated weight. In biological terms, this weight is equivalent to the strength of the connection between the two neurons trying to communicate. We sum together the weighted inputs and apply a threshold function. In the first perceptron, the internal function were:

$$f_{out} = \begin{cases} 1 \ if \ \mathbf{W} \cdot \mathbf{x} + b > 0, \\ 0 \ otherwise \end{cases} \tag{2.14}$$

where $\mathbf{x}$ is the input, $\mathbf{W}$ is a vector of real-valued weights, $\mathbf{w} \cdot \mathbf{x}$ is the dot product $\sum_{i=1}^{m} w_i x_i$, where m is the number of inputs to the perceptron, and b a constant bias.
The general formula of the perceptron is unchanged, though we have moved away from the "0-1 output" perceptron in favour for more complex output functions like the sigmoid in 1989 [26], and the ReLu, and tanh in the 2000s.

cite

$$f_{out} = max \begin{cases} \mathbf{w} \cdot \mathbf{x} + b \\ 0 \end{cases} \tag{2.15}$$

The typical ReLu preceptron.

2.4.2 Feed forward and backpropagation through a perceptron

The general concept of the learning process is similar to the one we presented in section 2.3.2. To better understand the function of a perceptron we will explain the same steps as we saw in the linear regression example, only for our perceptron.

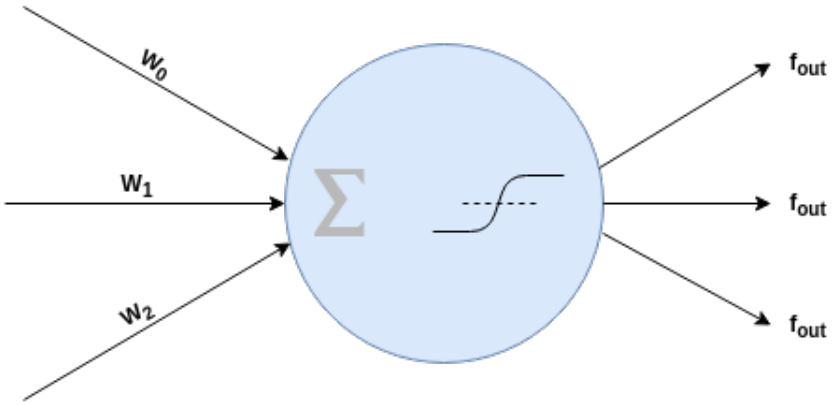


Figure 2.8: Simple perceptron that sends out an output that is the tanh of the the sum of the inputs

First our perceptron gets signals $x_{(i,0)} - x_{(i,n)}$ where n is the number of inputs to the perceptron (For instance three in Figure 2.8). The signals received is the input in the same way as we received an input an array in equation 2.4. For each of the input values, $x_{(i,0)} - x_{(i,n)}$, we multiply it by a weight $W^{(0,0)}$ - $W^{(0,n)}$. Here, in contrast to the linear regression example, every input has its own weight. After the weight multiplication, we sum the result to a scalar. We are now almost ready to give the result as an output, but to prevent the perceptron of only being able to solve linear problems when connection multiple perceptrons we need to use an activation function. This activation function can be ReLu as in equation 2.15, or something like tanh or Sigmoid shown in Figure 2.11. Note that we do not use the threshold function in equation 2.14 as an activation function in modern neural networks, as it is not applicable for gradient descent.

Now that we have an output, we look at the error between the output $f_{(j,0)}$ and the expected output and apply a loss function. We can now backpropagate the error to update the weights at the start of the perceptron.

2.4.3 Multilayer perceptrons

The neural network was a proposal made by Warren McCulloch and Walter Pitts (1943) [27] created a computational model for neural networks based on mathematics and algorithms called threshold logic.

The first multilayer network at the time used backpropagation with gradient decent in the same way described in equation 2.7.

Figure 2.9 shows the basic structure of a multilayer network. This model has one hidden layer and with the standard input and output layers. In our figure, each of

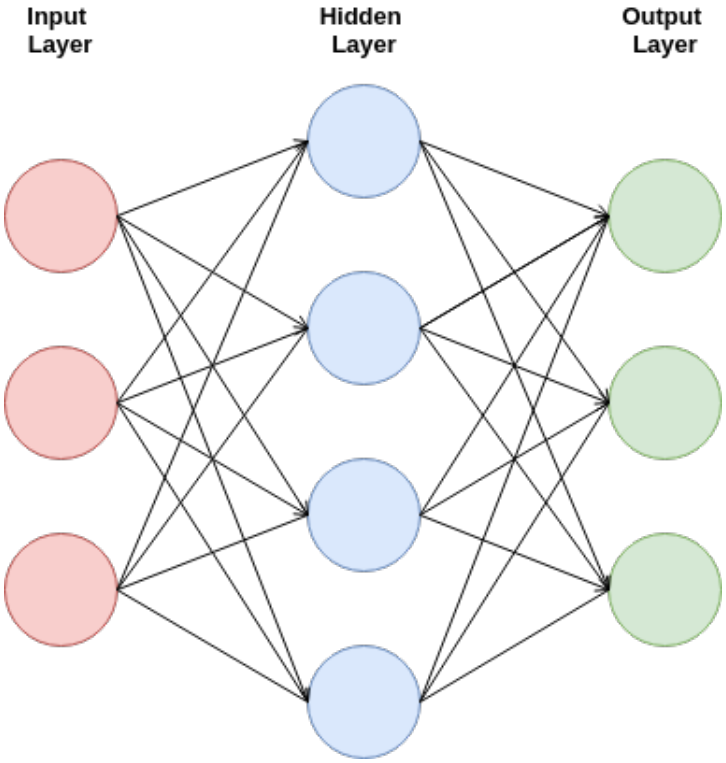


Figure 2.9: Simple illustration of a multilayer perceptron with three inputs, one hidden layer with four nodes, and one output layer with three nodes.

the nodes is a perceptron[4] as described in section 2.4.1.
The advantage of using the multilayer structure proposed by McCulloch and Pitts is the fact that we do not only have a linear boundary classification model. By using the multilayer structure the network is able to, for instance, tackle the XOR problem, not solvable by a single layer neural network.

cite

Do I want to talk about XOR?

2.4.4 Convolutional neural networks

The multilayer perceptron we have discussed is a robust tool that can learn a multitude of decision boundaries, and subsequently learn to classify thousands of different classes. As we get more data and more classes, the networks needed to solve our problem grows. We can recall from section 2.4.3 that the number of weights between neurons is $i \cdot j$ where i is the first layer and j is the connected second layer. As the number of perceptrons per layer in our neural networks increases, the total amount of storage space increases too.

$$height_i \cdot width_i \cdot channels_i \cdot height_j \cdot width_j \cdot channels_j = weights \tag{2.16}$$

[4]In reality, the input layer does not usually behave as a perceptron with an activation function. The input layer is only there to propagate the signal to the neurons further into the network.

Given an image with height and width of 128 pixels connected to the same shape in a network with a fully connected layers, the total amount of weighs per connection are:

$$(128 \cdot 128 \cdot 3) \cdot (128 \cdot 128 \cdot 3) = 2415919104 = 2.4 \cdot 10^9 \tag{2.17}$$

Given that we are working with quadratic images, the number of weights increases with a factor of four as we increase the layer height and width. As we saw in equation 2.17, the number of parameters for a relatively small image is already $2.4 \cdot 10^9$, not including the bias on top. The models we use to store our data saves the weighs as float32, which means that each weight is 4 bytes of storage. That means that the total storage for this *single* layer is:

$$4\text{b} \cdot 2.4 \cdot 10^9 = 9,66 \text{ GB} \tag{2.18}$$

Given that a standard computer usually have 8-16 GB of RAM, this one layer might not be able to load at all.
Another problem with the standard MLP is the fact that it is spatially dependent. Given an input x, the output, y, of the MLP will vary a lot if we shift the input data by one place, or if we flip the data. In some cases, this is something we want in our machine learning algorithms, but more often than not, this behaviour is not a desirable outcome. Given the downsides we have with regards to memory usage and non-spatiality in our multilayer perceptrons, we present Kunihiko Fukushima [28] solution to solve both complications. Convolutional neural networks (CNN) are the most popular form for image recognition, segmentation, and classification [4] [29]. When building a convolutional neural network we often use multiple layers stacked on opt of each other to give the network traits that a regular multilayer perceptron could not achieve. By far the most essential layer of the convolutional neural network is the convolutional layer.

Convolutional layers

Convolutional networks work with filters as opposed to perceptrons with weights assigned before and after the input in the vertices between perceptron as shown in Figure 2.10. Convolutional layers assign a weight to each position in a special filter matrix. This use of filters significantly reduces the number of weights between layers, since we now have weights that are not dependent on the input size, and only dependent on this matrix size.
The three main parameters of a convolutional layer are the number of filters, kernel size and strides. When making a convolutional layer, we start by pseudo-randomly initialising a $(F \times K \times K)$ matrix as our weights. In this matrix, F is the number of filters in the convolutional layer and K is the filter size. We can think of this as F filters of size $(K \times K)$ stacked on top of each other. When applying the convolutional layer to an input vector, we take each of the F filters and slide it over the image. For each position of the filter, we multiply the image value with the filter and sum the result. The scalar made by this multiplication is the value passed on to the next layer at that

specific point. Figure 2.10 shows this convolution process after 3 sliding operations with a (3×3) filter. When sliding across the image we have the option to take larger strides for the sliding window, in practice this striding still "see" the entire image, but the number of output connections are $\frac{1}{stride}$.

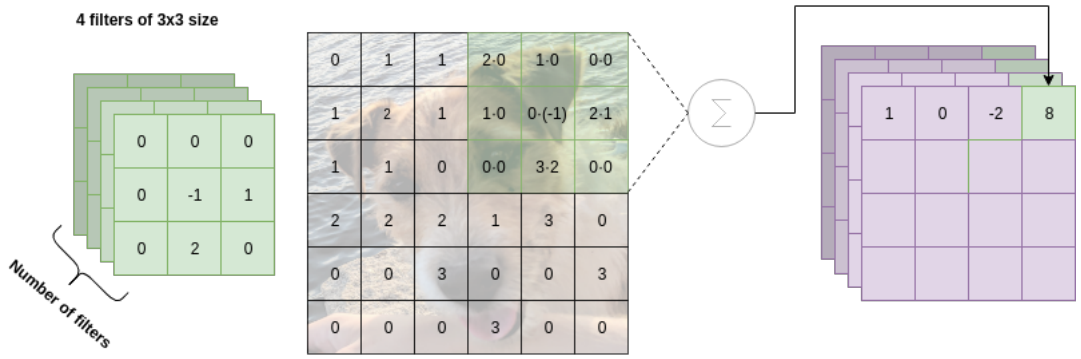


Figure 2.10: The values calculated when a convolutional filter after 4 sliding window operations, here the number of inputs does not represent how many inputs there usually is in an image

As we can see from this architecture, we can only change when weights in the filter matrix, as there are no other variables in the convolutional operation. Using this sliding window technique gives us the benefit that the filter only gathers information from the local area, and subsequently makes the convolutional operation non-spatially dependent.

Activation layers

As we discussed in section 2.4.1, in addition to summing the inputs and passing them on, we need to apply an activation function to the output. The same problem with our desire to make non-linear problems apply in the CNN model. To apply an activation layer to a CNN, we take every value in the matrix and apply the activation function separately to every data-point.
In this thesis, our CNNs used only the three activation functions shown in Figure 2.11.

Pooling

Pooling layers, often also called downsampling layers, are commonly also found in CNN's. Their primary role in the network is to reduce the spatial size of the data provided, and thus reducing the number of weights and variables used by the network. This reduction helps with reducing processing and memory costs. The two most common pooling operations are max pooling and average pooling. Both methods resemble convolutions in the way that they apply their function to a sliding filter over the data.

cite

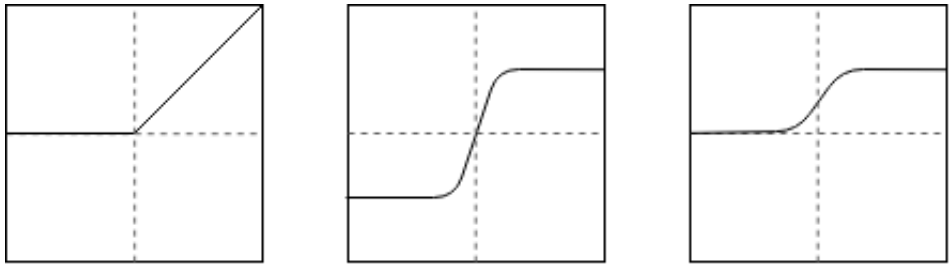


Figure 2.11: Activation functions from -2 to 2 on each axis.
From left to right: ReLu, Tanh, Sigmoid

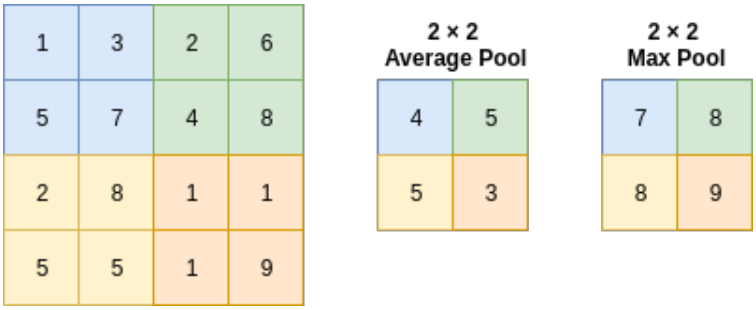


Figure 2.12: Both max and average pooling done on a 4×4 matrix

Just like in Figure 2.12 the most common pooling parameters are of filter size 2 and stride 2. Using this configuration leaves every point of the input checked once, and the resulting output size is halved. Pooling layers do generally not have any weights associated with them. This lack of weights means that the pooling layer routes the signal back without changing it during backpropagation.

unpooling

Normalisation

Training deep neural networks can often be complicated with the fact that the distribution of each layer changes during training, as the parameters of the previous layers change. In some cases, this constant shift of input values slows down the training by requiring a lower learning rate to keep the network stable. We refer to this problem as internal covariate shift, and we use normalisation as a tool to correct for it [30].
The most common way to normalise the data is to use batch normalisation. When calculating the batch normalisation, we take the following steps presented in S. Ioffe and C. Szegedy publication: "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" [30].

Input: Values of x over a mini-batch: $\mathcal{B} = \{x1...m\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{(mini-batch mean)}$$

$$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2 \qquad \text{(mini-batch variance)}$$

$$\widehat{x_i} \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}} \qquad \text{(normalise)}$$

$$y_i \leftarrow \gamma\widehat{x_i} + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{(scale and shift)}$$

In the original paper we the writers saw a significant reduction in the number of epochs needed to train the network, as well as increased accuracy.

Dropout

A major problem in machine learning is the act of overfitting the data. We say that a model is overfitting to the data when it learns the bias that comes with the dataset as shown in Figure 2.13.

more

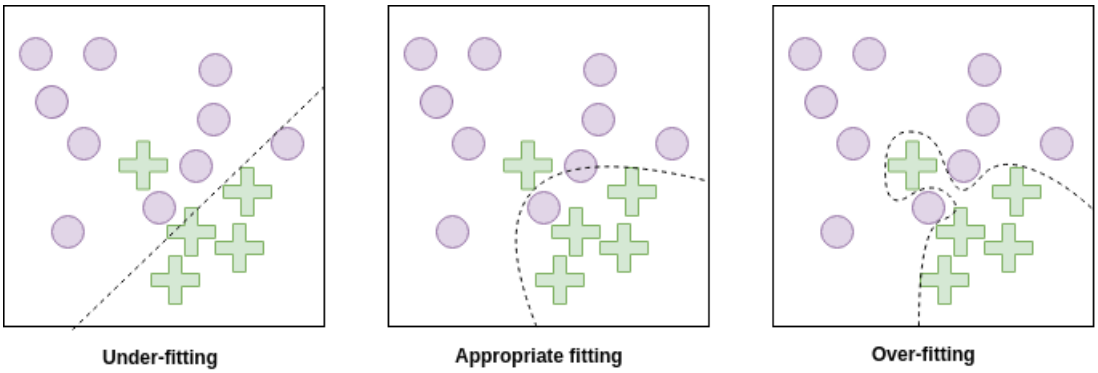


Figure 2.13: Three cases of data boundary prediction. In most cases we desire appropriate amount of fitting to our dataset to keep generalisation

Dropout, first proposed by Srivastava et al. [31], in 2014 was used to prevent overfitting of neural networks (As seen in figure 2.13). Dropout layers in the CNN will randomly cut a certain number of connections in a layer. The number of cut connections is usually between 25% and 50% of the total. The result of the use of dropout is that the network can not rely on only a few numbers of weights during training since there is a chance the input from the weight will be cut at random intervals.
This random cutting forces the network to spread out the essential weights throughout the network. Even though dropout does not explicitly change the weights, it implicitly forces the stong weights to be evenly distributed throughout the network.

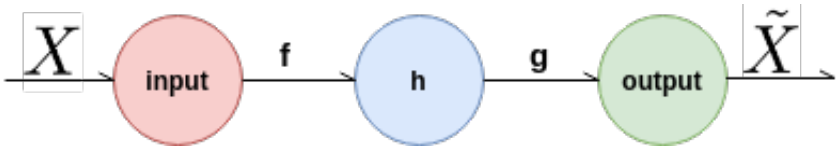


Figure 2.14: The general structure of an autoencoder, encoding **x** with **f**, then decoding **h** with **g** to an output **x**.

Another popular way to combat overfitting is to add a weight regulariser like L1 or L2 distance penalty. This penalty gives the network weights an additional penalty based on their size. As we recall, big weights in the networks have the most significant influence on the result, as each weight is multiplied by the input. If the network finds some bias hidden in the dataset, it is much harder for it to exploit this discovery. This difficulty is because, with L1 or L2 regularisation, the network is also penalised for strong weights. Dropout does, in practice, the same as an L1 or L2 regulariser. cite

2.5 Complex Neural Network models

We have now looked at the basic concept of the neural network, and a more in-depth look into the convolutional neural network and the relevant layers. Our primary focus has been from a computer vision perspective, where we have looked at image classification. To get a better insight into the act of inpainting, we need to present two other types of network, namely the Autoencoder [32] and the generative adversarial network [33].
We often base both models on the convolutional architecture, to have the option to work with image and video data. Both networks are a type of unsupervised learning, where the goal is to either recreate or construct data based on the training data.

2.5.1 Autoencoders

An autoencoder is a neural network that is trained to recreate a copy of the input given. As with the networks explained at this point, it has an input, some number of hidden layers and an output. The network can be considered to have two internal structures, the encoder (denoted as $h = f(x)$) and decoder (denoted as $r = g(h)$) shown in figure 2.14. The general goal of the autoencoder is to recreate the input $g(f(x)) = x$, but in the real world, we often add restrictions to make the autoencoder unable to copy the input perfectly. Because of this restriction, the autoencoder is forced to prioritise particular properties of the data, and hence it might learn underlying features that define the dataset.
Traditionally, autoencoders were used for tasks like feature learning and dimensionality reduction, though with the rise in computing power, autoencoders are now in use at the forefront of generative modelling.

To use an autoencoder for generative modelling, we need to set an objective for autoencoder to achieve. A when generating new images we can propose regulisers to the latent space[5] between the encoder and the decoder. The reguliser might be that the output of the latent space layer should follow a Gaussian noise pattern or another easily replicable pattern. This type of autoencoder is called a variational autoencoder. Variational autoencoders have the advantage of being able to create new completely unseen data if Gaussian noise is inputted at the latent space, effectively cutting out the encoder and supplying random noise instead. Another way to use an autoencoder for generative modelling is by masking areas in the input data the model used to train on, and give the unmasked data as the target output. Figure 2.15 shows this concept of an inpainting Autoencoder. The act of inpainting forces the autoencoder to learn the underlying features of the inpainted area to make a prediction.

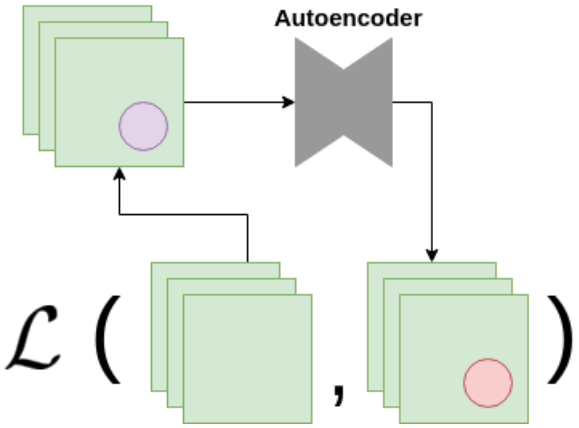


Figure 2.15: Autoencoder where the goal is to inpaint the masked area shown in the original dataset as a purple circle, and as a red circle generated by the autoencoder.

As with supervised classifiers, autoencoders use gradient descent to optimise the result, as the model work to recreate the input $\mathbf{x}$ from out output $\widetilde{\mathbf{x}}$
We achieve this by minimising the loss function:

$$\mathcal{L}(\mathbf{x}, g(f(\mathbf{x}))) \tag{2.19}$$

As this is a problem where the goal is to minimise error, autoencoders often use MSE or mean absolute error (MAE) as the optimiser for the gradient descent. As we noted earlier, Figure 2.15 show a visual example of equation 2.19 when used with inpainting.

[5]We often call the area between the encoder and the decoder the latent space. Here the information is at its maximum compression.

2.5.2 Advaserial neural networks

As an alternative to using autoencoders for generative modelling, we introduce the generative adversarial network (GAN) proposed by Goodfellow et al. [33].
As shown in Figure 2.16, the GAN consist of two components: a generative model G that captures the data distribution, and a discriminator model D that estimates the probability that the sample from G is fake, and does not come from the training data. The training procedure for the generator is to maximise the probability of the discriminator to make mistakes. The generator-discriminator network resembles a two-player game where both parties want to "win" over the other. As we are using multilayer perceptrons in our network, we can use backpropagation to train both the generator and discriminator.
In mathematical terms: The generator produces samples $x = g(z; \theta^{(g)})$, where g is the network given the weights θ. Then the discriminator network predicts if a sample is drawn from the dataset or the generator. More specific, it gives a probably given by $d(x; \theta^{(d)})$, and determines if x is from the generator or the data-set. Since we have two networks competing against each other we can look at this as a Zero-sum game with the generators payoff is determined by $v(\theta^{(g)}, \theta^{(d)})$, and the discriminators payoff is determined by $-v(\theta^{(g)}, \theta^{(d)})$.
Where v is a function that is determined by both the success rate of the discriminator and the generator, the most commonly used is:

$$v(\theta^{(g)}, \theta^{(d)}) = \mathbb{E}_{x \sim p_{data}} \log d(x) + \mathbb{E}_{x \sim p_{model}} \log (1 - d(x)) \tag{2.20}$$

as derived from [33].

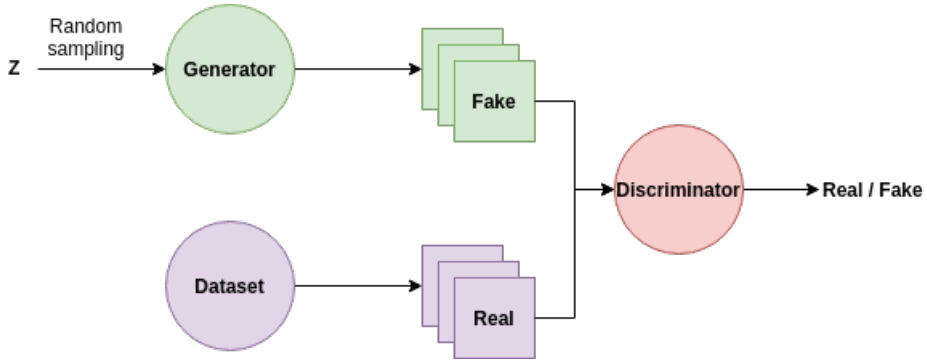


Figure 2.16: The basic concept of the generative adversarial network

2.5.3 Transfer learning

Neural networks are prone to overfitting when training on a small dataset or over multiple iterations. We often have problems with the fact that we need to train a network for a long time for it to learn the features associated with the data, though

we often learn the bias in the dataset either in addition or instead of the features that define the dataset.
The most obvious solution is to gather more training data to prevent this overfitting, or to use regulisers to force the network to diversify the layer weights, as proposed in section 2.4.4.
In this thesis, we have chosen to use transfer learning to address this problem. Transfer learning is the process of using knowledge from similar problems to help solve the current goal. Let us say we want to classify different wolf breeds, without the necessary amount of data. We can use our knowledge about dog breeds to help the classification, by using a pretrained model as a base for training. Since both wolf and dogs share many of the same features, like four legs and fur, the model does not need to learn those features from scratch. Similarly, we can use general models to help solve our specific problem.

2.6 The problem at hand / Summary

We started the background chapter by looking at the medical procedure associated with a colonoscopy, and we went in-depth into how the medical staff work with digital equipment. We got some insight into the time and complexity invested in finding polyps when doing the said colonoscopy. We then looked at systems for computer-aided diagnosis and the prevalent systems that is in use today to help medical staff with polyp detection. From this assessment, we saw that the polyp detection rate in datasets like Kvasir was relatively high, but the model overfitting was often the cause of the high-grade results. Though the advances in medical image diagnosis, the ability for models to generalise to other areas has not come far enough. We then took a close look at machine learning, from the building blocks in the perceptron to the complex pretrained neural networks used today in modern algorithms.
As we noted in chapter 2.2 we, in modern medical diagnosis systems, lack the ability to adapt to new datasets from a different distribution than the training set. We have a strong belief that this is related to the fact that we overfit the data on the training set and the fact that we learn dataset-specific features not present in other unknown datasets.
Hicks et al. in the paper "find paper" goes in-depth into the problem of overfitting with regards to the problem with dataset specific features. We can see from this paper that by removing dataset specific artefacts, the classification score increased when evaluating the model on previously unseen data.
Proglov et al. in the paper "in the paper" shows similar results when training on the Kvasir dataset and evaluating the data on different datasets.
In this thesis, we want to continue to explore this problem of lack of generalisation and find ways to improve models in that manner.

Chapter 3

Methodology

With our background in both medical background and machine learning, we can now look at how we want to solve the problems associated with setting up a system for medical diagnosis. We will first get a birds-eye view of the objective of this thesis, looking at the hypothesises behind this thesis, and take a look into how we can test the proposed hypothesises. We look at the proposed program setup to test the hypothesises both for classification and generation. Then we look at the language and packages suitable for this project. We go in-depth into the reasoning behind why we chose the tools and packages that became the foundation of the programs.

3.1 Bird's eye view (Chapter possibly removed when written)

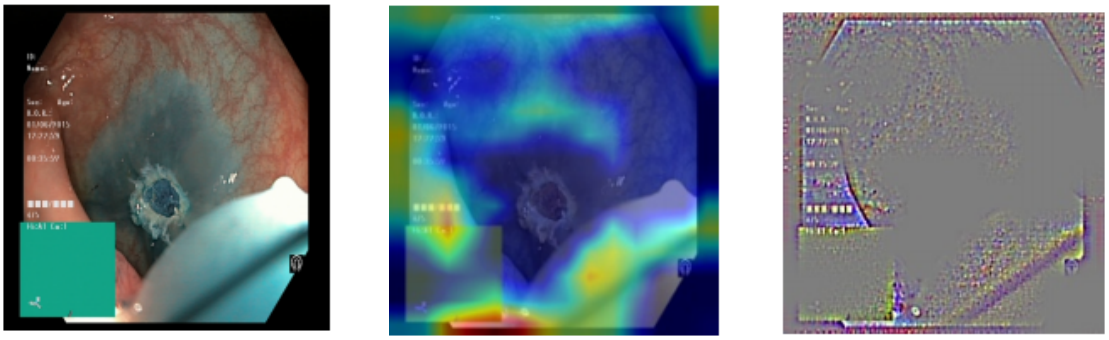
In the summary of the background chapter, we looked at two articles published by Pogorelov et al. and Hicks et al. where they discussed the effect of overfitting and the consequences of dataset-specific artefacts. To help solve these predicaments look back at the two hypothesises presented in section 1.2:

Hypothesis H_0 stated that sparse information in the images made it harder to classify the images correctly. Figure 3.1 from Hicks's paper shows the case where the black areas with sparse information affect the classification. As we can see, the black areas trigger as a "positive" in some om the saliency maps. We can interpret this as the network learning features not useful for the classification. Our quest is to check the validity of this hypothesis. We propose to test the classification of images with and without areas with sparse information.

Hypothesis H_1 stated that dataset specific artefacts create false positives and negatives. This error is clearly shown in 3.1b, where the classification is affected by the green square in the image. As long as the classification is affected by dataset specific artefacts, the ability to adapt the dataset to new use cases might suffer.

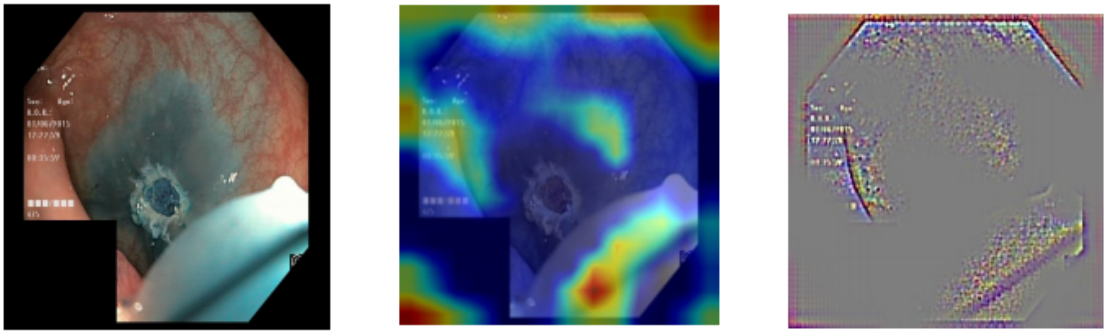
We note that H_0 concerns both training and testing on the same dataset, while H_1 is more concerned about the generalisability of the model, and hence the goal is to use different datasets for training and testing, as well as testing at the training set.

talk more about the differnt images in relation to steven.



(a) Original image (b) Heatmap (c) Saliency map

Figure 3.1: Heat and Saliency maps of an unaugmented dataset



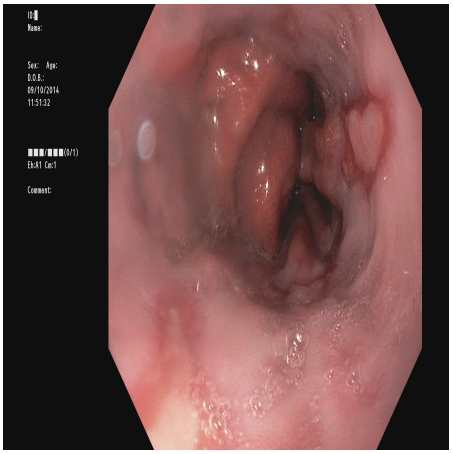
(a) Augmented image (b) Heatmap (c) Saliency map

Figure 3.2: Heat and Saliency maps of an augmented dataset where the green corner is removed

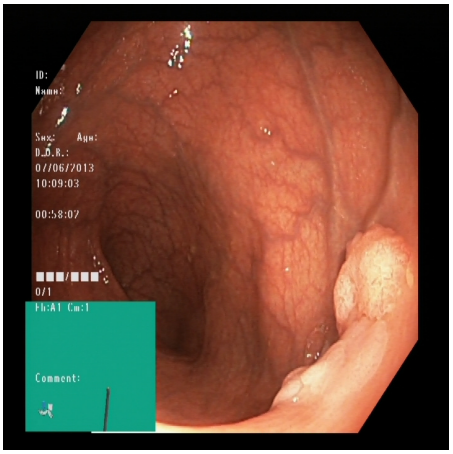
To test the two hypotheses, we first need new datasets to compare against a base case. In addition to the dataset with sparse information and dataset-specific artefacts, we need similar looking datasets without these unwanted features. In an ideal scenario, we would have the same dataset without the features added post-capture.
When it comes to real machine learning gathering (labelled) data for the training is often a challenging task. In this thesis, we have decided to focus our attention in to modifying existing data instead of finding new data.
We propose to use unsupervised machine learning to inpaint the areas with dataset specific artefacts as well as sparse areas. We then propose a transfer learning network to classify the newly created images. More here i feel

3.2 Design of the inpainting algorithms

We first want to set up a platform where every dataset is made with the same parameters, except for the dataset-specific parameters that define the dataset. When generating the datasets, we use the Kvasir dataset [34] as the training set, as Hicks et al. did in their paper on removing dataset specific artefacts [9]. In addition, we use datasets without the artefacts for testing. This selection of image source was made intentionally to have a fundamentally different test and training set, and the more differences between testing and training set the more of an indication of generalisation.
Figure 3.3 shows two different image types from the Kvasir dataset. Figure 3.3a shows an image of esophagitis. This image shows one of the main problems with the Kvasir dataset when it comes to artefacts. In addition to the cut corners, we have an extra wide area to the left of the image containing non-relevant information like name, sex, and other comments. This area gives us ample opportunity to test hypothesis H_0, given that the image contains a large amount of sparse information that we want to remove to see the effect on classification. We believe that if we can change images like Figure 3.3a into images like in Figure 3.3c we will see an improvement in classification when testing and training on the same images.
Figure 3.3b shows another problem with datasets like Kvasir. Here we have a green square in the bottom left corner that occludes parts of the image and the same type of text displaying name age and other non-relevant information. We recall from section 3.1 that information like this can give the classifier false positives, and subsequently provide us with a lower classification score. We believe that if we can change images like Figure 3.3b into images like in Figure 3.3d we will see an improvement in classification when testing and training on different datasets.
We propose three different types of inpainting to prove or disprove our two hypothesises.



(a) Example of an image with a large area without relevant medical information



(b) Example of an image with green square occluding the parts of the GI tract

(c) The same image as in Figure 3.3a with the non-relevant information removed

(d) The same image as in Figure 3.3b with the green square inpainted

Figure 3.3: Images where the troubling areas are removed before training

3.2.1 Removing black corners

The most straightforward experiment to conduct is to test how the removal of the black corners will affect the result. As we propose in hypothesis H_0, we believe that this masking can help with giving the classifier fewer areas with non-relevant information. Figure 3.4b shows the mask used to inpaint the corners.
As we recall from section 2.1.1, the black edges around the images in our datasets is also, in general, present in medical colonoscopy images. By removing the black corners around the image, we do not change *Kvasir specific* artefacts, but according to our first hypothesis, we believe we will get a higher classification accuracy since this removes areas with sparse information.
When classifying images during the testing stage, we need to take inpainting during training into account.

1. We can do the same masking and inpainting on the test set.
2. We can crop the image in a way that removes the black corners without inpainting.
3. We can forgo modifying the test data and just run the test set as is.

Method 1 We tested method 1) in our paper "Using preprocessing as a tool in medical image detection" [35]. The goal of this paper was related to hypothesis H_0 with the fact that we wanted to see how removing areas with sparse information affected the result of classification when training and testing on similar datasets. In the experiments run within this paper, both the training set and test set were inpainted. Since the focus of this task was to classify a test set we had from beforehand correctly, we had the option to preprocess the test set in addition to the training set without any penalties based on time restrictions. Our paper inpainted the test set as proposed in method 1) and from the results, it showed minimal improvement. The lack of improvement is mainly not connected to the fact that we inpainted the test set, but the fact that the test set came from the same distribution as the training set, and subsequently ended up with a model that overfitted to the data. Given that we, in the end, want to classify images from a live colonoscopy, we have decided to forgo the inpainting of the test set using method 1). By not augmenting the test set the experiments are also suited to reflect a larger research area of machine learning.

Method 2 Method number 2) was the proposal of cropping the images during evaluation. Thambawita et al. did similar methods in the Mediaeval 2018 conference but also had the same cropping during training. In the paper "The Medico-Task 2018: Disease Detection in the Gastrointestinal Tract using Global Features and Deep Learning" [36] we can see that this method worked with great success. The operation of just cropping images is also multiple times as fast as inpainting images, so it is feasible for a live recording. The downside of cropping the images from the test set is the fact

that we do not have control over what we remove from the data. Given that the test set might come from a completely different distribution, we might unwillingly remove information we desire to keep. We do also run into the problem that cropping the images are not feasible when the sparse areas are within the image, and not at the outer edges.

Method 3 The last proposed method is not to augment the images during testing. This method, since we are not preprocessing the test data at all, is the fastest when it comes to live classification. Without the augmentation, we risk getting a lower classification score, but we do not remove any data. Also, we make our results better reflect on how it will work on non-medical datasets.
In this thesis we will use method number 3). We base this decision on that we want the final product to have the option to be used live and be easily adaptable to other datasets.

3.2.2 Removing green squares

The next major area in question is the removal of the green squares located in the bottom left area of some of the medical images. This area is a Kvasir specific artefact and is found in 38% of the images spanning five out of eight classes. By inpainting the lower left area we can see if our hypothesis H_1 is correct since here we are removing a dataset-specific feature that the network can use to determine classes. We have also chosen to inpaint every image, regardless of the green square is there or not. We do this so that the network can not "learn" that the pattern from the inpainted area correlates with the five classes with the green square, and hence defeating the purpose of the inpainting. Figure 3.4c shows the mask used to inpaint the lower left corner.
When classifying images during the testing stage, we have the same decision to take when it comes to inpainting of the test set.

1. We can either do the same masking and inpainting on the test set.
2. We can crop the image in a way that removes the dataset-specific artefacts.
3. We can forgo modifying the test data and just run the test set as is.

To keep the consistency between to different inpainting methods, we use method 3) for the test set. As with the black corners to be inpainted, we choose not to preprocess the test data, as this takes time, and should in theory not be necessary to get the right classification.

3.2.3 Removing both corners and the green square

The last set we want to test is the combination of both inpainting the green square and the black corner as shown with the mask in Figure 3.4d. Here we hope that a combination of hypothesis H_0 and H_1 will give the strength of both methods without any harmful interference.

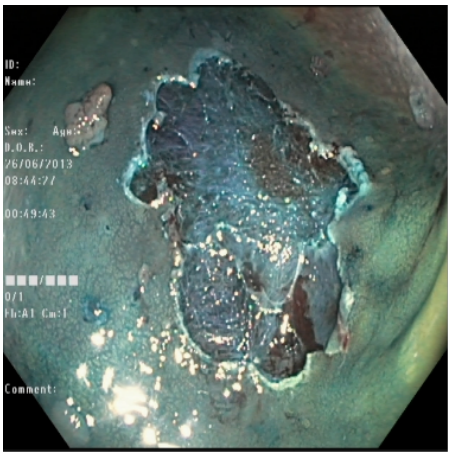
3.2.4 The generative modelling algorithms

With the three masks discussed, we now want to present the generative modelling algorithms. As addressed in section 3.1 our goal is to make new datasets without the unwanted artefacts based on the original dataset. We have chosen to use the two generative models presented in section 2.5.1 and in section 2.5.2, namely Autoencoders and GANs.
As we recall from section 2.5.1 the autoencoder and GAN[1] networks both use a ground truth when training. When training we need images that are already inpainted as a reference point. We solve this by zooming in the images to the limit that the edges are gone, as shown in figure 3.4b. When removing the dataset-specific features, we have the luxury that not all images contain the green square, giving us ample data for training after the images are sorted. By removing the corners by zooming and by using images without artefacts during training, we have images where we know the whole ground truth, giving us the option to use MSE for backpropagation.
When training the GAN, we often do not need a ground truth behind the mask, as the network only tries to discern if the image is real or fake. This gives us in practice the same restrictions as with backpropagating with MSE.

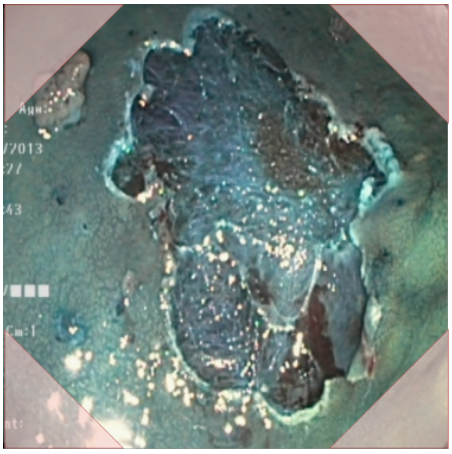
3.2.5 summary

We have now talked about the three main masks we use, combined with the two generative models that make the datasets. In total, we end up with six generated datasets, two for each mask type, and one unaugmented base case. The seven datasets will be the basis for our test of the problem statements.

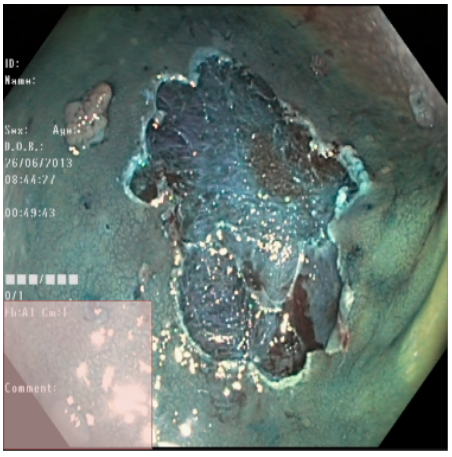
[1]It is necessary to mention that regular GANs does not use a ground truth when training, though our modified GAN do.



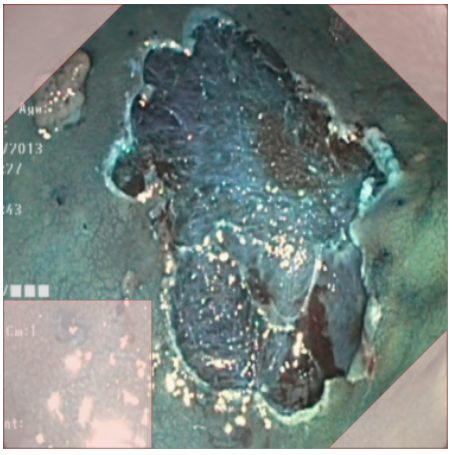
(a) Image from the original non-augmented dataset



(b) Red area shows the area masked in the first of the generated datasets



(c) Red area shows the area masked by the second of the datasets



(d) Red area shows the area masked by the third of the datasets

Figure 3.4: All three mask types used in this thesis, and associated images used during training. At dataset in

3.3 Design of the transfer learning experiments

To test the hypothesises we need a system in place to compare the datasets we generate with inpainting. As we recall, both our hypothesis is based on an improvement on a base score. To see if we have any improvement we propose to use a classifier based on transfer learning to see how the newly generated datasets give a better classification score compared to the base dataset without augmentations.

We want to test our system with a range of different models and classifiers to test the validity of the system and to make sure that our results are not just good outliers.

Figure 3.5 shows the general structure of the transfer learning model we use in this thesis.

Hyperparameter optimisation of the models is a challenging task [37]. In this thesis, we have chosen to use automatic hyperparameter optimisation provided by the hyperparameter optimiser made by Borgli et al. in the MediaEval workshop 2019 [37], to give us the optimal model for training. Based on our dataset, the optimal network, DenseNet121 [38], was chosen as our primary model for evaluation. In addition to an optimal model, we have chosen to use InceptionResNetV2 [7] as a more general model. InceptionResNetV2 showed the highest top 1 and top 5 accuracies on imagenet when we designed the transfer learning program based on the Keras website in August 2018. We find a more recent table with more pretrained models in Figure 3.1.

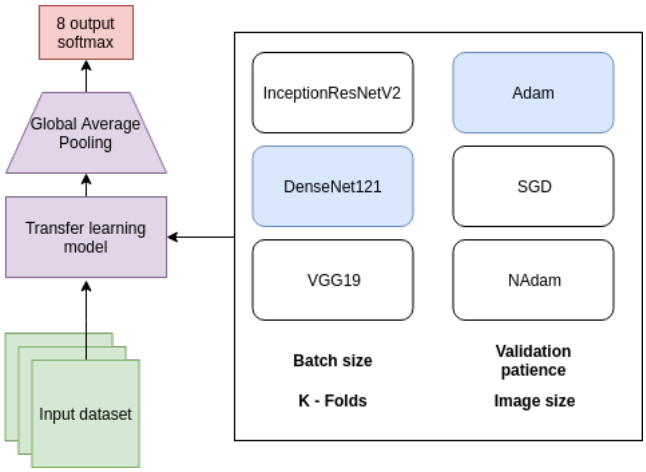


Figure 3.5: The model we use for classifying with the most important options for the learning process.

3.3.1 models

The models used in the evaluation of our results are fundamentally different and to better understand the results we give s short introduction of both pretrained network.

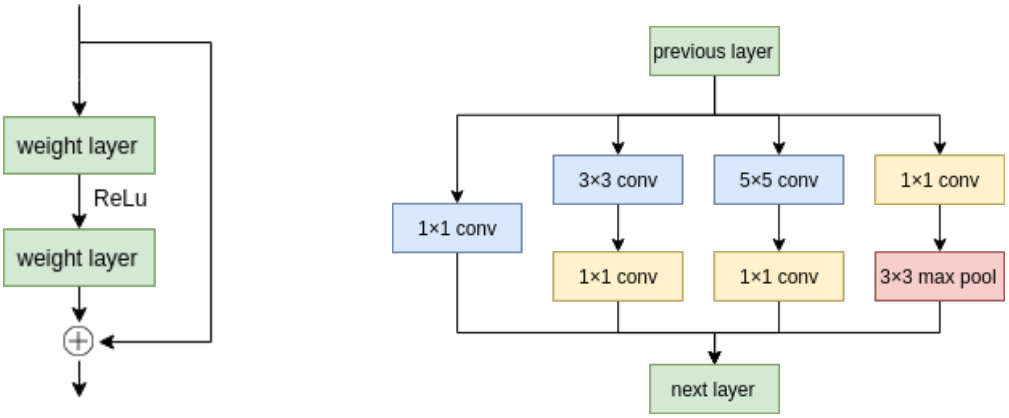
Inception Residual Network architecture We can often see performance gains in our network architectures when we increase the size of the network. To increase the size, we can either increase the depth of the network, i.e. the number of layers, or we can increase the width of the network, i.e. increase the number of units per layer. Both methods are often easy to implement, but it often comes with the drawback of either increasing the number of parameters to train, which can result in overfitting, or it can result in the computation time of the network to increase to an undesirable length. To find a balance between accuracy and memory is often a hard task when it comes to not only medical images but images in general.

Szegedy et al. presented an architecture for the imagenet challenge with the intention of reducing the computational cost of training large neural networks by decreasing the number of parameters [39]. The network presented, GoogLeNet, used inception modules to reduce the computational cost. Inception modules (networks within networks) tries to answer the question of witch convolution is the correct for each layer. Often it is up to the network architect to decide this, but with inception modules, the network chooses for itself. In the GoogLeNet paper, the inception modules are the same as in Figure 3.6b, showing the option between 1×1, 3×3, 5×5 convolutions, and 3×3 max pooling, followed by a concatenation of the four options. In addition to the convolution in parallel, the original authors added 1×1 convolution before the larger filter. This layer uses 128 filters for dimensionality reduction.

Microsoft proposed another solution to training very deep neural networks in the form of residual modules [6] as shown in Figure 3.6a. When using sufficiently large enough networks the backpropagation often encounter the problem with vanishing gradients. By adding a direct link between the layers, we counteract the vanishing gradient problem. A consequence we encounter when using residual modules is that instead of recreating the input for each layer, each layer learns only to modify the input value instead of completely recreating it.

Finally, combining the inception and residual modules, we end up with the modules used in InceptionResNetV2. Here we combine the strength of both networks. A compressed view of the network is shown in Figure 3.7 as shown in the article by Mahdianpari et al. [40]

DenseNet architecture Huang et al. published the DenseNet architecture, as shown in Figure 3.8, with the same goal as InceptionResNet, namely to reduce network size and to get better classification accuracy from a better gradient flow [41]. In the DenseNet model, all layers are connected in a way to ensure the maximum flow of information between layers. The network achieves this flow by letting each layer convey all its information to all subsequent layers with similar dimensions. This gradient flow gives the network the ability not to store redundant information, and Huang et al. showed significant results for classification with small training datasets where overfitting often would serve a problem.



(a) Residual module concept as proposed in [6]

(b) Inception module concept as proposed in [39]

Figure 3.6: The two main components in InceptionResNetV2

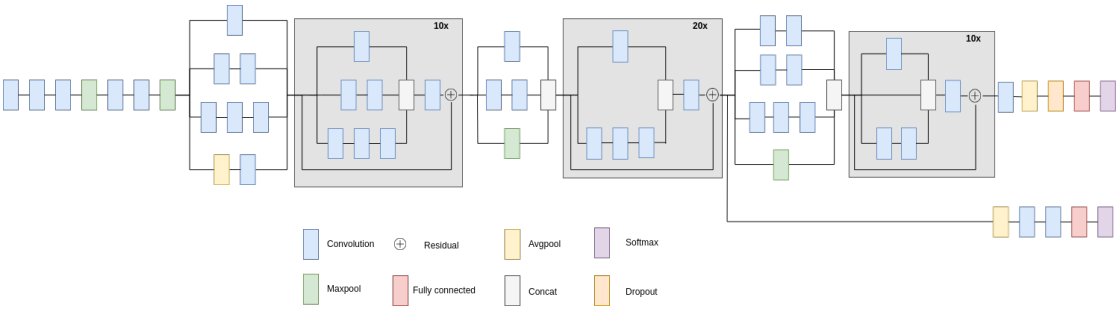


Figure 3.7: The compressed view of the InceptionResNetV2 network inspired by Mahdianpari et al.

3.3.2 Pooling

The pretrained networks were initially trained for the 1000 classes in the imagenet dataset. Without doing anything to the network, it would output one of the 1000 classes when confronted with an image. In this thesis, this is not desirable since none of the original classes is usable for medical image classification. When we introduce our pretrained network, we cut the last two layers from the model and introduce a pooling layer instead. The two layers cut is the Fully connected layer with 1000 nodes, and a softmax layer (The Purple and The Red blocks at the right side of the networks in Figure 3.7 & 3.8) It is common in the practice of transfer learning to add a pooling followed by a softmax layer as a substitute for the original layers removed. The pooling can either be average or max pooling, and we have the option of using global or local pooling.

rememver to talk about global pooling in pooling chapter

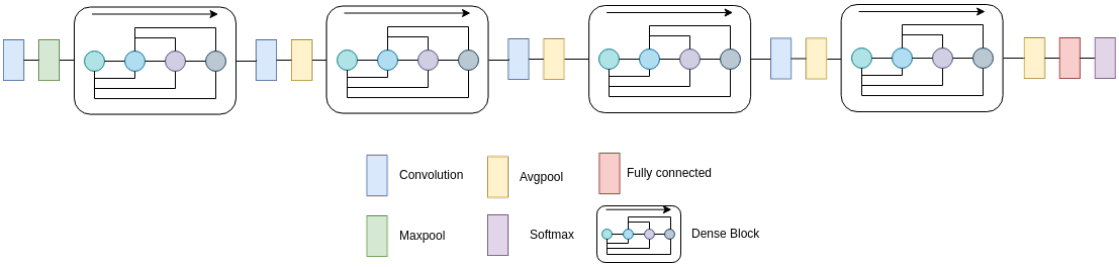


Figure 3.8: The compressed view of the DenseNet network inspired by Mahdianpari et al.

3.3.3 What does the different combination do to the result in our opinion (renamed to something cool :sunglassemoji:)

Based on the article by Huang et al. [41], we believe that the densenet model would give us the best result. As stated, the densenet model has good gradient flow, and the architecture helps small datasets with overfitting. We have also decided to use global average pooling as our pooling layer for both models. We base our decision of the usage of global average pooling on the models used in the Mediaeval workshop by Hicks et al. [42] and Borgli et al. [37] whom both got great results when using global average pooling on medical image data.

3.3.4 summary

We have, in this section, talked about how we want to use transfer learning to help with the classification. We have looked at the two main advantages with transfer learning, namely the time saved in training and the accuracy gained during evaluation. The advantages coupled with the fact of the widespread use of transfer learning makes it the best option for us, compared to regular classification. We then took the results from the SAGA tool made by Borgli et al. and used the most optimal network as our primary network in this thesis. In addition to the best network, based on Hyperparameter optimisation, we used InceptionResNet as a secondary network for evaluation, reasoning that the use of a generally good network would yield a scenario closer to the real world. After a brief discussion about the composition of the networks, we concluded with the DenseNet model would yield the best result based on our datasets.

3.4 Libraries

With the general structure of the algorithms discussed, we will now go more in-depth into the libraries used in the creation of the programs.
In this section, we will discuss the foundation of our code, important external and internal libraries, and the setup and execution of our project. We first discuss the

programming language in question, give insight into the reasoning behind it. Then we will look into the framework used for machine learning, and in detail how it implemented in our programming language. Lastly, we look into the wrapper we use to get a higher level of abstraction over our code, together with custom functions that are in use by our wrapper.

3.4.1 Python

When doing machine learning, some of the most popular languages, in no particular order, are Python, Java, R, C++, and C [43]. Some of these languages, like C and C++, are chosen for their speed, which is often a significant factor in Machine learning. Other languages, like R, is chosen because of its integration into the scientific community long before machine learning became a trend. The last group, consisting of Java and Python has gained popularity because of its already big user base and user-friendliness. Python is also the winner when it comes to machine learning because of, like R, its integration into the scientific community. Right now Python is the leading language for machine learning. Driven by this, there is considerable focus into making it faster, to compete with already fast languages, like the C family.
Python is an interpreted, high-level, general-purpose programming language created in 1991. It, like many other modern languages, is object-oriented and supports functional programming. Mainly because of the excellent support when it comes to machine learning, and the general "easy to use and no compiling" we have chosen python as the base for our code in this thesis.

3.4.2 Tensorflow

Arguably the biggest reason for the success of machine learning in python lies in Tensorflow [44]. Tensorflow is a machine learning package released by Google in November 2015 and has since then become the leading framework for machine learning worldwide [43]. Tensorflow is in use by companies like AMD, Twitter, eBay and Snapchat.
Tensorflow is today a multi-language tool, but it had its origin in python. It is just in later years that other languages have gotten tensorflow support. The data flows through a graph network, where the objects in the graph describe the mathematical operations used in the machine learning, and the edges between graphs are the multidimensional arrays storing the weights associated with the operation in question. The name Tensorflow is a combination of the flow we experience during calculation and the tensors between the mathematical operations.
As stated, Python, and subsequently machine learning in Python, would be much slower than a counterpart in C. Because of this, Tensorflow works as a layer of abstraction to code running in the C language. In addition to the layer of abstraction, Tensoflow can do the computation using a graphical processing unit (GPU). Using a GPU for machine learning will often give a considerable speedup to the training, given

that the GPU is designed with matrix multiplication in mind. This gives the GPU the option to easily calculate large chunks of the data in a fraction of the time.
Other packages for machine learning like PyTorch [45] and Microsofts CNTK [46] would be other options, but as they are not as widespread. Because of the widespread usage, we have chosen to use Tensoflow as our primary machine learning package.

3.4.3 Keras

One of the least attractive things with tensorflow is its unnecessary complexity. Even though Tensorflow offers more abstraction compared to running the code in pure C, the Tensorflow library can be unnecessarily complex. As a result of this, many external libraries try to simplify many of the complexities that accompany tensorflow. Libraries like TFlearn is a modular and transparent deep learning library on top of tensorflow. It gives a higher-level API to Tensoflow to reduce complexity and speed up experiments [47]. The most successful library for on top of Tensorflow is Keras . Just as TFlearn, Keras is a high-level package written in python **chollet2015keras**. It is capable of running on top of TensorFlow, CNTK, or PyTorch, which is the tree most popular machine learning libraries at this time. Keras is made to be user-friendly, modular, easily extensible, and to work with python.

cite keras

One of the core elements of Keras that makes it a better choice than just running, for instance, Tensorflow, is the concept of a model. A model in Keras is a way to organise the layers of the network in a more organised way, giving a better understanding of how the network is set up, and how each layer type contributes to the graph.
This thesis relies on Keras as a wrapper for tensorflow. As stated, the use of models and the simplicity of the language makes it an excellent choice of such a large project. Also, Keras has good support for convolutional operations which is the most used methods when managing images. Keras also has the most popular pretrained convolutional neural network models available in its package.

3.5 Custom functions for Keras, tensorflow and python

3.5.1 Channel wise fully connected layer

A problem often encountered when working with autoencoders which are not undercomplete is the fact that they learn to represent the data flawlessly. [48] When this problem arises, the network does not learn the fundamental features that define the dataset and instead passes the signal through the network without any consideration of the input data. This flaw will often defeat the purpose of the algorithm, so data scientists often put in safeguards, like undercompleteness or regulisers, to combat this lack of feature learning. This problem extends to other types of generator networks where there are not sufficient compression or regularisation in the layers of the network. Even though this problem often is solved by compressing the network into a space that

can not contain the information exactly, the network does not always learn the features that define the network.
We propose a custom "channel-wise fully-connected layer" in Keras to help with the problem of correctly learning features. This layer is based on the work done by Pathak et al. in their paper on context encoders[49]. The channel wise fully connected layer is primarily used in the GAN to transfer information within each feature map, without using convolutions to do it. As we recall, fully connected layers are often not suitable because of the large size of the weight layer associated with it. This layer is essentially a fully connected layer with groups, where the goal is to propagate the information within each feature map. Given the latent space of $n \times n$ with m feature maps, by not connecting the feature maps together in the fully connected layer we achieve a parameter reduction from m^2n^4 to mn^4 (ignoring bias terms) [49]
With the "channel-wise fully-connected layer" the network can learn features from the entirety of the image, and not just local regions as it would with just convolutions.
Listing 3.1 shows the source code used for the channel-wise fully-connected layer.

```
from keras.engine.topology import Layer
import keras.backend as K

class CWDense(Layer):
    def __init__(self, **kwargs):
        super(CWDense, self).__init__(**kwargs)

    def build(self,input_shape):
        _, self.width, self.height, self.n_feat_map = input_shape
        self.kernel = self.add_weight("CWDense",
                                      shape=(self.n_feat_map,
                                      self.width*self.height,
                                      self.width*self.height),
                                      initializer='glorot_uniform',
                                      trainable=True)
        super(CWDense, self).build(input_shape)

    def call(self, x):
        x = tf.reshape(x,[-1,self.width*self.height,self.n_feat_map])
        x = tf.transpose( x, [2,0,1] )

        x = K.dot(x,self.kernel)

        x = tf.transpose(x, [1,2,0])
        x = tf.reshape(x,[-1,self.height,self.width,self.n_feat_map])
        return x
```

Listing 3.1: The channel-wise fully-connected layer source code

3.5.2 Subpixel

When working with the generative adversarial network, we wanted to achieve more realistic representations at a reasonable image size. Making large scale images in

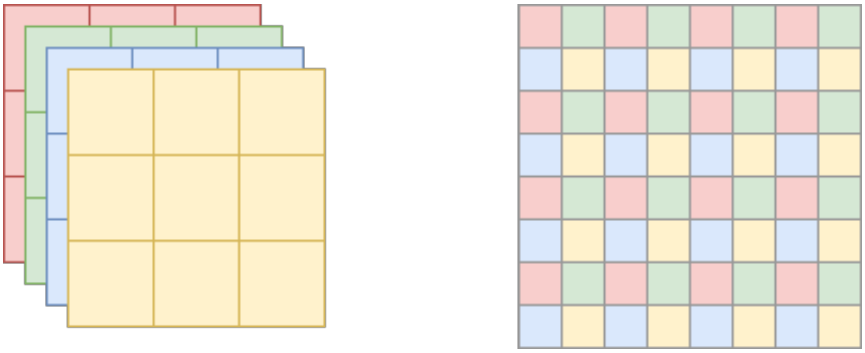


Figure 3.9: How the layers in the sub pixel layer is stacked. Recreated from the SubPixel paper by Shi et al. [52]

generative adversarial networks has been a challenge that has only recently been cracked [50] [51]. As an early measure to fix this problem, we experimented with the use of a Sub-pixel layer as presented by Shi et al. [52] to give a more realistic output compared to just a standard conv-tanh layer.
During this thesis, we have done multiple tests with this layer. In the datasets used for evaluation, we did not end up using any Subpixel layers.

3.5.3 Masklaod and Setload

The default way of loading data into Keras is with the usage of the default generators provided by the library. The Keras.ImageDataGenerator() does an adequate job when loading images, though it had some shortcomings. Maskload and Setload are custom loaders we made for this thesis to get a more simple way of loading, augmenting, and storing images.
Maskloader has the primary job of adding and removing masks from images. In addition to masking, it can flip and mirror the images. Maskloader can mask text in images, though at a timeframe not feasible for this theis at over 2 seconds per image.
Setload is a library for loading a large number of images for training. In addition to loading images, we designed Setload to remove black borders from images if necessary. We use morphological opening[2] to remove any nonblack pixels in the edges of the image that original from text added post-capture, which gives ut a clean cut as close to the ´´true" image as possible. The Setload function defaults to normalise the images around zero. This normalisation is to stabilise both the GAN and AE.

3.5.4 Self attention

There are features in the images that are more important than others. One of the things we often want to preserve when we recreate images are hard edges. To get

[2]Morphological opening is the dilation of the erosion of an image

a semantically meaningful image, we often want to differentiate between background and the mucous. To see if the network can learn the features needed we are Introducing the Self Attention layer to help with this. We base the self-attention on the work done by Zhang et al. [53].
Our adaptation of the paper in code form is found in listing 3.2. Here we have the framework for a simple attention layer that, in our testing, was used with a residual connection. We do not use self attention in the final datasets used for evaluation.

```
class Attention(Layer):
    def __init__(self, filters, **kwargs):
        self.filters = filters
        super(Attention, self).__init__(**kwargs)

    def build(self,filters):
        self.kernelf = self.add_weight(name='kernelf',
                                       shape=(1,1,self.filters,self.filters),
                                       initializer='uniform',trainable=True)
        self.kernelg = self.add_weight(name='kernelg',
                                       shape=(1,1,self.filters,self.filters),
                                       initializer='uniform',trainable=True)
        self.kernelh = self.add_weight(name='kernelh',
                                       shape=(1,1,self.filters,self.filters),
                                       initializer='uniform',trainable=True)
        super(Attention, self).build(filters)

    def call(self, x):
        f = K.conv2d(x, self.kernelf, strides=1, padding="same")
        g = K.conv2d(x, self.kernelg, strides=1, padding="same")
        h = K.conv2d(x, self.kernelh, strides=1, padding="same")

        f = tf.transpose(f,perm=[0,1,2,3])
        i = f*g
        j = K.softmax(i)
        k = j*h

        return k
```

Listing 3.2: The self attention layer source code

3.5.5 Masked loss

A problem encountered for both the autoencoder and the GAN is what happens when the non-inpainted area gets too close to correct ground truth. As most of the image remains unchanged between the input-output space, the loss for most of the image approaches 0 while the inpainted area stays at a high loss value for most of the training. As we recall from section 2.4.4, we store our data as float32, and as the loss gets smaller and smaller, the squaring of the float32 gives at some point a number so small that the loss flips to a large integer and subsequently ruins the run.
To improve the stability of the training, we have modified the loss to only apply to the

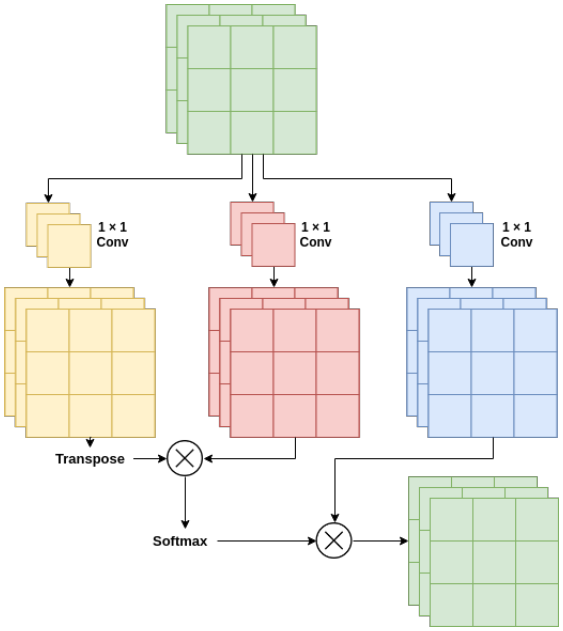


Figure 3.10: How the layers in the Self-Attention layer is stacked. Recreated from the Self-Attention paper by Zhang et al. [53]

areas we inpaint, leaving the rest of the image without a gradient to improve itself. Listing 3.3 shows the source code for the masked MSE. For each point in the MSE we apply a binary mask. If the binary mask is zero, we do not consider that space during backpropagation.

The masked loss showed a considerable increase in stability during training, but in the finished product, we only used it for the autoencoder when we struggled with stability problems.

```
import keras.backend as K

def masked_mse(mask_value):
    def f(y_true, y_pred):
        mask_true = K.cast(mask_value,K.floatx())
        masked_squared_error = K.square(mask_true *(y_true - y_pred))
        masked_mse = K.sum(masked_squared_error) / K.sum(mask_true)
        return masked_mse
    f.__name__ = 'Masked MSE'
    return f
```

Listing 3.3: The self attention layer source code

3.6 Stabilising the GAN

Before we ended up with the model we used in the thesis we ran multiple experiments to make the generative adversarial network stable for training. In contrast to the autoencoder, the GAN does not use the ground truth as a reference point. Where the autoencoder always has a gradient based on the input data, the generator in the GAN gets its learning gradient from another network.
This lack of a ground truth gives the GAN many pitfalls that cause the training process to crash [3].

Normalise the inputs One of the first measures we did to prevent training collapse was to normalise the inputs. Instead of using images in the range 0 to 255 in pixel values we switched the values to -1 to 1. Later, when the images were generated, we switched out the standard sigmoid output layer with a tanh output layer. As we wanted the output to be between -1 and 1, this was necessary, as the sigmoid only outputs between 0 and 1.

Using gaussian noise perhaps write about this

Normalising the batches One of the most significant challenges we encountered when training the adversarial network was the use of correct normalisation.
The practice of training the discriminator with real and fake samples separately gave higher stability overall.
The use of instance normalisation gave a better result compared to using batch normalisation. We believe this is contributed to the fact that the discriminator learned that the average pixel value was lower for the whole batch since the area inpainted had 0 as the pixel value.
The final model ended up not using batch or instance-normalisation.

Avoiding sparse and vanishing gradients Most of the well-known networks use the ReLu[54] activation function [5] [7] [6]. We saw the best result when we used non-sparse gradients during training. Instead of using ReLu we used the slightly modified LeakyReLu [55].
In addition to trying to remove sparse gradients, we also wanted to address the problem with vanishing gradients during training. Given that we have fully saturated pixels (with the value of 1 or 255) and we have fully darkened pixels (with the value of -1 or 0) we, at the end of the experimentation phase, ended up removing the tanh layer. The removal of the tanh layer meant that the pixel values could be arbitrary on both positive and negative value, so we had to clip the value not to get an error at test time.

mention RReLu, but the number of parameters not worth it.

[3] Crashing is not the right word to use, but the result is the same: The learning process stops.

Avoiding residual and inception layers When training the GAN experiments shows that the usage of both residual [32] and inception [39] models does note contribute to a good result when training the GAN.
Residual modules primary strength is that they always send the image/signal throughout the network in addition to the standard layers. Instead of the network needing to generate the whole image for each layer, the network instead adds or subtract from the original image for each layer. This modification to the original image might seem reasonable when it comes to inpainting, but in reality, this does not work. Given an image where we want to change only the inpainted area, the image is about 80% unchanged. The network could focus on just filling in the inpainted area in theory, but in practice, the network tries to change the rest of the image in addition to the square. This incorrect inpainting gives us a generator that changes too much of the image and a discriminator that does not learn any important features since the input and output are relatively similar from the start.
Inception modules primary strength is the fact that the gradient can flow throughout the path most suited to the problem at hand. We tested some training runs with inception modules, but the result was not impactful enough to continue to use this architecture.
From multiple training runs, it seems like just a straight forward encoder-decoder network for the generator yielded the best result. While the best result for the decoder was to use convolutions with stride do downsample the signal.

this might not be the right chapter for describing what layers are in the GAN, so please move

3.7 Code Description

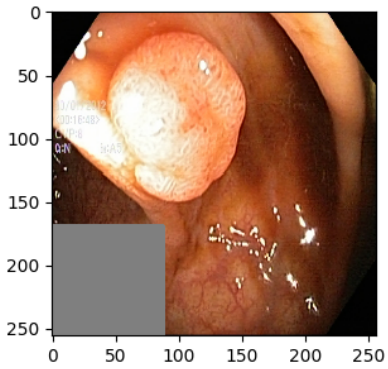
We have, at this point, gone through the goal of our thesis, and shown how we want our result to be generated and evaluated in practice. We will now go more in-depth into the two networks used for generating the new datasets and go in-depth into the model we use for classification.

3.7.1 Autoencoder

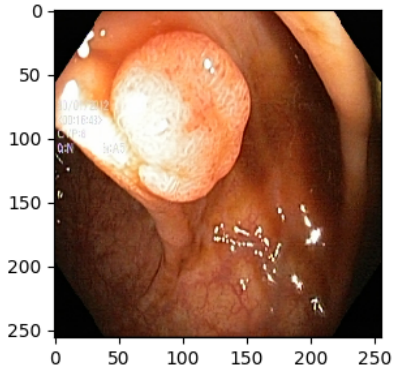
The autoencoder we used to generate the datasets used in this thesis bears a resemblance to the standard autoencoder proposed in chapter 2.5.1.

Loss, Optimiser To get the autoencoder to give the best result, we have chosen to use the mean square error loss as in equation 2.3 and the Adam[56] optimiser. The mean square error was a logical choice since we already have the ground truth and we only want to recreate the inpainted area based on what used to be there before the masking. The Adam optimiser was chosen by the widespread usage in machine learning, coupled with the fact that it works well with sparse gradiens.

Encoder The input to the autoencoder were the masked images at 256×256px to compress the information in the encoder we imply used convolutions with a stride of 2.



(a) Image the autoencoder receives as an input



(b) The missing part the autoencoder tries to replicate

Figure 3.11: A standard image taken in by the autoencoder

An option to using stride for the downsampling would be to use pooling, as described in section 2.4.4. Here, there are still room for experimentation.

Decoder between the encoder and decoder we added a 25% dropout layer. This layer is the only reguliser in the network, though since the job were to inpaint and not recreate, the autoencoder needed information about a rather large area of the image, and hence had little possibility to overfit.

Upsampling could either be achieved with upconvolution or with upsampling. Using upconvolution gives the network more variables (as the filters use weights, and upsampling does not), and hence would require more training.

In this thesis we achieved the greatest results by using upsampling compared to upconvolution, though we can not rule out that upsampling would be better with more complex images, or at larger image sizes.

To describe the model we will look at the example where we try to inpaint the green square in the image, and nothing else.

To train the autoencoder for inpainting, we divide the dataset in two, first images with the green square and images without the green square. We discard the images with the green square since they are not viable for training. The resulting dataset will only contain images without green sources.

The next step before training is to cut the images according to the mask provided. Figure 3.11a shows what the finished masking looks like, and 3.11b shows what we want to achieve after training.

We feed 3.11b into the autoencoder consisting of convolutional layers, leakyReLu layers, and a tanh layer.

more

remember to talk about loss

3.7.2 Generative adversarial network

The gan used the same generator discriminator elements as the Goodfellow gan, however the most significant difference is the fact that our model does not generate the image from Gaussian noise.

Instead of the standard Gaussian noise as input to the generator, the inpainted image is taken as input. Here, as in the autoencoder, we use stride to downsample the image.

perhaps draw the shape?

3.7.3 Transfer learning classifier

The dataset we are making with our generators needs to be classified.

The classifier we use is based on the idea of reusing networks we already know apply well to the real world. We have made a classifier that, by default, use one of the pretrained networks provided by the Keras framework.

Model	Size	Top-1 Accuracy	Top-5 Acc	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
ResNeXt50	96 MB	0.777	0.938	25,097,128	-
ResNeXt101	170 MB	0.787	0.943	44,315,560	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

Table 3.1: Models provided by keras

Table 3.1 shows the pretrained networks available to load in the Keras framework. When training we did some extra steps at the end, namely added global average pooling and a fully connected layer with the desired number of outputs (usually eight classes, and eight outputs)

3.8 Summary

In this chapter we have, in more detail, looked at the process and purpose of inpainting and classifying. We started with the discussion regarding where to inpaint, and the problem statements we wanted to test for each of the inpaintings. We made a decision regarding if inpainting of the test set were feasible, and decided the two types of generative modelling algorithms we use in this thesis. We looked at the need for a classification model to discern the results of our inpainting and decided to use transfer learning to represent a real-world scenario better. We looked into the two transfer learning models we use in this thesis, namely Densenet121 and InceptionResNetV2, and gave a brief overview of their similarities and differences.
After a more conceptual view, we talked more about how we would achieve the experiments in practice. Here we talked about why we chose python, tensorflow and keras, followed by custom functions needed for our machine learning algorithms. We ended the chapter with the description of the three models in practice.

Chapter 4

Experiments

In the previous chapter, we described our methodology and our system for performing inpainting to improve the input data. We presented the framework we used for our setup, and we went into detail on the configuration of the experiments, and the project as a whole describing both the masking and the neural networks. In this chapter, we will start with the description of the datasets used on the results and the metrics chosen as a reference point for the evaluation of our results. Finally, for each model, we will evaluate it for the datasets we have presented.

4.1 Datasets

To show and explain the experiments, we need to look at the datasets we have to evaluate our results. In this thesis, we primarily use three datasets to train and evaluate our results. Here we will look at the different datasets. We will comment on similarities and differences.

4.1.1 Kvasir

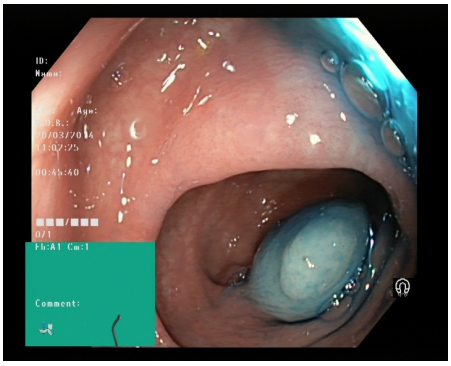
Kvasir [34] is a Multi-Class Image Dataset for Computer Aided Gastrointestinal Disease Detection made in Norway. The data is gathered from the Vestre Viken Health Trust, and it contains not only polyps but also two other findings, two classes related to polyp removal and three anatomical landmarks in the GI tract. The dataset contains the eight classes mentioned above with 1000 samples for each of them. In the Kvasir dataset, we have primarily two dataset specific artefacts and one artefact often seen in colonoscopy images. The first two artefacts are text and the green box in the lower left corner. The last artefact is the rounded edges; this kind of frame is a shared feature between all three datasets. Over 60% of the dataset contains images with text, and text is represented in all classes. The text is located on the left side of the images, and it is always white. The green square is present in 38% of the images but only in 5 of the classes. Figure 4.3 shows a sample with images from each class. The classes with images containing green

squares are also the same images that contain green squares in the figure. As we can see in Figure 4.3 all images got the rounded corners.

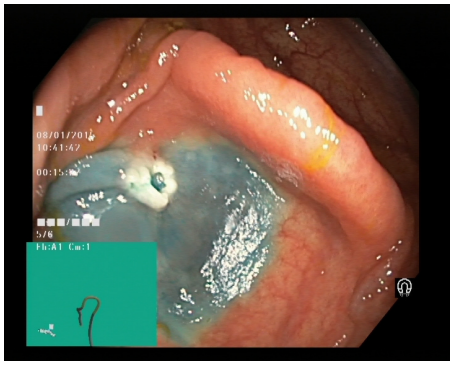
4.1.2 CVC 356

The CVC 356 dataset is an publicly available dataset

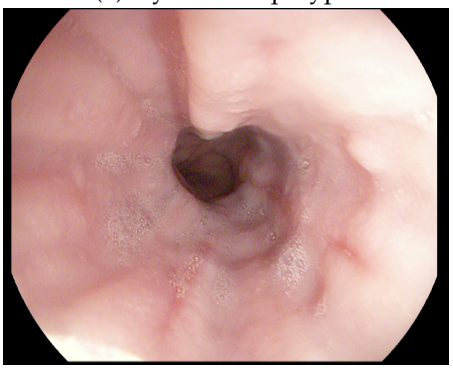
4.1.3 CVC 12k



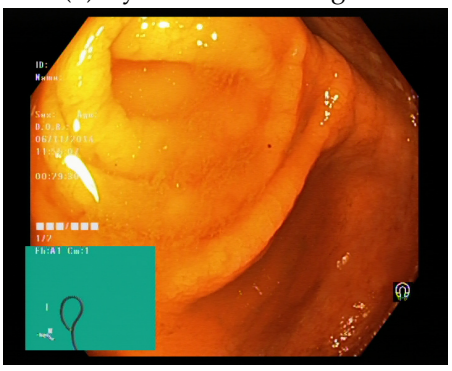
(a) Dyed lifted polyps



(b) Dyed resection margins

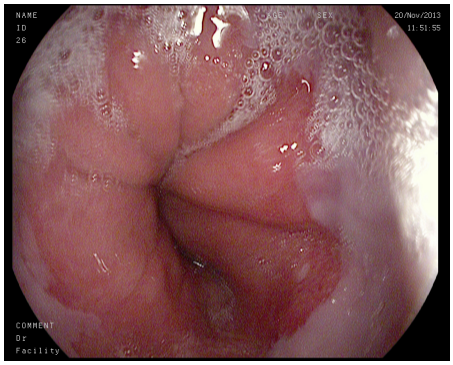


(c) Esophagitis

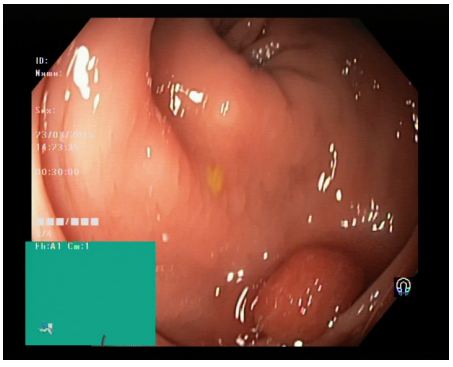


(d) Normal cecum

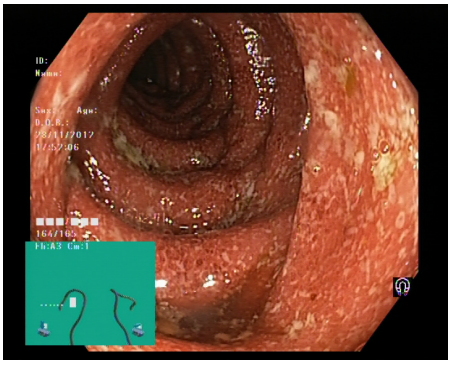
(e) Normal pylorus



(f) Normal z line

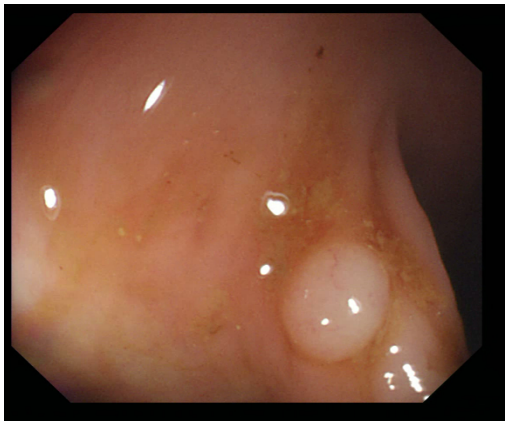


(g) Polyps

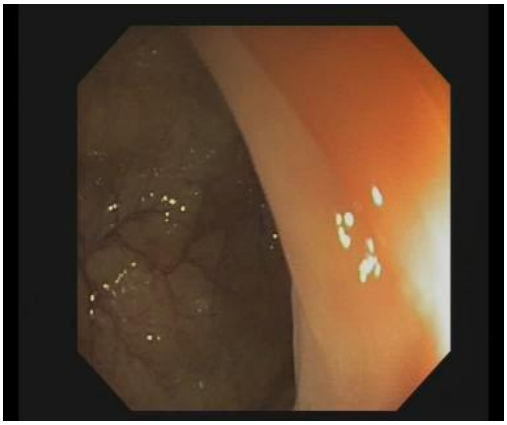


(h) ulcerative-colitis

Figure 4.1: The Kvasir dataset with each of the eight classes



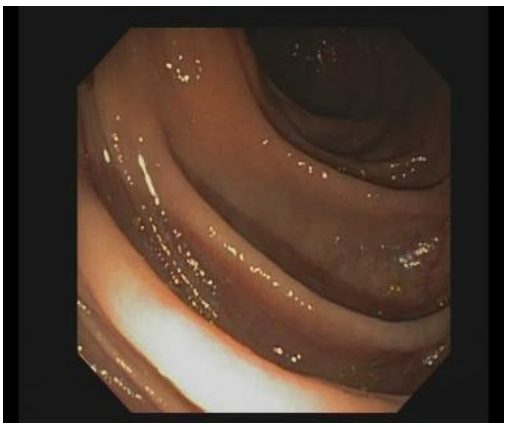
(a) Polyp



(b) Non-polyp

Figure 4.2: The two classes from the CVC 356 dataset

(a) Polyp



(b) Non-polyp

Figure 4.3: The two classes from the CVC 12k dataset

4.2 Metrics

To discern the results of our experiments we introduce multiple metrics and tables to get an indication of our success. The primary dataset we used for training, Kvasir, was split into k number of folds, using k-fold cross-validation. K-fold cross-validation is a tool used in statistics and machine learning to help to get an accurate representation of the data based on finding a statistical average of the dataset. In machine learning, it is a powerful tool that can help with adapting to new datasets and prevent overfitting. We recall that the Kvasir dataset contains 8000 images, with 1000 of each class. In our testing, we split the dataset up in k=6 folds. This split means that we split our dataset into six pieces before training. With the six folds, we assign one of them as a test set, and we assign the four other for training and validation. We then train our data five times, using four folds for training and the last fold for validation during training. For each training run, we rotate the validation set, so each of the five folds is used for validation once.

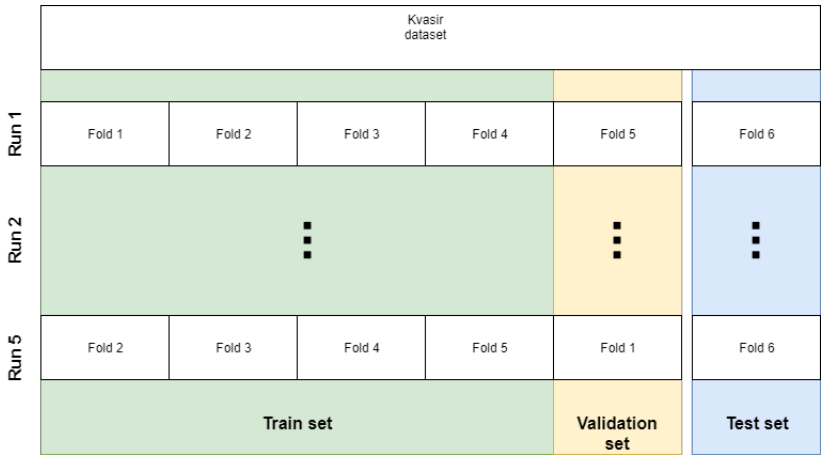


Figure 4.4: The Kvasir divided in to 6 folds

The advantage of using K-fold cross-validation is that we maximise the utility of the dataset. We find the distribution of the dataset that hopefully covers the most significant range of the unseen data.

4.2.1 The confusion matrix

With k-fold cross-validation, we end up with the dataset that scored the highest during the final validation step. There are multiple ways to calculate metrics for how well a dataset is doing, but they all are a comparison of the predicted class versus the true class.
Take for instance the case with the 8-class dataset Kvasir, where we predict an image to be normal-cecum. We translate this to an integer representation of, for instance, class 3. In our example, the actual True class is normal-z-line, here represented as 5.

We can represent this as

$$\begin{bmatrix} 3 & 5 \end{bmatrix}$$

Storing value pairs like this can very quickly get cluttered and unorganised. The most common way to store these value pairs is to use a confusion matrix. We initiate the matrix as a $N \times N$ matrix where N is the total number of classes as shown in Figure 4.5

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.5: An empty confusion matrix

After we have initialised the confusion matrix, we add each value pair as to the matrix at its corresponding position. Given the pair $\begin{bmatrix} 3 & 5 \end{bmatrix}$ we add one to the position corresponding to (x=3,y=5). Another example could be given the pair where we guessed class 0 and the true class was 0: $\begin{bmatrix} 0 & 0 \end{bmatrix}$, we add one to the matrix at position (x=0,y=0). With the two examples we get the following confusion matrix as shown in Figure 4.6

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.6: The confusion matrix with [3 5] and [0 0] inserted

As we fill in the matrix with more predictions, we can start to infer properties of the classifier. After approximately 1600 evaluations, our result might at the end look like Figure 4.7. We can see that the majority of predictions lie around the diagonal. This centralisation means that most of our results were classified correctly, as values at the diagonal are the same x and y values, and subsequently a correct prediction. We can also discern something about the four primary metrics associated with a value in the matrix.

True Positive (TP): True positive for a class is when the sample is predicted positive, and the True label is also positive. In the Kvasir dataset, we have a True positive result

$$\begin{bmatrix} 195 & 50 & 0 & 0 & 0 & 0 & 2 & 1 \\ 4 & 148 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 152 & 0 & 3 & 40 & 0 & 5 \\ 0 & 1 & 0 & 198 & 0 & 0 & 13 & 4 \\ 0 & 0 & 0 & 0 & 195 & 1 & 5 & 2 \\ 0 & 0 & 47 & 0 & 1 & 159 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 172 & 8 \\ 0 & 1 & 0 & 2 & 1 & 0 & 8 & 180 \end{bmatrix}$$

Figure 4.7: The confusion matrix with almost 1600 predictions

if, for the class polyp, we predict a polyp.

True Negative (TN): True Negative is the opposite of true positive. Given the class Polyp from the Kvasir dataset, we guess that the image is not a polyp when the True label is non-polyp.
False Positive (FP): False positive is, given a True label, we predict it to be False. We often call this type of error a "Type 1 error". In the polyp case, this is the case where we predict a polyp when there is no polyp present.
False Negative (FN): False positive is the case where we fail to predict the class when it is True. We call this error for "Type 2 error". False Negative is, in our case, the least desirable outcome for our classes with pathological findings like Esophagitis, Polyps and Ulcerative Colitis.

The metrics described is in the case of "single class" labelling. In most medical cases we often want to use more than two classes for the data. When labelling data based on multiple classes we use the metrics as shown in Figure 4.8, here we can see that for a given class we only have one option for True positive, and the diagonal and horizontal represent False positive and False Negative respectively. The rest of the options are True negative.

4.2.2 Common metrics

When evaluating our results, we use a set of common metrics used in the field of statistics and machine learning. The metrics we will be using in this thesis are Recall (REC), Precision (PREC), Specificity (SPEC), Accuracy (ACC), Matthews correlation coefficient (MCC), and F1 score (F1).

Accuracy: Accuracy is the percentage of the predictions that were classified correctly. It describes how many of our predictions were correct out of the total predictions made as shown in equation 4.1. It is the most common metric given its simplicity both in calculation and understanding. In general when our data is balanced, and we only have a few classes, we can get away with using accuracy. A pitfall with the accuracy metric is the lack of a comprehensive overview of the data, as it is just a summation

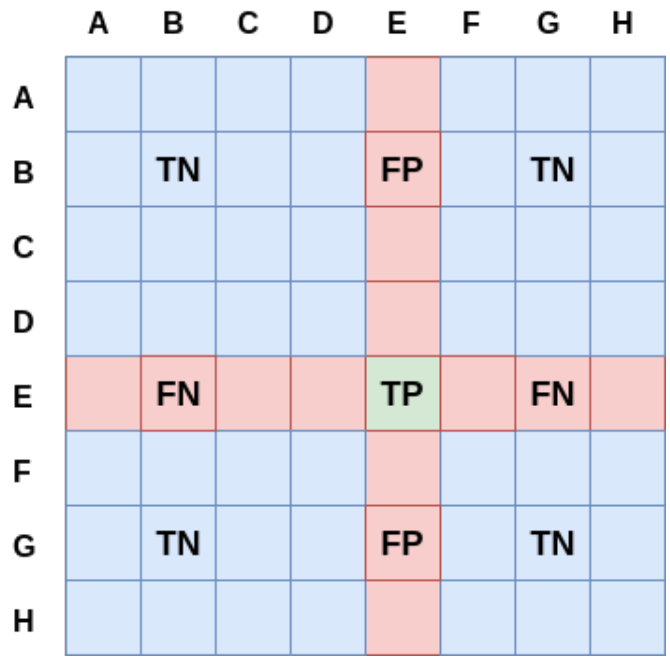


Figure 4.8: Confusion matrix with eight classes, here True positive is marked in green, False Negative and False positive marked in red, and True negative in blue.

without respect to classes involved. In this project, we use accuracy during the training step as a metric of success. After training is complete, we do not use accuracy as an indication of success given our unbalanced datasets.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Recall: Recall is the probability of detection, often called sensitivity. This metric is a measure of the fraction of relevant instances that have been retrieved over the total amount of relevant instances for a binary classification example. It is calculated using equation **??**.
Recall together with Specificity and Precision gives a complete view of the data compared to for instance accuracy alone. This difference is shown when we have uneven datasets. Given a dataset where we know that only 1% of the data is negative, we will get an accuracy of 99% if we label all the data as positive. When adding recall, specificity and precision, our score will reflect that we mislabeled all the negative values.

$$REC = \frac{TP}{TP + FN} \tag{4.2}$$

Specificity: Specificity measures the proportions of our samples that were correctly identified as negative, when the true class were also negative. Specificity is related to

recall as an opposite in the binary class example. Equation 4.3 shows the equation used for specificity.

$$TPR = \frac{TN}{TN + FP} \tag{4.3}$$

Precision: Precicion is the meassure of relevance in the binary classification case. As we can see from equation 4.4 the formula is similar to recall, but it only looks at the positive samples.

$$PREC = \frac{TP}{TP + FP} \tag{4.4}$$

F1 score: F1 score is a combination of precision and recall as shown in equation 4.5.

$$F1 = \frac{precision \times recall}{precision + recall} \tag{4.5}$$

Matthews correlation coefficient: Matthews correlation coefficient is a metric that takes all four possible states of TP, TN, FN and FP in to account. As with the F1 score, the Matthews correlation coefficient gives a score that is based on a more complete understanding of the data compared to how metrics like recall and precision only looks at a subset of the data.
Equation 4.6 shows the formula. It can output a score from -1 to 1, where 1 is a correct classification and -1 a total incorrect prediction. A score of 0 shows no statistical relevance in the result we have.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.6}$$

4.2.3 Singleclass vs Multiclass Metrics

The metrics presented are, in general, a solid way to present the validity of a model. However, not all metrics presented is the same when switching between single and multiclass classification. Metrics like Accuracy is designed to work for multiclass classification, given that there is only one way to calculate the score as shown in either equation 4.1 or in equation 4.7.

$$\frac{\sum(diag(CovarienceMatrix))}{\sum(CovarianceMatrix)} \tag{4.7}$$

The problem with multiclass metrics arises when there is more than one way to calculate the metrics needed, this can be for instance Recall and Specificity, where we have multiple ways to add together the class-wise scores. The three most common ways to calculate the average are:
Micro average: calculates the mean value of each of the binary metrics and averages the result over the total number of samples. Micro average ignores all class frequencies and gives us a metric based on all samples gathered. Micro-averaging may be preferred

in multilabel settings, including multiclass classification where a majority class is to be ignored.

Macro average: calculates the mean of each of the binary metrics, giving the same weight to each of the classes. Macro average gives importance to classes with few samples, and infrequent classes play the same roles as frequent ones. The disadvantage with Macro average is the fact that in the real world some classes often plays a more significant role than others, and doing especially bad on one of the classes can worsen the total result.

Weighted average calculates the mean of each of the binary metrics but gives a weighted sum for each of the scores before it is averaged. The weight of each class depends on the size of the true data samples. The weighted average gives us the advantage that small classes still count more than it would with for instance Micro average, but since it depends on the number of samples from each class it can end up more or less as a black box during calculation. Weighted average gives us the best of both worlds, but it lacks the intuitiveness from the two other classes.

With the three methods presented, we have chosen to Macro average our results. While both Macro and Weighted average would give a good indication given that not all our datasets are balanced, we argue that the weighted average would give metrics that are harder to explain when we are working on datasets with unbalanced classes.
In addition to looking at the Macro average of precision and recall, we want to look at specific cases of the classification. In many cases, we have multiple classes, where we are most interested in just one or a handful of the classes shown. For instance, a focus we have in this thesis is to give a score on how predictable polyp detection is, and on that case, we want to discuss the True positive rate (TPR) of the polyp detection and not the TPR of the non-polyp classes.
Take for instance the matrix 4.9

$$\begin{bmatrix} 10 & 1 \\ 3 & 12 \end{bmatrix}$$

Figure 4.9: Nearly balanced matrix

x

Here we can calculate the weighted average recall to be . This can be an interesting observation in itself, but often the first or second True label is much more important relative to the other. In a more practical example: We are more interested in finding areas with polyps when we know they are present, compared knowing there is not a polyp in an area when none are present.
These Metrics becomes a more prominent topic when it comes to inpainting. With

Name	Version
Ubuntu	18.04.2
Python	3.6.7
Tensorflow	1.13.0
Keras	2.2.4
CUDA	10.0.130
cuDNN	7

Table 4.1: Software specifications for our system

Category	Name
CPU	Intel i5-4590 CPU @ 3.30GHz
GPU	Nvidia GeForce GTX 1080 TI
RAM	Kingston 16 GB DDR3 @ 1600 MHz

Table 4.2: Hardware specifications for our system

inpainting, we take areas with no relevant information and makes it into areas that are similar to the rest of the image. Given that we can inpaint over polyps by mistake, or that we might train our classifiers to not look in certain areas when classifying, we have an interest if also comparing single cases of recall and precision included to the average values.

4.2.4 Hardware and Software setup

For the experiments we used the hardware setup shown in table 4.2. The software type and version for the experiments is shown in table 4.1.

4.3 Setup of experiments

As we recall, we proposed the following hypothesis.

Hypothesis H_0: When classifying images, we will get the best result when we have images with the least amount of sparse information. Hence by removing areas with sparse information, we will see an increase in classification performance compared to not removing areas. check if right

Hypothesis H_1: When training a classifier, we will get a higher mode of generalisation of our results when removing the dataset specific artefacts compared to not removing artefacts.

In this thesis we use the hypothesises as the basis to solve the problem statements. We divide our work into two parts, inpainting and classifying. First, we will look at the

process of inpainting in detail, and inspect the results we have. We will look at how parameters affect the results, and how different networks will differ in the generating process.

more

After a rundown of the generation of the custom datasets trough inpainting, we will show how the classification scores for each of the created dataset. Here we look at the datasets generated by inpainting and compare them with a base case.
Our primary goal in this thesis is to see if any of the generated datasets can help with classification. We will both compare the different areas inpainted, and the method used to inpaint the images. For the classification, we will use MCC as our primary metric for comparison, as described in section 4.2.2. As we described in the MCC paragraph, this metric gives us the best indication of success based on the imbalance of data in the CVC datasets. We chose to show the other metrics from section 4.2.2 as well, but they are only there to get a comprehensive picture of the results.

4.4 Results of the Inpainting

We will first take a look at the generation of the new datasets and the machine learning methods used to inpaint the problematic areas.
We can recall that based on Hypothesis H_0, we can assume that the removal of areas with sparse information we achieve a higher classification score. We have chosen our first dataset to contain the images from the Kvasir dataset where the black corners and edges inpainted. From this, we hypotosise that we will see an increase in classification when training and testing on the same dataset, as well when testing on a new dataset, given that the network has less sparseness to take into account.
Based on Hypothesis H_1, we predict assume that the removal of areas with dataset specific artefacts will result in a higher classification score. By removing the green squares in the corners and the text overlayed on top of the images, we predict that we get a higher classification score on previously unseen datasets compared to not removing dataset specific artefacts.
With Hypothesis H_0 and H_1 in mind, we present three types of datasets with two different generators to see if our hypothesises are correct.

BC: Black corner. **GS**: Green square. **BC+GS**: Black corner and Green square

Dataset labels	Size	Inpainted area	Generator network used
I - Base Case	256x256 px	-	-
II - Autoencoder with black corner	256x256 px	BC	Autoencoder
III - Autoencoder with green square	256x256 px	GS	Autoencoder
IV - Autoencoder with both	256x256 px	BC+GS	Autoencoder
V - GAN with black corner	256x256 px	BC	GAN
VI - GAN with green square	256x256 px	GS	GAN
VII - GAN with both	256x256 px	BC+GS	GAN
VIII - Base Case	512x512 px	-	-
IX - Autoencoder with black corner	512x512 px	BC	Autoencoder
X - Autoencoder with green square	512x512 px	GS	Autoencoder
XI - Autoencoder with both	512x512 px	BC+GS	Autoencoder
XII - GAN with black corner	512x512 px	BC	GAN
XIII - GAN with green square	512x512 px	GS	GAN
XIV - GAN with both	512x512 px	BC+GS	GAN

Table 4.3: Details of all datasets we generate in the experiments.

4.4.1 Black corners

When generating the two first datasets (II & V in Table 4.3), we used the mask shown in Figure 3.4b. Here we did two operations, first cropping, then inpainting. The result is shown in Figure 4.10 and Figure 4.11.
In Figure 4.10 present representative images from both datasets inpainting the four edges around the image. Figure 4.10 show images that, for both datasets, images are well inpainted, and a good portion could pass as real images. We can discern that, in general, the autoencoder dataset gives a much more blurry image compared to the dataset generated by the GAN.
Figure 4.11 show images that are more problematic. In Figure 4.11b the image generated by the Autoencoder has drawn its colour from the nearby white oversaturated area, and subsequently, it has misdrawn the corner. In Figure 4.11c we do not have the same problem, as it has drawn information from a larger part of the image, and hence not drawn the oversaturated area. Both models failed in getting the right colour for the green box when present. In general, we remove a lot of the areas with sparse information solely by cutting the black border of the image. The finished product is without any areas with sparse information, compared to for instance 4.10a, where 25% of the image contains no relevant information.

The training time for the AE regarding removing the black edges took under 4 hours. This wait, in the realm of machine learning, is fairly quick and based only on the training time, might be worth doing. The training time of the GAN was over two days. The result of the GAN is a significant improvement over the AE, but might not be worth it at 12 times the training time.

(a) Original polyp image

(b) AE generated polyp image

(c) GAN generated polyp image

(d) Original normal-z-line image

(e) AE generated normal-z-line image

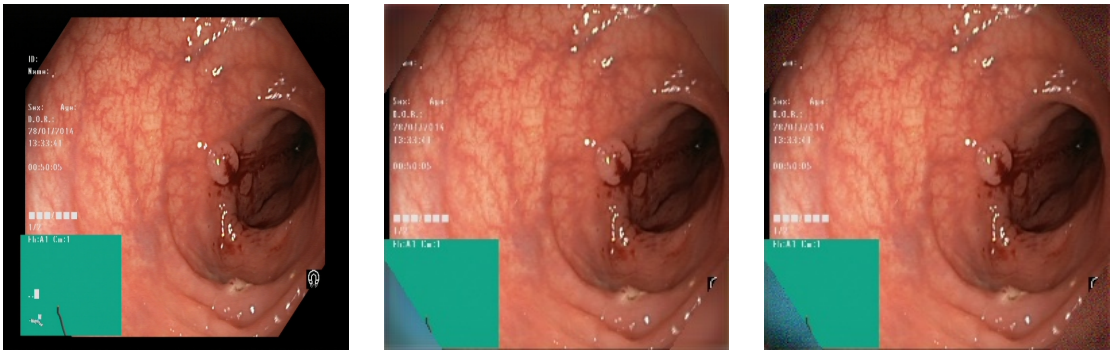
(f) GAN generated normal-z-line class image

Figure 4.10: Images from the polyp class and the z-line class. Both the AE and the GAN performed well in this scenario.

(a) Original ulcerative colitis image

(b) AE generated ulcerative colitis image

(c) GAN generated ulcerative colitis image



(d) Original polyp image

(e) AE generated polyp image

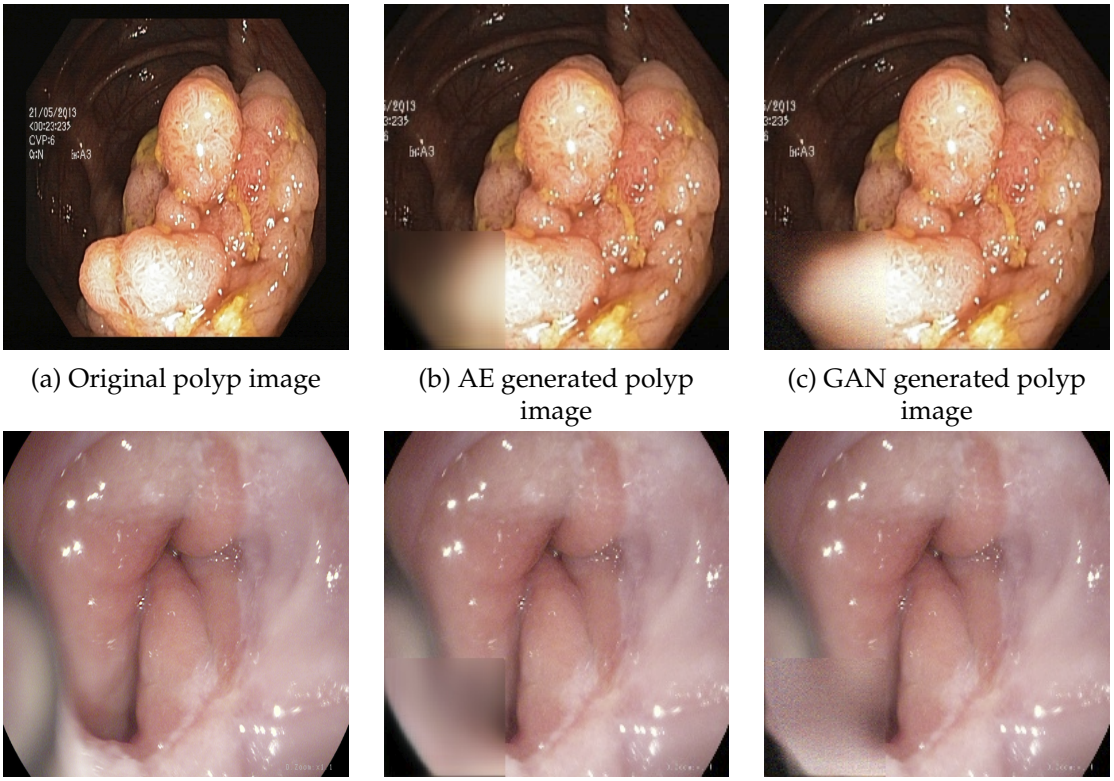
(f) GAN generated polyp image

Figure 4.11: Images from the polyp class and the ulcerative colitis. Here we see results that are not up to a good standard with regards to to light and to green colours.

4.4.2 Green square

The next two datasets (III & VI in Table 4.3) were generated from the mask in Figure 3.4c. The goal of the datasets generated by the Autoencoder and the GAN is to remove the dataset-specific green square. The same cropping-inpainting procedure were applied. The result is shown in Figure 4.12 and Figure 4.13.
Figure 4.12 and Figure 4.13 shows two examples from the datasets with the green square inpainted. Figure 4.12 show images that, for both datasets, images are challenging areas to get correct, and require more than just colour matching to pass as real images. In the first image, a large part of a polyp is covered, and we can see that both algorithms try to recreate the area to varying success. We can see that the GAN is much better, in both examples, to find the right colours at the edges, giving a more natural look. Another advantage with the GAN is the ability to estimate the corner size better, as we can see that the corner is unnaturally large in Figure 4.12e compared to Figure 4.12f.
Figure 4.13a,4.13b,4.13c shows how the two algorithms handles unexpected data. Here the green area was moved to the right to the point it was outside the mask range. Both algorithms added a green tint to the images, though the GAN did a better job making it natural looking. The last image set shows that the GAN has learned to connect structures throughout the inpainted area. The mucosa in Figure 4.13f the foreground is a full structure as we would expect if we removed the green square.

Just as with the black corners, the training time for the AE regarding removing the green square took about 4 hours. The training time of the GAN was over two days, closer to three days. However, the ability to predict structures and overall a greater awareness of context might be worth the training time.



(a) Original polyp image

(b) AE generated polyp image

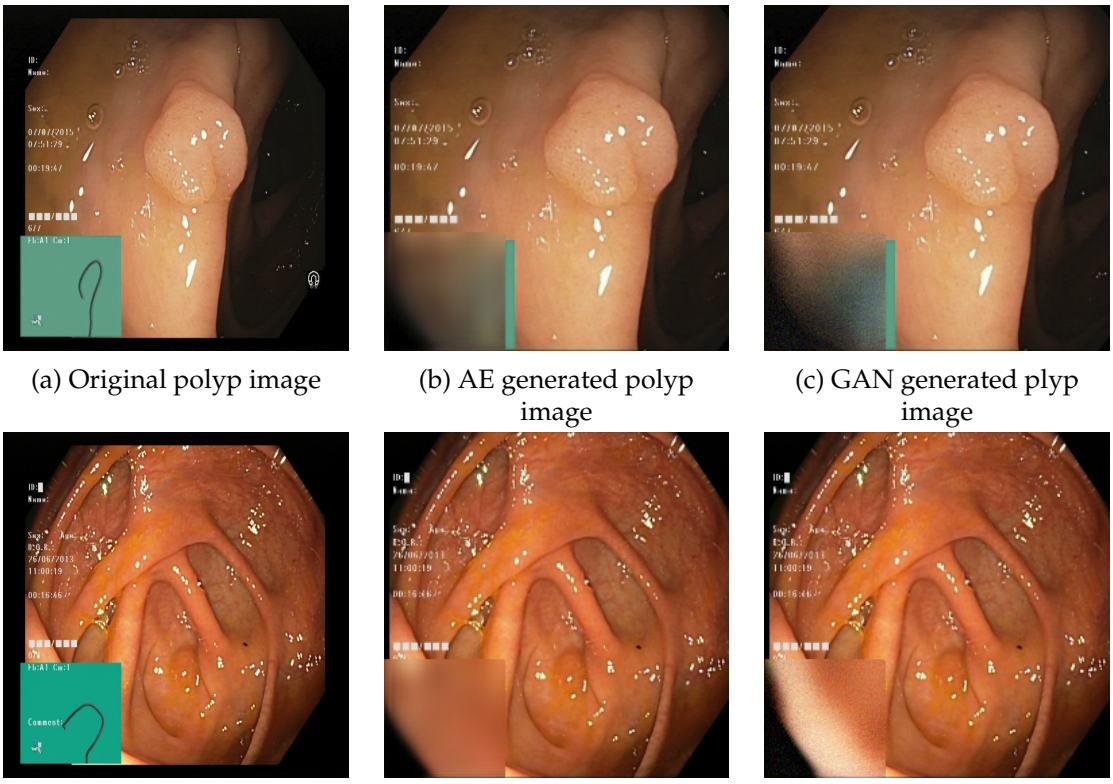
(c) GAN generated polyp image

(d) Original z-line image

(e) AE generated z-line image

(f) GAN generated z-line image

Figure 4.12: Images from the polyp class and the normal-z-line class. Here we see results that needed finer detail when inpainting.



(a) Original polyp image

(b) AE generated polyp image

(c) GAN generated plyp image

(d) Original normal-cecum image

(e) AE generated normal-cecum image

(f) GAN generated normal-cecum image

Figure 4.13: Images from the polyp class and the normal-cecum class. Here we have images with a problematic green square, and an image with details drawn from both sides of the inpainted area.

4.4.3 Combination

The last two datasets (IV & VII in Table 4.3) were generated from the mask in Figure 3.4d. The goal of the datasets generated by the AE and the GAN check how a combination of the previous datasets would do do to the classification. The same cropping-inpainting procedure were applied as with the other sets. The result is shown in Figure 4.14 and Figure 4.15.
Figure 4.14 and Figure 4.15 shows two examples from the datasets with the green square inpainted as well as the black corners inpainted. The first set of images in Figure 4.14 shows cases where both algorithms did reasonably well. Here, most of the job was cutting the images, and because the areas inpainted were homogeneous, the main task was to colour-match the areas.
Figure 4.15 show some more problematic areas. Image 4.15b show a downfall with the AE by the fact that it uses a green colour not present in any of the training images. Luckily the GAN in Figure 4.15c manages to use the dyed mucosa to better inpaint the area. We encounter the opposite problem in Figure 4.15d, 4.15e and 4.15f. Here the AE manages to not use the blue strip to inpaint the square green, while the GAN does. This mispainting by the GAN is probably caused by the blue ´´instrument" in the top left corner of the image. Here we believe the GAN misclassifies the image, assuming it is a dyed-lifted-polyp, and subsequently fills in the blue ink into the image.

The training time for the total AE inpainting was the longest, but still within 5 hours. Just as with the inpainted corners, the training time of the GAN was closer to three days. The results were reasonably better.

4.4.4 Double resolution

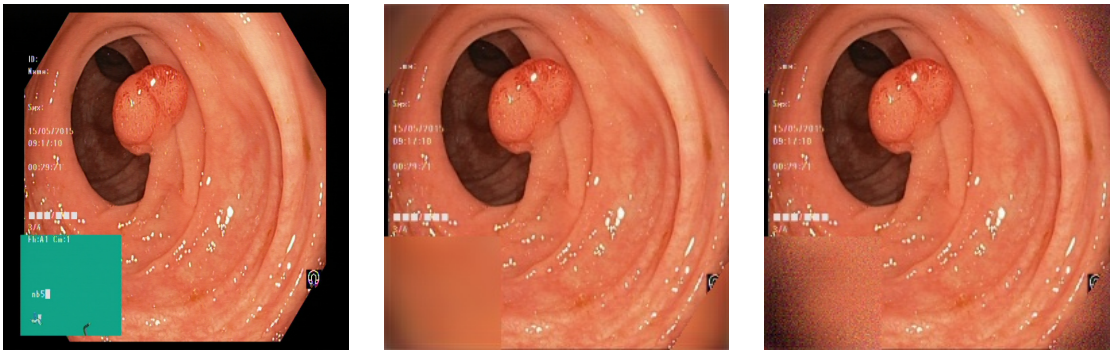
Make shure 512 resolution is described proppery before table tab:datasets

In addition to the datasets at standard size, we created the datasets at double resolution (IX & XIV in Table 4.3). The goal of the seven datasets generated at double resolution was to see how image size affects the results. We apply the same cropping-inpainting procedure as with the other sets. The result is shown in Figure 4.16.
As we see in figure 4.16 the inpainted areas is more smeared out in contrast to the previous datasets. In this thesis, we have chosen to keep our focus primarily on the images at 256×256 px resolution though we keep the double resolution images as a means to draw more context from our results.

(a) Original normal-pylorus image

(b) AE generated normal-pylorus image

(c) GAN gnerated normal-pylorus image

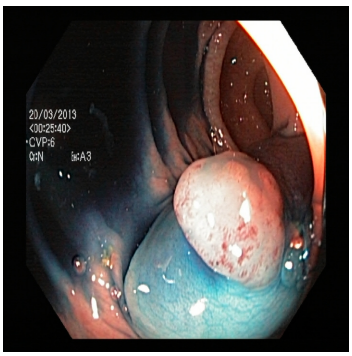


(d) Original polyp image

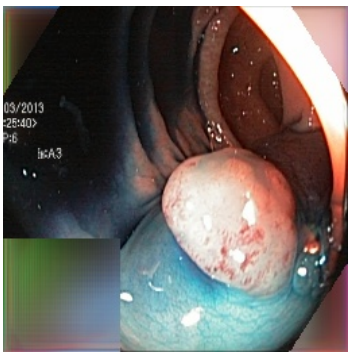
(e) AE generated polyp image

(f) GAN generated polyp image

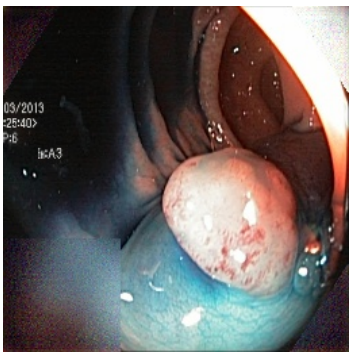
Figure 4.14: Images from the normal-pylorus an the polyp class. These images represent good images where most of the job was just to match the colour, rather than understanding complex structures in the images.



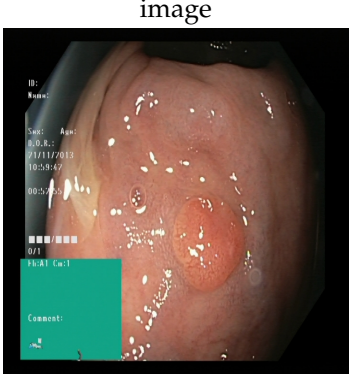
(a) Original dye lifted polyp image



(b) AE generated dye lifted polyp image



(c) GAN generated dye lifted polyp image

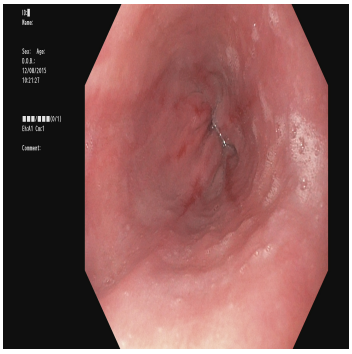


(d) Original polyp image

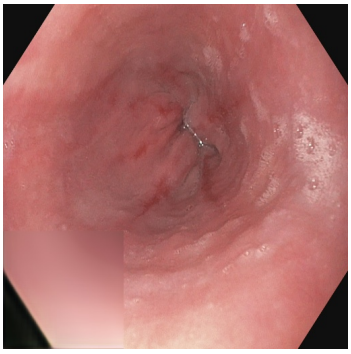
(e) AE generated polyp image

(f) GAN generated polyp image

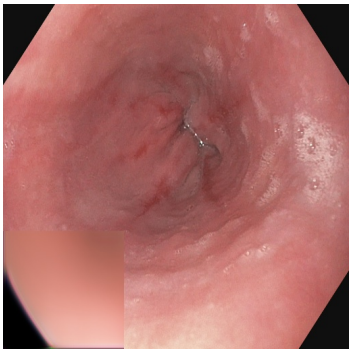
Figure 4.15: Images from the dye lifted polyp an the polyp class. The images were chosen because it highlighted flaws in both models.



(a) Original esophagitis image



(b) AE generated esophagitis image



(c) GAN gnerated esophagitis image

Figure 4.16: Images from the esophagitis class.The images from the double resolution dataset is much more smeared out compared to the smaller images.

4.5 Results of the transfer learning experiments

In the previous section, we looked at the generation of the new datasets. We looked at how the different hypothesises formed the basis for different masking, and we discussed downfall and advantages with the autoencoder method versus the GAN method. We will now look at the results when using the newly created datasets when training a classifier for each one of them. The model used for the classifier is the same for all the datasets. We use the same learning rate, the same model, and the same parameters for early stopping of the training. In this section we will first go through the model, then we will look at the results for each of the runs.

4.5.1 models

The classification model is constructed to give the best real-world correspondence.
First, we supply info about the model. This information is for instance batch size, type of network, and choice of an optimiser. The transfer learning network chosen is loaded with the imagenet weights. With the model loaded into memory, the global average pooling layer is added, followed by a fully connected layer with the number of nodes equal to the number of classes in the input dataset and a softmax activation step as described in more detail in chapter 3.3.
With the model loaded we complete it by adding the desired optimiser, and set parameters like batch size, learning rate, validation patience, and image size.
As described, we increase our confidence in our results by using K-fold cross-validation. This method ensures a realistic result using the theoretically best model for the task. When we are looking at the models, we use the base case results as a reference point improve upon. This base case is the result of a standard run of classification without any inpainting.

4.6 Densenet121

As stated in Section 3.3, we use the research provided by Borgli et al. in hyperparameter optimisation [37]. We concluded that Densenet121 is the optimal network to achieve the highest score when both training and evaluating on the Kvasir dataset.
Based on the results, we assume that Densenet121 will give the best results both when evaluating our network on the Kvasir dataset, and when we use the generalised models on the CVC datasets. The stats for the Densenet training routine is shown in Table 4.4.
The reasoning behind the training attributes for each of the parameters can be summarised with:
The max number of epochs is an artificial roof where we do not allow any more training, even when the network is improving. We believe that the network, should it come to 20 epochs, will only overfit the results. We have chosen training patience of three. This patience is a compromise between a higher value, more likely to overfit, and a lower value, more likely to stop too early to learn all the meaningful representations.

Table 4.4: Training attributes for Densenet121 base model

Atribute	Value
Max Number of epochs	20
Patience for validation	3
Folds	6
Image size	256x256
Batch size	24

In addition to the number of epochs trained we divided Kvasir into six folds. Based on the six folds we got 1333 images for both validation and testing, while the majority of the images would be used for training. A smaller fold would leave the training with too little data, and hence we could not expect as good of a result. A higher number of folds would increase the number of training runs substantially, since we are training on 14 datasets. Finally, the batch size was decided to be the highest number that did not give any memory errors. We believe that a batch size of 24 is well within the scope of reality.

4.6.1 Densenet121 base model

We present the results from the Densenet121 network in table 4.24. This Table shows the base case result with the Densenet121 model, meaning the unaugmented dataset.
The Matrices in Figure 4.17 shows the evaluation of the test set on each of the three datasets used. On the Kvasir dataset, we used five folds for training and validation, and the last sixth fold for the testing. That left us with 1342 images divided evenly between the classes. The CVC datasets were only used for training and hence did not need k fold cross-validation.

CVC 356 The CVC 356 base case MCC lies at 0.42 with an accuracy of 71%. From the Confusion matrix, we can discern that most polyps were classified correctly, except for the case that 37 of the images did not classify as a polyp when it was. Moreover, we also had 618 cases of non-polyps classified as polyps. In a vacuum, the results are adequate, showing a high accuracy on an unseen dataset.

The Kvasir In this Confusion matrix, we see that most of the samples lie throughout the diagonal, which, as we recall, is the correct classification. The Kvasir base case MCC lies at 0.92 which coincides with a near perfect accuracy given the complexity of the dataset. The misclassification is between the esophagitis class, and the normal-z-line class, which is already a notoriously different class do differentiate. There is

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	G
I	1310	37
G	618	319

(a) Baseline Confusion matrix for the CVC 356 dataset

	A	B	C	D	E	F	G	H
A	150	6	0	0	0	0	1	0
B	14	160	0	0	0	0	0	0
C	0	0	130	0	1	19	0	0
D	0	0	0	162	0	0	1	3
E	0	0	0	0	164	0	0	0
F	0	0	36	0	0	147	1	0
G	1	0	0	3	1	0	161	2
H	1	0	0	1	0	0	2	161

(b) Baseline Confusion matrix for the Kvasir dataset

	I	G
I	1311	3553
G	618	6472

(c) Baseline Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.4244
F1	0.6467
Precicion	0.7878
Recall	0.6565
Accuracy	0.7132

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.9202
F1	0.9299
Precicion	0.93
Recall	0.9313
Accuracy	0.93

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.2435
F1	0.5711
Precicion	0.6626
Recall	0.5912
Accuracy	0.6511

(f) The CVC 12k dataset Metrics

Figure 4.17: Densenet121 Base results

also a small mixup between dyed lifted polyps and dyed resection margins, another notoriously different classification task.

CVC 12k The CVC 12k base case MCC lies at 0.24 with an accuracy of 65%. From the Confusion matrix, we can discern that most polyps were classified correctly, though a third of the polyps were misclassified.

4.6.2 Corners inpainted result

Based on Hypothesis H_0 we would expect the act of inpainting the corner in the images to give Kvasir a higher score compared to the baseline. Also, we should see improvement in both CVC 356 and CVC 12k. We present the dataset created by the GAN followed by the dataset created by the Autoencoder evaluated with the Densenet121 model.

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	G
I	1647	179
G	281	177

(a) GAN corners Confusion matrix for the CVC 356 dataset

	A	B	C	D	E	F	G	H
A	157	12	0	0	0	0	0	0
B	8	154	0	0	0	0	0	0
C	0	0	116	0	0	11	0	0
D	0	0	0	162	0	0	3	5
E	0	0	1	0	166	0	3	0
F	0	0	49	0	0	155	1	0
G	1	0	0	2	0	0	158	2
H	0	0	0	2	0	0	1	159

(b) GAN corners Confusion matrix for the Kvasir dataset

	I	G
I	1648	4699
G	281	5326

(c) GAN corners Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.3184
F1	0.6562
Precicion	0.6757
Recall	0.6442
Accuracy	0.7986

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.914
F1	0.9233
Precicion	0.9239
Recall	0.9287
Accuracy	0.9239

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.2842
F1	0.5398
Precicion	0.6928
Recall	0.6048
Accuracy	0.5834

(f) The CVC 12k dataset Metrics

Figure 4.18: Densenet121 Inpainted corners with the GAN results

Figure 4.18 and 4.19 shows the evaluation of the test set on each of the three datasets made by both the Autoencoder and GAN.

CVC 356 We got the highest MCC from the Autoencoder with a score of 0.56 compared to a score of 0.31 from the GAN. In general, the Autoencoder has a higher score overall compared to the GAN, where the Autoencoder reaches a higher score than the base case, and the GAN does not. This difference indicates that the way the Autoencoder inpaints sparsly covered areas gives an edge in this use case.

The Kvasir When evaluating our results on the Kvasir dataset we see similar scores for both the GAN and Autoencoder with 0.91 and 0.92 respectively. Here the Autoencoder reaches a higher score than the base case, though the margin is too small to significant. We can discern that the number of small errors throughout the confusion matrices increases, indicating that the inpainted areas negatively help with classification.

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	G
I	1746	106
G	182	250

(a) AE corners Confusion matrix for the CVC 356 dataset

	A	B	C	D	E	F	G	H
A	150	9	0	0	0	0	1	0
B	12	157	0	0	0	0	0	0
C	0	0	141	0	0	21	0	0
D	1	0	0	160	0	0	4	2
E	0	0	0	0	166	0	1	0
F	0	0	25	0	0	145	0	0
G	3	0	0	3	0	0	157	2
H	0	0	0	3	0	0	3	162

(b) AE corners Confusion matrix for the Kvasir dataset

	I	G
I	1747	5227
G	182	4798

(c) AE corners Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.563
F1	0.7792
Precicion	0.8039
Recall	0.7607
Accuracy	0.8739

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.9226
F1	0.9321
Precicion	0.9322
Recall	0.9322
Accuracy	0.9322

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.2867
F1	0.516
Precicion	0.6921
Recall	0.607
Accuracy	0.5475

(f) The CVC 12k dataset Metrics

Figure 4.19: Densenet121 Inpainted corners with the AE results

CVC 12k The MCC scores for the GAN and Autoencoder both reaches a higher value than the base case. When inpainting the corners we see an increase in MCC value of 0.04 for both methods, giving some indication that removing areas with sparse information might give a higher classification score on other datasets with the same sparse areas.

4.6.3 Green square inpainted result

We can recall from Hypothesis H_1 that removing dataset specific artefacts we will achieve a higher classification score when evaluating our models on previously unseen datasets. In this experiment, we expect the Kvasir dataset not show any improvements, though both the CVC 356 and the CVC 12k dataset we hope to improve compared to the base case. We present the dataset created by the GAN followed by the dataset created by the Autoencoder and evaluate their performance with the Densenet model together.

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	G
I	1848	92
G	80	264

(a) GAN green square Confusion matrix for the CVC 356 dataset

	A	B	C	D	E	F	G	H
A	157	13	0	0	0	0	0	0
B	7	153	0	0	0	0	0	0
C	0	0	146	0	1	37	0	0
D	0	0	0	161	0	0	8	4
E	0	0	1	0	165	1	1	0
F	0	0	19	0	0	128	0	0
G	2	0	0	3	0	0	153	0
H	0	0	0	2	0	0	4	162

(b) GAN green square Confusion matrix for the Kvasir dataset

	I	G
I	1849	6850
G	80	3175

(c) GAN green square Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.71
F1	0.8549
Precicion	0.85
Recall	0.86
Accuracy	0.9247

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.9116
F1	0.9222
Precicion	0.9224
Recall	0.9237
Accuracy	0.9224

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.2275
F1	0.4131
Precicion	0.6376
Recall	0.594
Accuracy	0.4203

(f) The CVC 12k dataset Metrics

Figure 4.20: Densenet121 Inpainted green square with the GAN results

Figure 4.20 and 4.21 shows the evaluation of the test set on each of the three datasets used both for the GAN and Autoencoder.

CVC 356 As this is a dataset unseen by the classifier during training, we would expect, given that Hypothesis H_1 is True, that the classification score would be higher than the base case. Here we have *significantly* higher score with both the GAN and Autoencoder. The GAN reached an MCC score of 0.71 compared to the base case of 42. The Autoencoder also beat the base case with a large margin, given the MCC score of 0.60. For both models, we see an increase in the recall, suggesting the number of mismatches decreased.
The results highly suggest that our hypothesis about removing dataset-specific artefacts to improve accuracy is indeed correct.

The Kvasir As with the previous tests we see little change in the scores when evaluated on the Kvasir dataset. Here both the GAN and the Autoencoder got a

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	G
I	1746	106
G	182	250

(a) AE green square Confusion matrix for the CVC 356 dataset

	A	B	C	D	E	F	G	H
A	150	9	0	0	0	0	1	0
B	12	157	0	0	0	0	0	0
C	0	0	141	0	0	21	0	0
D	1	0	0	160	0	0	4	2
E	0	0	0	0	166	0	1	0
F	0	0	25	0	0	145	0	0
G	3	0	0	3	0	0	157	2
H	0	0	0	3	0	0	3	162

(b) AE green square Confusion matrix for the Kvasir dataset

	I	G
I	1747	5227
G	182	4798

(c) AE green square Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.6072
F1	0.8017
Precicion	0.8248
Recall	0.7837
Accuracy	0.8879

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.9158
F1	0.9262
Precicion	0.9262
Recall	0.9271
Accuracy	0.9262

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.2029
F1	0.4185
Precicion	0.6257
Recall	0.5819
Accuracy	0.4287

(f) The CVC 12k dataset Metrics

Figure 4.21: Densenet121 Inpainted green square with the AE results

lower score compared to the base, though only with a small margin. Just like with the experiment with the corners inpainted, inpainting the green square does little to nothing with classification when the classifier already overfits.

CVC 12k As with the CVC 356 dataset, the CVC 12k dataset is unseen by the classifier during training. Here, both the GAN and Autoencoder reaches a lower MCC value when evaluated on the dataset. This result is a direct contradiction to the hypothesis H_1, showing that removing dataset specific artefacts does not always yield better scores. It is important to note that there are significant differences between the two CVC datasets which can influence the results. The most prominent feature is the edges around the images in the 12k dataset. The much larger frames could ´´overwrite" much of the work done by inpainting the green square, and could subsequently disrupt the classifier.

4.6.4 Combined corners and dataset specific artefacts inpainted result

In addition to testing the two hypothesises separately we, as we recall, want to check both types of inpainting at the same time. When testing both inpainting types, we do not draw any predictions, given that we do not know if the two methods interfere with each other.
We present the GAN inpainting of both the areas followed by the same inpainting with the Autoencoder.

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	*G*
I	1916	129
G	12	227

(a) GAN both areas Confusion matrix for the CVC 356 dataset

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*
A	159	7	0	0	0	0	2	1
B	7	158	0	0	0	0	0	0
C	0	0	118	0	0	8	0	0
D	0	0	0	161	0	0	3	4
E	0	0	1	0	164	0	3	0
F	0	0	47	0	0	158	1	0
G	0	0	0	0	1	0	154	0
H	0	1	0	5	1	0	3	161

(b) GAN both areas Confusion matrix for the Kvasir dataset

	I	*G*
I	1917	7851
G	12	2174

(c) GAN both areas Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.7483
F1	0.8638
Precicion	0.8157
Recall	0.9434
Accuracy	0.9383

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.9192
F1	0.9278
Precicion	0.9285
Recall	0.9339
Accuracy	0.9285

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.2005
F1	0.3419
Precicion	0.6053
Recall	0.5954
Accuracy	0.3422

(f) The CVC 12k dataset Metrics

Figure 4.22: Densenet121 Inpainted both areas with the GAN results

Figure 4.22 and 4.23 shows the evaluation of the test set on each of the three datasets used both for the GAN and Autoencoder.

CVC 356 Evaluating on the CVC 356 dataset we see the most significant increase in the MCC score compared to the other models. We have, compared to the base case, a 76% increase in MCC score, from 0.42 to 0.74 This increase shows that there might be some viability to do more inpainting when training a model for unseen data. In addition to getting the highest score for evaluation, we also got one of the lowest

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	G
I	1889	238
G	39	118

(a) AE both areas Confusion matrix for the CVC 356 dataset

	A	B	C	D	E	F	G	H
A	149	11	0	0	0	0	1	0
B	11	155	0	0	0	0	0	0
C	0	0	143	0	0	25	1	0
D	0	0	0	156	0	0	0	2
E	0	0	0	0	166	1	2	1
F	0	0	23	0	0	140	0	0
G	4	0	0	5	0	0	154	3
H	2	0	0	5	0	0	8	160

(b) AE both areas Confusion matrix for the Kvasir dataset

	I	G
I	1890	7452
G	39	2573

(c) AE both areas Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.4462
F1	0.6959
Precicion	0.6556
Recall	0.8198
Accuracy	0.8787

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.9097
F1	0.9209
Precicion	0.9209
Recall	0.9213
Accuracy	0.9209

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.2105
F1	0.3713
Precicion	0.6182
Recall	0.5937
Accuracy	0.3733

(f) The CVC 12k dataset Metrics

Figure 4.23: Densenet121 Inpainted both areas with the AE results

scores too. The main difference in score is most likely the result of the precision of the autoencoder being much lower than the precision of the GAN. The difference in score can also be the result of variation within the K-fold transfer learning. Based on previous research done by us we have shown that this method had the highest standard deviation for both generator models followed by the base case [57]. This can indicate that we should not rely on the result given its high variance.

This cite is incomplete

The Kvasir As with the earlier tests, we see little change in the scores when evaluating on the Kvasir dataset. Here we can see that the MCC values are within a margin to close to draw any firm conclusions.

CVC 12k As with the inpainted square example, the MCC values when inpainting the largest possible area is lower than the base case. For the Densenet model, on the CVC 12k set, removing artefacts gives not any advantages compared no not removing the artefacts. Here we can notice that the recall is constant between each model and the base

case, while the precision decreases for both inpainted datasets. This drop in precision can indicate that the inpainted models less frequently gets its predictions correct.

Snakk med noen om dette

4.7 InceptionResNetV2

In addition to just testing our results with the Densenet121, we wanted to see if the results were replicable with other networks. We present our result from InceptionResNetV2 (IRV2) as a base of comparison versus the Densenet121 model. As mentioned in the metodology, InceptionResnetV2 is a model with more parameters, and hence needs more time to train compared to the Densenet architecture. For the IRV2 model, we used greater patience while training to ensure that the model did not underfit, or skipped essential features when training. The stats for the IRV2 training is shown in Table 4.5.

MICHAEL OR PAAL: This is a bit redundant, but might help people whi skips chapters.. thoughts?

Table 4.5: Training attributes for Inceptionresnetv2 base model

Atribute	Value
Max Number of epochs	20
Patience for validation	4
Folds	6
Image size	256x256
Batch size	24

From the previous tests in done on the same datasets with the IRV2 network, we had gotten inconclusive results when the training patience was too low [57]. With the patience of four we, on average, train longer and hence get a more stable result without overfitting.

4.7.1 InceptionresNetV2 base model

We present the results with InceptionResNetV2 in table 4.24. This Table shows the base case result with the larger Inceptionresnetv2 model.
The Matrices in Figure 4.24 shows the evaluation of the test set on each of the three datasets used. As with the Densenet model we used k-fold cross-validation with six folds for the Kvasir set and evaluated on the CVC sets.

CVC 356 The CVC 356 base case MCC lies at 0.20 with an accuracy of 61%. Compared to the Densenet model we lost half of our MCC score and 10From the Confusion matrix, we can discern that the model had trouble finding polyps with only 243 of the 356 total polyps found. The results from this model are arguable to low to be used successfully as a classifier.

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	G
I	1142	113
G	786	243

(a) Baseline Confusion matrix for the CVC 356 dataset

	A	B	C	D	E	F	G	H
A	132	2	0	0	0	0	1	0
B	31	163	0	0	0	0	0	0
C	0	0	124	0	0	15	0	0
D	0	0	0	160	0	0	4	6
E	0	0	2	0	166	1	1	1
F	0	0	39	0	0	150	1	0
G	3	0	0	1	0	0	146	1
H	0	1	1	5	0	0	13	158

(b) Baseline Confusion matrix for the Kvasir dataset

	G	I
I	1142	4857
G	787	5168

(c) Baseline Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.2005
F1	0.5342
Precicion	0.6375
Recall	0.5731
Accuracy	0.6064

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.89
F1	0.902
Precicion	0.9029
Recall	0.9083
Accuracy	0.9029

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.0791
F1	0.4675
Precicion	0.5538
Recall	0.5291
Accuracy	0.5279

(f) The CVC 12k dataset Metrics

Figure 4.24: InceptionResNetV2 Base results

The Kvasir As with the Densenet model we have scores close to the highest MCC possible. This score, as previous, shows that the model most likely is overfitting to the data. The misclassification is between the esophagitis class, and the normal-z-line class, as well as a small mixup between dyed lifted polyps and dyed resection margins.

CVC 12k The InceptionResNetV2 model evaluated on the CVC 12k dataset gives a relatively low MCC score of 0.07. The score gives us a good indication that the model did not learn the dataset-specific features that define the dataset, and hence gave us such a low score.

4.7.2 Inceptionresnetv2 Inpainted Corners

For the first comparison, we look at the dataset with the inpainted corner. We present both the work done by the GAN and the Autoencoder.
Figure 4.25 and 4.26 shows the evaluation of the test set on each of the three datasets made by both the Autoencoder and GAN.

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	*G*
I	1489	188
G	439	168

(a) GAN corners Confusion matrix for the CVC 356 dataset

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*
A	155	8	0	0	0	0	2	0
B	10	158	0	0	0	0	0	0
C	0	0	133	0	0	43	0	2
D	0	0	0	162	0	0	3	1
E	0	0	1	0	166	0	8	0
F	0	0	31	0	0	123	1	0
G	0	0	0	2	0	0	143	0
H	1	0	1	2	0	0	9	163

(b) GAN corners Confusion matrix for the Kvasir dataset

	I	*G*
I	1490	3507
G	439	6518

(c) GAN corners Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.2005
F1	0.5875
Precicion	0.6221
Recall	0.5823
Accuracy	0.7255

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.8927
F1	0.9056
Precicion	0.9059
Recall	0.9072
Accuracy	0.9059

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.3152
F1	0.5989
Precicion	0.7113
Recall	0.6175
Accuracy	0.6699

(f) The CVC 12k dataset Metrics

Figure 4.25: InceptionResNetV2 Inpainted corners with the GAN results

CVC 356 Evaluating our GAN generated model gives us an MCC value equal to the baseline. We can see a greater accuracy, F1, precision and recall score with this model, though the skewness of the dataset causes it. The AE MCC value here is Negative, indicating that the model did worse than pure guessing.

The Kvasir When evaluating our results on the Kvasir dataset we see similar scores for both the GAN and Autoencoder with 0.892 and 0.891 respectively. The same potential overfitting gives us no clear indication if any of our hypothesises are valid.

CVC 12k Both models beat the baseline for the CVC 12k MCC value. Here we can see clearly that the GAN has gotten results on par with the Densenet model.

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	*G*
I	225	49
G	1703	307

(a) AE corners Confusion matrix for the CVC 356 dataset

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*
A	158	23	0	1	0	0	0	1
B	8	143	0	0	0	0	1	1
C	0	0	107	0	0	11	0	0
D	0	0	0	160	0	0	8	4
E	0	0	0	0	163	0	0	1
F	0	0	58	0	1	155	0	0
G	0	0	0	1	2	0	157	2
H	0	0	1	4	0	0	0	157

(b) AE corners Confusion matrix for the Kvasir dataset

	I	*G*
I	225	491
G	1704	9534

(c) AE corners Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	-0.0234
F1	0.2319
Precicion	0.4895
Recall	0.487
Accuracy	0.2329

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.8913
F1	0.9026
Precicion	0.9036
Recall	0.9114
Accuracy	0.9036

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.1049
F1	0.5335
Precicion	0.5338
Recall	0.5813
Accuracy	0.8164

(f) The CVC 12k dataset Metrics

Figure 4.26: InceptionResNetV2 Inpainted corners with the AE results

4.7.3 Inceptionresnetv2 Inpainted Square

Here we present the results from the dataset with the inpainted green square, both with the Autoencoder and the GAN. Based on our hypothesis we expect the same improvement here as we had in our Densenet model.

Figure 4.27 and 4.28 shows the evaluation of the test set on each of the three datasets used both for the GAN and Autoencoder when the green square was inpainted.

CVC 356 Evaluating our GAN generated model gives us an MCC value of 0.57 which is close to thee times the MCC value of the baseline. This score highly indicates that our hypothesis H_1 holds ground. Similarly, our AE generated model gives us the MCC score of 0.43. Though not as good, it is still a doubling of the baseline. The results here gives us a strong belief that, even with unoptimised datasets, we can get merit from removing dataset specific artefacts.

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	G
I	1871	174
G	57	182

(a) GAN square Confusion matrix for the CVC 356 dataset

	A	B	C	D	E	F	G	H
A	147	9	0	0	0	0	1	0
B	15	157	0	0	0	0	0	1
C	0	0	135	0	0	19	0	0
D	1	0	0	165	0	0	18	18
E	0	0	0	0	166	4	3	0
F	0	0	31	0	0	143	0	0
G	2	0	0	0	0	0	140	2
H	1	0	0	1	0	0	4	145

(b) GAN square Confusion matrix for the Kvasir dataset

	I	G
I	1872	7821
G	57	2204

(c) GAN square Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.5708
F1	0.7768
Precicion	0.7408
Recall	0.8382
Accuracy	0.8989

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.8888
F1	0.9019
Precicion	0.9021
Recall	0.9064
Accuracy	0.9021

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.1788
F1	0.3405
Precicion	0.5952
Recall	0.584
Accuracy	0.341

(f) The CVC 12k dataset Metrics

Figure 4.27: InceptionResNetV2 Inpainted square with the GAN results

The Kvasir As with the previous tests we see little change in the scores when evaluated on the Kvasir dataset. Here, as with the corners inpainted, both the GAN and the Autoencoder got a lower score compared to the base with the same small margin.

CVC 12k The results from the CVC 12k dataset is here similar to the Densenet model.

more when im not tired

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	G
I	1784	195
G	144	161

(a) AE square Confusion matrix for the CVC 356 dataset

	A	B	C	D	E	F	G	H
A	149	12	0	0	0	0	2	0
B	16	154	2	0	0	0	0	0
C	0	0	138	0	0	40	0	0
D	0	0	0	158	0	0	14	7
E	0	0	0	0	159	1	3	0
F	0	0	22	0	2	116	0	0
G	1	0	1	5	1	4	142	2
H	0	0	3	3	4	5	5	157

(b) AE square Confusion matrix for the Kvasir dataset

	G	I
I	1785	6055
G	144	3970

(c) AE square Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.4026
F1	0.7002
Precicion	0.6888
Recall	0.7147
Accuracy	0.8516

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.867
F1	0.8822
Precicion	0.8833
Recall	0.8836
Accuracy	0.8833

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.2488
F1	0.4635
Precicion	0.6607
Recall	0.5963
Accuracy	0.4814

(f) The CVC 12k dataset Metrics

Figure 4.28: InceptionResNetV2 Inpainted square with the AE results

4.7.4 Inceptionresnetv2 both Inpainted

Here we present the results from the dataset with the maximum inpainted area. Figure 4.29 and 4.30 shows the evaluation of the test set on each of the three datasets used both for the GAN and Autoencoder.

CVC 356

The Kvasir

CVC 12k

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	*G*
I	295	37
G	1633	319

(a) GAN both areas Confusion matrix for the CVC 356 dataset

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*
A	143	2	0	0	0	0	0	0
B	19	164	0	0	0	0	0	0
C	0	0	138	0	0	22	0	0
D	0	0	0	163	0	0	6	3
E	0	0	2	0	166	1	3	1
F	0	0	26	0	0	143	0	0
G	3	0	0	1	0	0	152	0
H	1	0	0	2	0	0	5	162

(b) GAN both areas Confusion matrix for the Kvasir dataset

	I	*G*
I	295	478
G	1634	9547

(c) GAN both areas Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.0505
F1	0.2687
Precicion	0.5245
Recall	0.5260
Accuracy	0.2688

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.8718
F1	0.8864
Precicion	0.8870
Recall	0.8918
Accuracy	0.8870

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.1574
F1	0.5594
Precicion	0.5526
Recall	0.6177
Accuracy	0.8233

(f) The CVC 12k dataset Metrics

Figure 4.29: InceptionResNetV2 Inpainted both areas with the GAN results

A:dyed-lifted-polyps , **B**:dyed-resection-margins , **C**:esophagitis , **D**:normal-cecum , **E**:normal-pylorus , **F**:normal-z-line , **G**:polyps , **H**:ulcerative-colitis , **I**:non-polyp

	I	*G*
I	1167	89
G	761	267

(a) AE both areas Confusion matrix for the CVC 356 dataset

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*
A	141	3	0	0	0	0	4	0
B	24	163	0	2	0	0	0	0
C	0	0	130	0	1	15	0	0
D	0	0	0	148	0	0	1	2
E	0	0	0	0	163	0	0	0
F	0	0	36	0	0	151	0	0
G	1	0	0	14	1	0	160	7
H	0	0	0	2	1	0	1	157

(b) AE both areas Confusion matrix for the Kvasir dataset

	I	*G*
I	1167	3061
G	762	6964

(c) AE both areas Confusion matrix for the CVC 12k dataset

CVC 356 dataset	
MCC	0.2590
F1	0.5594
Precicion	0.6776
Recall	0.5944
Accuracy	0.6278

(d) The CVC 356 dataset Metrics

Kvasir dataset	
MCC	0.9097
F1	0.9209
Precicion	0.9209
Recall	0.9213
Accuracy	0.9209

(e) The Kvasir dataset Metrics

CVC 12k dataset	
MCC	0.9017
F1	0.9134
Precicion	0.9134
Recall	0.9178
Accuracy	0.9134

(f) The CVC 12k dataset Metrics

Figure 4.30: InceptionResNetV2 Inpainted both areas with the AE results

4.8 Classification results based on the Densenet model

Tables 4.6 to 4.8 shows the summary of the results gathered from the Densenet model, and figure Figures 4.31 to 4.33 shows the visualised MCC values.

The CVC 356 dataset (Fig: 4.6 & Bar: 4.31a) Here we can see a considerable improvement when inpainting in general. The results coincide well with our hypothesis H_1, and we can argue that hypothesis H_0 also holds some ground when evaluated on this dataset.

The Kvasir dataset (Fig: 4.7 & Bar: 4.32a) Here we do not expect the same result as we would for a dataset not seen by the classifier. We hypothesised that model **III** and **VI** would get us the best results, as they only removed areas with sparse information. From the results, we can see no significant improvement to the results when evaluated on the datasets. Thus we can not prove hypothesis H_0 with the densenet model.

The CVC 12k dataset (Fig: 4.8 & Bar: 4.33a) Here we see the most significant change when we removed sparse areas at the corner as opposed to removing dataset specific artefacts. As the MCC rose with 0.08 for both the datasets and no others, it seems like removing dataset specific artefacts does not always yield the best results.

Might need to talk about recall and prec and stuff... In how much detail do i want to go??

4.9 Classification results based on the InceptionResnetV2 model

Tables 4.9 to 4.11 shows the summary of the results gathered from the Densenet model, and figure Figures 4.31 to 4.33 shows the visualised MCC values.

The CVC 356 dataset (Fig: 4.9 & Bar: 4.31b) Here we can see the same considerable improvement when inpainting the *dataset-specific artefacts* as we saw with the densenet model. We see that we can, with this model, double the MCC score with the GAN dataset. A significant difference in the InceptionResNetV2 model compared to the Densenet model is the fact that some of the results did considerably worse than the base case.

The Kvasir dataset (Fig: 4.10 & Bar: 4.32b) The Kvasir dataset shows minimal to no improvement. We can conclude that when inpainting to only remove sparseness the the machine learining algortithms draws no benefit from it, independently of the model used.

I: Base Case. **II**: GAN Green square. **III**: GAN Black corner. **IV**: GAN Both inpainted. **V**: AE Green square. **VI**: AE Black corner. **VII**: AE Both inpainted.

Table 4.6: DenseNet121 CVC 356

Dataset	MCC	F1	Precision	Recall	Accuracy
I	0.4244	0.6467	0.7878	0.6565	0.7132
II	0.7100	0.8549	0.8500	0.8600	0.9247
III	0.3184	0.6562	0.6757	0.6442	0.7986
IV	0.7483	0.8638	0.8157	0.9434	0.9383
V	0.6072	0.8017	0.8248	0.7837	0.8879
VI	0.5630	0.7792	0.8039	0.7607	0.8739
VII	0.4462	0.6959	0.6556	0.8198	0.8787

Table 4.7: DenseNet121 CVC 356

Dataset	MCC	F1	Precision	Recall	Accuracy
I	0.9140	0.9233	0.9239	0.9287	0.9239
II	0.9116	0.9222	0.9224	0.9237	0.9224
III	0.9140	0.9233	0.9239	0.9287	0.9239
IV	0.9192	0.9278	0.9285	0.9339	0.9285
V	0.9158	0.9262	0.9262	0.9271	0.9262
VI	0.9226	0.9321	0.9322	0.9322	0.9322
VII	0.9097	0.9209	0.9209	0.9213	0.9209

Table 4.8: DenseNet121 CVC 356

Dataset	MCC	F1	Precision	Recall	Accuracy
I	0.2435	0.5711	0.6626	0.5912	0.6511
II	0.2275	0.4131	0.6376	0.5940	0.4203
III	0.2842	0.5398	0.6928	0.6048	0.5834
IV	0.2005	0.3419	0.6053	0.5954	0.3422
V	0.2029	0.4185	0.6257	0.5819	0.4287
VI	0.2867	0.5160	0.6921	0.6070	0.5475
VII	0.2105	0.3713	0.6182	0.5937	0.3733

The CVC 12k dataset (Fig: 4.11 & Bar: 4.33b) For the CVC 12k dataset every model shows improvement compared to the base case, with the highest score for the GAN removing dataset specific features. Though all the scores are low compared to the CVC 356 dataset, we can get double MCC when using inpainting.

4.10 Classification results based on the Densenet model at double size

In addition to the two models tested with the generated dataset, we also looked at what would happen at larger resolutions. We made the same datasets as in the previous section but at twice the size. The results can be found in table 4.14 with the visualised examples in Figure 4.34.
We can see from this summary that the results seem to keep the same trend at the larger size. It is important to not that the image quality is harder to maintain at double size.

The CVC 356 dataset (Fig: 4.12 & Bar: 4.34a) At double the size the inpaining results tend to have the same shape as with the smaller size, except for the fully inpainted dataset produced by the GAN. The baseline were also a bit higher at double size, while we would believe that it would be lower due to the non-polyp images from the CVC 356 set is around 288 $\times$ 384 px, meaning that we interpolate the images to get them to the appropriate size. The fact that we interpolate only the non-polyp images might have helped the classifier due to now different quality of the images.

The CVC 12k dataset (Fig: 4.14 & Bar: 4.34b) In the CVC 12k dataset all images are at the size 288 $\times$ 384 px. The upscaling of the images to 512 $\times$ 512 px distorts all the images, curiously enough the baseline is higher compared to the 256 $\times$ 256 px images. Figure 4.31

I: Base Case. **II**: GAN Green square. **III**: GAN Black corner. **IV**: GAN Both inpainted. **V**: AE Green square. **VI**: AE Black corner. **VII**: AE Both inpainted.

Table 4.9: IRV21 CVC 356

Dataset	MCC	F1	Precision	Recall	Accuracy
I	0.2004	0.5342	0.6375	0.5731	0.6064
II	0.5708	0.7768	0.7408	0.8382	0.8989
III	0.2005	0.5875	0.6221	0.5823	0.7255
IV	0.0505	0.2687	0.5245	0.5260	0.2688
V	0.4026	0.7002	0.6888	0.7147	0.8516
VI	-0.0234	0.2319	0.4895	0.4870	0.2329
VII	0.2590	0.5594	0.6776	0.5944	0.6278

Table 4.10: IRV2 Kvasir

Dataset	MCC	F1	Precision	Recall	Accuracy
I	0.8900	0.9020	0.9029	0.9083	0.9029
II	0.8888	0.9019	0.9021	0.9064	0.9021
III	0.8927	0.9056	0.9059	0.9072	0.9059
IV	0.8718	0.8864	0.8870	0.8918	0.8870
V	0.8670	0.8822	0.8833	0.8836	0.8833
VI	0.8913	0.9026	0.9036	0.9114	0.9036
VII	0.9017	0.9134	0.9134	0.9178	0.9134

Table 4.11: IRV2 CVC 12k

Dataset	MCC	F1	Precision	Recall	Accuracy
I	0.0791	0.4675	0.5538	0.5291	0.5279
II	0.1788	0.3405	0.5952	0.5840	0.3410
III	0.3152	0.5989	0.7113	0.6175	0.6699
IV	0.1574	0.5594	0.5526	0.6177	0.8233
V	0.2488	0.4635	0.6607	0.5963	0.4814
VI	0.1049	0.5335	0.5338	0.5813	0.8164
VII	0.2305	0.5819	0.6498	0.5887	0.6802

I: Base Case. **II**: GAN Green square. **III**: GAN Black corner. **IV**: GAN Both inpainted. **V**: AE Green square. **VI**: AE Black corner. **VII**: AE Both inpainted.

Table 4.12: DN121 512 × 512px CVC 356

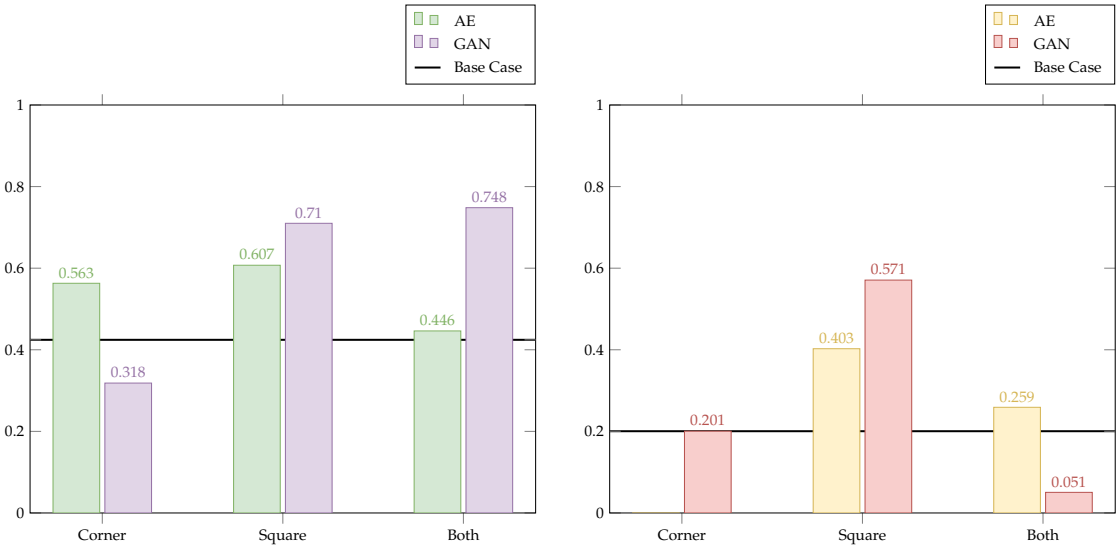
Dataset	MCC	F1	Precision	Recall	Accuracy
I	0.5908	0.7948	0.8074	0.7839	0.8875
II	0.6946	0.8306	0.7766	0.9360	0.9264
III	0.3850	0.6602	0.7481	0.6494	0.7526
IV	0.4502	0.7102	0.7709	0.6871	0.8104
V	0.5401	0.7680	0.7902	0.7512	0.8682
VI	0.4655	0.7174	0.7804	0.6932	0.8148
VII	0.5707	0.7812	0.8152	0.7583	0.8717

Table 4.13: DN121 512 × 512px Kvasir

Dataset	MCC	F1	Precision	Recall	Accuracy
I	0.9215	0.9303	0.9307	0.9345	0.9307
II	0.9118	0.9222	0.9224	0.9251	0.9224
III	0.9068	0.9181	0.9179	0.9222	0.9179
IV	0.9158	0.9261	0.9262	0.9269	0.9262
V	0.9091	0.9200	0.9202	0.9218	0.9202
VI	0.9209	0.9306	0.9307	0.9310	0.9307
VII	0.9218	0.9313	0.9315	0.9319	0.9315

Table 4.14: DN121 512 × 512px CVC 12k

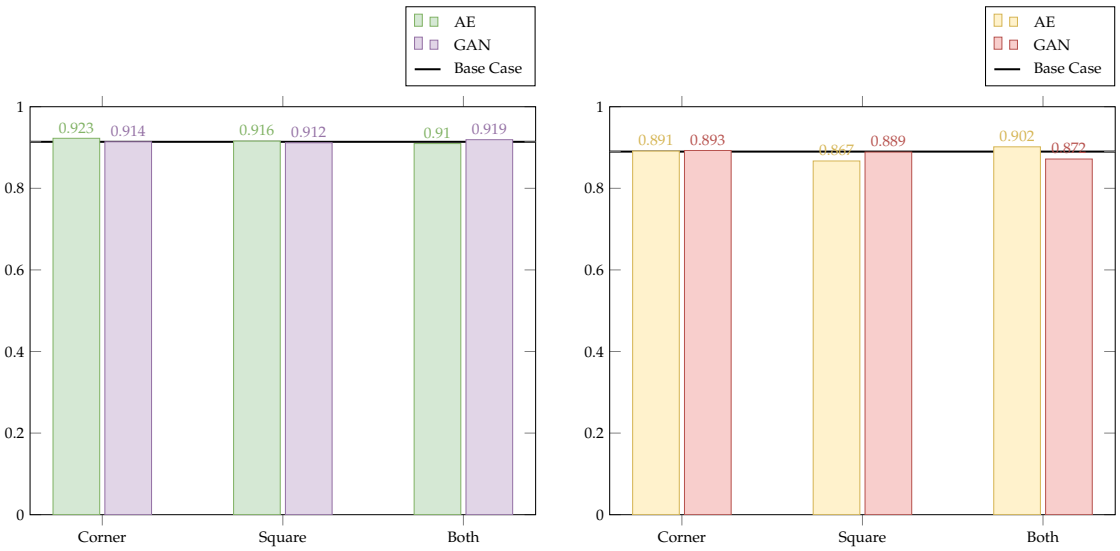
Dataset	MCC	F1	Precision	Recall	Accuracy
I	0.3146	0.5342	0.7118	0.6168	0.5682
II	0.2804	0.4473	0.6743	0.6127	0.4576
III	0.2256	0.5299	0.6533	0.5830	0.5846
IV	0.2833	0.5506	0.6925	0.6042	0.6005
V	0.2613	0.4909	0.6729	0.5987	0.5167
VI	0.2325	0.5023	0.6565	0.5863	0.5387
VII	0.2423	0.4748	0.6590	0.5924	0.4975



(a) CVC356 with DN121

(b) CVC356 with IRV2

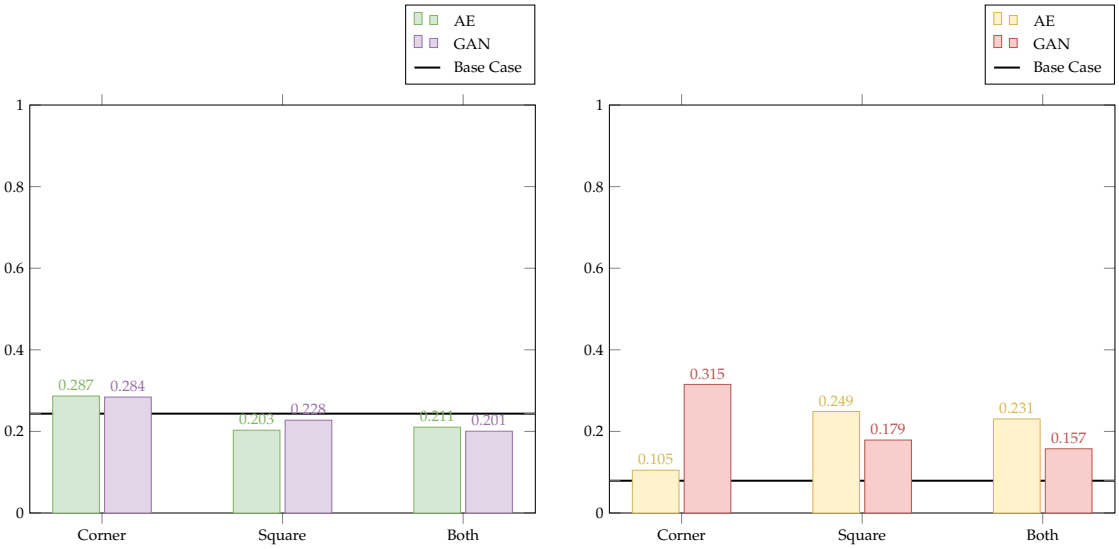
Figure 4.31: Visualisation of the CVC356 dataset MCC values made by both Densenet121 and InceptionresnetV2



(a) Kvasir with DN121

(b) Kvasir with IRV2

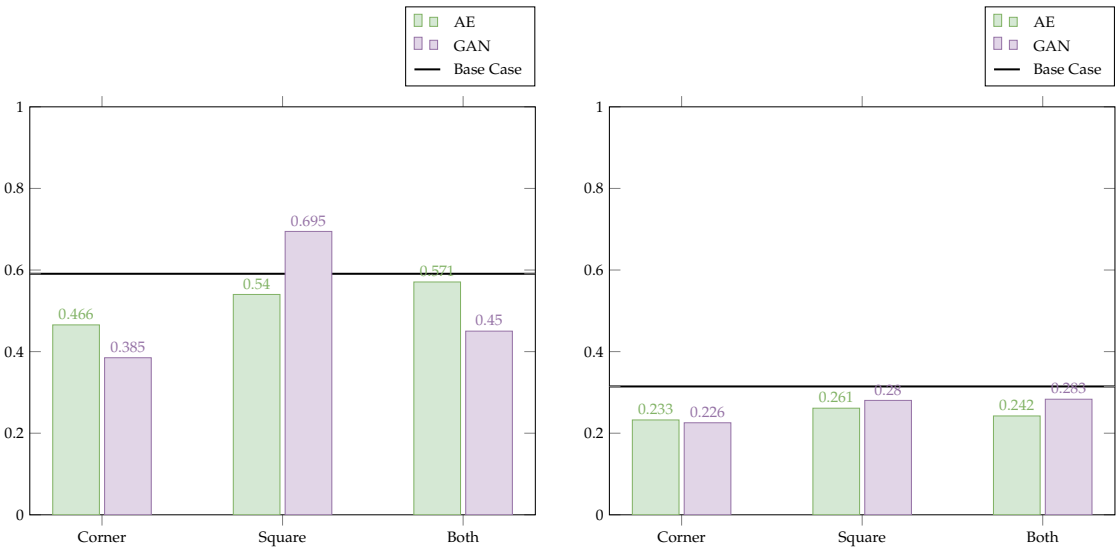
Figure 4.32: Visualisation of the Kvasir dataset MCC values made by both Densenet121 and InceptionresnetV2



(a) CVC 12k with DN121

(b) CVC 12k with IRV2

Figure 4.33: Visualisation of the CVC 12k MCC values made by both Densenet121 and InceptionresnetV2



(a) CVC 356 with DN121 at 512 × 512 px

(b) CVC 12k with DN121 at 512 × 512 px

Figure 4.34: Visualisation of the The three datasets with Densenet121 at 512 × 512 px

Chapter 5

Discussion and future work

The task of making general models to cover a broad aspect of medical images is still a widely researched area today, and most likely, it will continue to be so in the future. There are a plethora of different ways to build models for the medical domain, and in most of them, there is room for improvement.

5.1 Conclusion of the results

We have seen the power of inpainting when it comes to datasets previously unseen during training. Both shown in our publications and the thesis, we have an increase in classification score on multiple types of inpainting. Unfortunately, we often get inconsistent results during testing, which means that we cannot recommend any method over another, as both the testing and training dataset influences the results.

5.2 Future work

The Work done in this thesis show that there might be improvements when classifying images with non-perfect information. There is still much work that can be done, both to improve inpainting and classification, and to better understand the ´´black box" that is machine learning.

Better performance GAN Since the start of this project, there have been published multiple new papers concerning making realistic GANs including [51] [58]. As this is still a relatively new research field, there are still many improvements that could be done to make the models better. The best way to improve the GAN models is just to let them train longer. The latest model used to generate the inpainted dataset were running for approximately 40 hours, which is still away from reaching the best result. By using more time when training the models, we might achieve even better MCC when classifying the medical datasets. Another way that might improve the GAN is better utilisation of the channel-wise fully-connected layer. In this thesis, this layer was

a quintessential part of the result we got, and by tweaking the layers might give even better results.

Looking into using the generated images for classification The images generated by the GAN will most likely have features that are an essential part of the original image. If this is the same underlying features that are used in, for instance, DenseNet or Inceptionresnet we might not need to paste the inpainted area back into the original image. Further research regarding the generator learning features from the different classes could show good results. Another promising aspect of this is to let the discriminator guess the class in addition to real and fake images. If that is the case, the generator needs to learn features that define the different classes. We can see from images like Figure 4.15f that this, to a case, is already happening without making an auxiliary GAN. In the end, the ability to compress the images with a GAN or AE might give us a new way to classify images.

Experiment with self attention *TODO*

Make a generator for new data We have used the GAN and AE to exclusively inpaint images, but both models can, without any extensive modifications, generate data from the same image domain from the original dataset, just like the original DCGAN [59] does. By using the dataset to generate new previously unseen data, we might help classification not to overfit.

Improving the program to work cross domain For future work, we would like to automate the process of inpainting by making the models look better, and give the user the option of choosing their areas to inpaint. The model presented can be used at any dataset, but the user has to edit the masks manually.

Using OCR to remove text We tested the option to remove text using an Optical Character Recognition (OCR) during this thesis, but the time used by OCR algorithms are too slow to work in real-time. Combining the system presented in this thesis with a system like Rosetta [60] or EAST [61] might give a speedup, but at the conclusion of this thesis, we are not able to run OCR and classification without a multi-GPU setup.

NASNetLarge When we chose our general model for classification back in August 2018, we used InceptionResNetV2 as our model. Since then F. Chollet has added NASNetLarge [62] to the list of transfer learning models. NASNetLarge has a higher accuracy on the imagenet model and should possibly be the standard model for the classification.
Do i here talk about everything i didnt do?

- ***removing text, and the permutations regarding this***

- ***using a pixel CNN to inpaint***
- ***using the contextencoder as shown in first paper.***
- ***OCR***

Bibliography

[1] World Health Organization. (February 2018.). Who cancer facts, [Online]. Available: http://www.who.int/mediacentre/factsheets/fs297/en/ (visited on 04/19/2018).

[2] C. R. UK. (). Breast cancer statistics, [Online]. Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/breast-cancer (visited on 03/29/2019).

[3] ——, (). Bowel cancer statistics, [Online]. Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/bowel-cancer#heading-One (visited on 03/29/2019).

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[7] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016. arXiv: 1602.07261. [Online]. Available: http://arxiv.org/abs/1602.07261.

[8] S. Hicks, M. Lux, T. de Lange, K. R. Randel, M. Jeppsson, K. Pogorelov, P. Halvorsen, and M. Riegler, "Mimir: An automatic reporting and reasoning system for deep learning based analysis in the medical domain," Amsterdam, Netherlands: ACM, 2018, pp. 369–374, ISBN: 978-1-4503-5192-8. DOI: 10.1145/3204949.3208129.

[9] S. Hicks, M. Riegler, K. Pogorelov, T. de Lange, K. R. Randel, K. V. Ånonsen, M. Jeppsson, P. Halvorsen, and S. L. Eskeland, "Dissecting deep neural networks for better medical image classification and classification understanding," Karlstad, Sweden: IEEE, 2018. DOI: 10.1109/CBMS.2018.00070.

[10] D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young, "Computing as a discipline," *Commun. ACM*, vol. 32, no. 1, P. J. Denning, Ed., pp. 9–23, Jan. 1989, ISSN: 0001-0782. DOI: 10.1145/63238.63239. [Online]. Available: http://doi.acm.org/10.1145/63238.63239.

[11] Holme, Bretthauer, Fretheim, Odgaard-Jensen, and Hoff, "Flexible sigmoidoscopy versus faecal occult blood testing for colorectal cancer screening in asymptomatic individuals," *Cochrane Database of Systematic Reviews*, no. 9, 2013, ISSN: 1465-1858. DOI: 10.1002/14651858.CD009259.pub2. [Online]. Available: https://doi.org//10.1002/14651858.CD009259.pub2.

[12] n.d. (2011). Colonoscope, WHO, [Online]. Available: https://www.who.int/medical_devices/innovation/colonoscope.pdf (visited on 03/01/2019).

[13] S. Hicks, M. Riegler, P. Konstantin, T. de Lange, D. Johansen, M. Jeppsson, K. R. Randel, S. Eskeland, and P. Halvorsen, "Mimir: An automatic reporting and reasoning system for deeplearning based analysis in the medical domain," in *InProceedings of 9th ACM Multimedia Systems Conference, Amsterdam, Netherlands, June 12–15, 2018 (MMSys'18)*, ACM, 2018. DOI: 10.1145/3204949.3208129. [Online]. Available: https://doi.org/10.1145/3204949.3208129.

[14] T. M. Mitchell, *Machine learning*, eng, New York, 1997.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[16] OpenAI, *Openai five*, https://blog.openai.com/openai-five/, 2018.

[17] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, Third Edition*. 2010, ISBN 9780136042594.

[18] K. P. F.R.S., "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. [Online]. Available: https://doi.org/10.1080/14786440109462720.

[19] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR*, vol. abs/1507.06527, 2015. arXiv: 1507.06527. [Online]. Available: http://arxiv.org/abs/1507.06527.

[20] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," Tech. Rep., 1994.

[21] A. M. TURING, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950, ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: http://oup.prod.sis.lan/mind/article-pdf/LIX/236/433/9866119/433.pdf. [Online]. Available: https://dx.doi.org/10.1093/mind/LIX.236.433.

[22] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theor.*, vol. 13, no. 1, pp. 21–27, Sep. 2006, ISSN: 0018-9448. DOI: 10.1109/TIT.1967.1053964. [Online]. Available: https://doi.org/10.1109/TIT.1967.1053964.

[23] R. Herbert and M. Sutton", ""a stochastic approximation method"," *"Ann. Math. Statist."*, vol. "22", no. "3", "400–407", "1951". DOI: "10.1214/aoms/1177729586". [Online]. Available: "https://doi.org/10.1214/aoms/1177729586".

[24] S. Haykin, *Neural networks*. Prentice hall New York, 1994, vol. 2.

[25] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, 1958. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf.

[26] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Netw.*, vol. 2, no. 3, pp. 183–192, May 1989, ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90003-8. [Online]. Available: http://dx.doi.org/10.1016/0893-6080(89)90003-8.

[27] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[28] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980, ISSN: 1432-0770. DOI: 10.1007/BF00344251. [Online]. Available: https://doi.org/10.1007/BF00344251.

[29] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. arXiv: 1502.03167. [Online]. Available: http://arxiv.org/abs/1502.03167.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1," in, D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds., Cambridge, MA, USA: MIT Press, 1986, ch. Learning Internal Representations by Error Propagation, pp. 318–362, ISBN: 0-262-68053-X. [Online]. Available: http://dl.acm.org/citation.cfm?id=104279.104293.

[33] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14, Montreal, Canada: MIT Press, 2014, pp. 2672–2680. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969033.2969125.

[34] K. Pogorelov, K. R. Randel, C. Griwodz, S. L. Eskeland, T. de Lange, D. Johansen, C. Spampinato, D.-T. Dang-Nguyen, M. Lux, P. T. Schmidt, M. Riegler, and P. Halvorsen, "Kvasir: A multi-class image dataset for computer aided gastrointestinal disease detection," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, ser. MMSys'17, Taipei, Taiwan: ACM, 2017, pp. 164–169, ISBN: 978-1-4503-5002-0. DOI: 10.1145/3083187.3083212. [Online]. Available: http://doi.acm.org/10.1145/3083187.3083212.

[35] M. Kirkerød, V. Thambawita, M. Riegler, and P. Halvorsen, "Using preprocessing as a tool in medical image detection," Nice, France: MediaEval, 2018.

[36] V. Thambawita, D. Jha, M. Riegler, P. Halvorsen, H. L. Hammer, H. D. Johansen, and D. Johansen, "The medico-task 2018: Disease detection in the gastrointestinal tract using global features and deep learning," Nice, France: MediaEval, 2018.

[37] R. J. Borgli, P. Halvorsen, M. Riegler, and H. K. Stensland, "Automatic hyperparameter optimization in keras for the mediaeval 2018 medico multimedia task," CEUR Workshop Proceedings (CEUR-WS.org), 2018.

[38] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. arXiv: 1608.06993. [Online]. Available: http://arxiv.org/abs/1608.06993.

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. arXiv: 1409.4842. [Online]. Available: http://arxiv.org/abs/1409.4842.

[40] M. Mahdianpari, B. Salehi, M. Rezaee, F. Mohammadimanesh, and Y. Zhang, "Very deep convolutional neural networks for complex land cover mapping using multispectral remote sensing imagery," *Remote Sensing*, vol. 10, no. 7, 2018, ISSN: 2072-4292. DOI: 10.3390/rs10071119. [Online]. Available: http://www.mdpi.com/2072-4292/10/7/1119.

[41] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. DOI: 10.1109/cvpr.2017.243. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2017.243.

[42] S. Hicks, P. Smedsrud, P. Halvorsen, and M. Riegler, "Deep learning based disease detection using domain specific transfer learning," MediaEval, 2018.

[43] R. Vikraman, *Global report on state of data science and machine learning 2018 based on kaggle survey*, https://rpubs.com/cvrajesh/kagglesurvey2018, Accessed: 2019-04-09, 2018.

[44] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[45] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[46] F. Seide and A. Agarwal, "Cntk: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: ACM, 2016, pp. 2135–2135, ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2945397. [Online]. Available: http://doi.acm.org/10.1145/2939672.2945397.

[47] A. Damien *et al.*, *Tflearn*, https://github.com/tflearn/tflearn, 2016.

[48] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[49] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. DOI: 10.1109/cvpr.2016.278. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2016.278.

[50] E. L. Denton, S. Chintala, A. Szlam, and R. Fergus, "Deep generative image models using a laplacian pyramid of adversarial networks," *CoRR*, vol. abs/1506.05751, 2015. arXiv: 1506.05751. [Online]. Available: http://arxiv.org/abs/1506.05751.

[51] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," *CoRR*, vol. abs/1809.11096, 2018. arXiv: 1809.11096. [Online]. Available: http://arxiv.org/abs/1809.11096.

[52] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," *CoRR*, vol. abs/1609.05158, 2016. arXiv: 1609.05158. [Online]. Available: http://arxiv.org/abs/1609.05158.

[53] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," *CoRR*, vol. abs/11805.08318, 2018. arXiv: 11805.08318. [Online]. Available: https://arxiv.org/abs/1805.08318.

[54] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10, Haifa, Israel: Omnipress, 2010, pp. 807–814, ISBN: 978-1-60558-907-7. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104425.

[55] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[56] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014.

[57] M. Kirkerød, R. J. B. V. Thambawita, S. Hicks, M. A. Riegler, and P. Halvorsen, "Unsupervised preprocessing to improvegeneralisation for medical image classification," ACM, 2019.

[58] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *CoRR*, vol. abs/1812.04948, 2018. arXiv: 1812.04948. [Online]. Available: http://arxiv.org/abs/1812.04948.

[59] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. [Online]. Available: http://arxiv.org/abs/1511.06434.

[60] F. Borisyuk, A. Gordo, and V. Sivakumar, "Rosetta: Large scale system for text detection and recognition in images," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 2018, pp. 71–79.

[61] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "EAST: an efficient and accurate scene text detector," *CoRR*, vol. abs/1704.03155, 2017. arXiv: 1704.03155. [Online]. Available: http://arxiv.org/abs/1704.03155.

[62] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *CoRR*, vol. abs/1707.07012, 2017. arXiv: 1707.07012. [Online]. Available: http://arxiv.org/abs/1707.07012.

Chapter 6

Appendix

Using preprocessing as a tool in medical image detection

Mathias Kirkerød [1,3], Vajira Thambawita[1,2], Michael Riegler[1,2,3], Pål Halvorsen[1,3]
[1]Simula Research Laboratory, Norway
[2]Oslo Metropolitan University
[3]University of Oslo
mathias.kirkerod@gmail.com,vajira@simula.no,michael@simula.no,paalh@simula.no

ABSTRACT

In this paper we describe our approach to gastrointestinal disease classification for the medico task at MediaEval 2018. We propose multiple ways to inpaint problematic areas in the test and training set to help with classification. We discuss the effect that preprocessing does to the input data with respect to removing regions with sparse information. We also discuss how preprocessing affects the training and evaluation of a dataset that is limited in size. We will also compare the different inpainting methods with transfer learning using a convolutional neural network.

1 INTRODUCTION

Medical image diagnosis is a challenging task in the industry of computer vision. In the last couple of years, as computing power has increased, machine learning has become a tool in the task of image detection, segmentation and classification. In this paper we are looking in depth how to use machine learning to help solve classification tasks on the data-set from the Medico task [8]. The Medico task focuses on image classification in the gastrointestinal (GI) tract. The data is divided in to 16 different classes.

Similar to other parts of image detection, the Medico dataset encounter the challenges that the amount of data is too small, or that the training data does not cover the full distribution of the data in the test case. The main goal of this task is to classify medical images. Our proposal is to use unsupervised machine learning for removal of the green corners that are in the Medico dataset. The details of the task are described in [5, 7].

2 APPROACH

Our approach is divided in to two steps: first preprocessing, then classifying. Our focus is mainly on the preprocessing of the data to remove the green corners in the medical images.

After the preprocessing the dataset we run it through a Convolutional Neural Network (CNN) based on transfer learning. We chose the CNN model based on the top 5 and top 1 accuracy of the pre-trained networks on the Keras documentation pages.

In our approach we use the InceptionResNetV2 [9] network. We also remove the top layer and replace it with a global average pooling layer and a dense 16 layer output, to match the number of classes wanted. In addition, we do not freeze any layers of the model. The five submissions that we run is with the same hyperparameters in the transferlearning model. This means that the difference in results should only come from the different training datasets we use.

(a) Image before inpainting **(b) Image after inpainting**

Figure 1: Differences of images after inpainting

The medical data has 1 main feature that we focus on during the preprocessing, namely the green square in the bottom left corner. A neural network often struggle with areas with really sparse information. Our hypothesis is that just replacing the green area with a similar black area will not yield a better result.

We have a dataset that we use as a base case. This dataset was not augmented, other than shrinking the size of every image to a fixed resolution. The other datasets were augmented in a way that would cover up the green square in one way or another.

Our hypothesis it that if we recreate the areas as they would look like without any sparse areas, the classifier can focus on the right features for classifications. We propose 4 different methods on how to inpaint the corner area of the medical images.
An autoencoder [4], a context conditional generative adversarial network[2, 3], a context encoder [6], and a simple crop of the image.

2.1 Autoencoder

For the autoencoder approach, we created and trained a custom autoencoder from scratch. Our autoencoder consist of a encoder-decoder network, with 2D convolutions as well as rectified linear units as activation functions. In the layer between the encoder and the decoder we included a 25% dropout. [1]

To preprocess the medical data we feed the whole image through the encoder-decoder network. We take the loss of the whole reconstructed image, but only keep the inpainted part. Under training, the goal is to minimize the loss: $L(x, g(f(\tilde{x})))$ Where x is an image without a green corner, and $\tilde{x}$ is the same image with an artificial green corner. In theory we can replace any part of the image with this method.

MediaEval'18, 29-31 October 2018, Sophia Antipolis, France

Table 1: Validation set' results

Method	REC	PREC	SPEC	ACC	MCC	F1
Autoencoder	0.929	0.929	0.981	0.929	0.923	0.928
CC-GAN	0.931	0.932	1.000	**0.931**	**0.926**	0.931
Contextencoder	0.926	0.928	0.945	0.926	0.920	0.926
Clipping	0.903	0.904	0.980	0.903	0.895	0.903
Non-augmenteted	0.925	0.927	0.981	0.925	0.919	0.924

Table 2: Official Results

Method	REC	PREC	SPEC	ACC	MCC	F1
Autoencoder	0.915	0.915	0.994	**0.989**	0.910	0.915
CC-GAN	0.915	0.915	0.994	**0.989**	0.910	0.915
Contextencoder	0.910	0.910	0.994	0.988	0.905	0.910
Clipping	0.904	0.904	0.993	0.988	0.898	0.904
Non-augmenteted	0.917	0.917	0.994	**0.989**	**0.911**	0.917

2.2 Context encoder

For the context encoder approach, we created a new encoder-decoder network. Here the encoder has a similar structure to the autoencoder, but our decoder is only making outputs at the size of the desired area to inpaint. In addition to the loss generated from taking a MSE loss[6]:
$L(\hat{x}, g(f(x)))$ *Where* $\hat{x}$ *is an image with an artificial green corner, and* x *is the part that was replaced by the corner,* we include an adversarial loss, as described in [6].

With the context encoder we feed images without a green corner in to the encoder-decoder network. The output of the network is the same size as the area we want to fill.

2.3 Context conditional generative adversarial network

For the generative adversarial approach, we create a similar structure as the autoencoder. We have a constant 10% dropout at each layer in the discriminator. As with the autoencoder we have the same size input as output, but we only decide to keep the parts we want to inpaint.

We use the same type of loss as the context encoder, with 15% of the loss coming from a MSE loss, and the remaining 85% coming from the adversarial loss.

2.4 Clipping instead of inpainting

The last method was just to crop the images in a way that excluded the green corner. Since every image is scaled down to 256x256 px during preprocessing, the same is done with the clipped version (after the clip the size was reduced to 256x256).

The clipping was done in a way so that we had the most amount of center frame, and minimal amount of the bottom left corner, without sacrificing to much of the image.

3 RESULTS AND ANALYSIS

We made the augmented datasets before we trained the preprocessing model. This means that the transferlearning model did not augment the images at runtime. We split the data into a 70% train set, and a 30% validation set.

Our results on the test set are tabulated in Table 1. The official Results on the test set are tabulated in Table 2. Table 3 shows the confusion matrix from the CC-GAN from the official test set.

The results show that the CC-GAN got the highest MCC score with 0.926, and also the most realistic inpaintings. The context encoder had the lowest MCC score with 0.920, and also the worst inpainted areas. The official result did have the same pattern in MCC score, though the base case got the best result. In both cases the clipping gave significantly worse result.

As expected, most of the images was classified correctly, but we had some problems distinguishing between esophagitis and normal-z-line. We also had a few cases of instruments where there were none.

Table 3: Confusion Matrix

A:ulcerative-colitis , **B**:esophagitis , **C**:normal-z-line , **D**:dyed-lifted-polyps , **E**:dyed-resection-margins , **F**:out-of-patient , **G**:normal-pylorus , **H**:stool-inclusions , **I**:stool-plenty , **J**:blurry-nothing , **K**:polyps , **L**:normal-cecum , **M**:colon-clear , **N**:retroflex-rectum , **O**:retroflex-stomach , **P**:instruments

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
A	510	0	1	0	1	0	1	0	69	0	5	24	0	3	0	13
B	3	401	68	0	1	0	5	0	0	0	0	0	0	0	1	0
C	0	153	489	0	0	0	3	0	0	0	0	0	0	0	0	0
D	0	0	0	502	39	0	0	0	0	0	3	0	0	1	0	45
E	0	0	0	46	517	1	0	0	0	0	1	0	0	0	0	15
F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	2	2	3	0	0	0	547	0	0	0	0	0	0	0	1	0
H	0	0	0	0	0	0	0	486	35	0	0	0	0	0	0	0
I	3	0	0	0	2	0	0	1	1857	0	3	1	0	0	0	3
J	1	0	0	0	0	1	0	1	0	36	0	0	1	0	0	0
K	8	0	1	5	2	3	4	0	0	0	349	17	0	2	1	55
L	11	0	1	2	1	0	1	0	1	1	11	542	0	0	0	3
M	2	0	0	0	0	0	0	18	2	0	1	0	1064	0	1	3
N	2	0	0	1	1	0	0	0	0	0	1	0	0	183	4	5
O	0	0	0	0	0	0	0	0	1	0	0	0	0	2	389	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	131

4 CONCLUSION

In general, when training on a dataset that is homogeneous, the preprocessing is less valuable. We want to remove areas with sparseness, and areas that has nothing to do with the classification.
In our example we used 3 different methods to do this, and we had no improvements in the results. As we can see from the validation set, we saved under a percent on the best method, and we got a worse score on the official results.
We conclude that preprocessing the Medico dataset is not worth the hassle. The effort put in to preprocess the images yields little to no improvement to the result. We recommend that the time is used to find the right network, with the right hyper-parameters instead. A reason to lackluster results might be caused that the training and the test set have the same green squares in the same classes. We suspect that the similarity in the test and train set makes the squares an essential part of the image. We believe that the result would be much better if the test set would be completely without the squares, as they would if they were "real time" images.

In a future test we would also recommend removing the four black edges too. With the images being round, this might be a challenge, since there are no full-resolution images (without zoom) that captures the edges. With the medico dataset, this method will probably not give a better score, on the basis that every image in the dataset has the same four black corners.

REFERENCES

[1] Aaron Courville Yoshua Bengio David Warde-Farley, Ian J. Goodfellow. 2013. An empirical analysis of dropout in piecewise linear networks. abs/1609.05158 (2013). arXiv:1312.6197v2 https://arxiv.org/pdf/1312.6197v2

[2] Emily L. Denton, Sam Gross, and Rob Fergus. 2016. Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks. *CoRR* abs/1611.06430 (2016). arXiv:1611.06430 http://arxiv.org/abs/1611.06430

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[4] Y. Kamp H. Bourlard. 1988. Auto-Association by Multilayer Perceptrons and Singular Value Decomposition. (1988). http://ace.cs.ohio.edu/~razvan/courses/dl6890/papers/bourlard-kamp88.pdf

[5] Pål Halvorsen Thomas de Lange Kristin Ranheim Randel Duc-Tien Dang-Nguyen Mathias Lux Konstantin Pogorelov, Michael Riegler. 2018. Mediaeval information. http://multimediaeval.org/mediaeval2018/medico/. (2018). Accessed: 2018-10-16.

[6] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. 2016. Context Encoders: Feature Learning by Inpainting. *CoRR* abs/1604.07379 (2016). arXiv:1604.07379 http://arxiv.org/abs/1604.07379

[7] Konstantin Pogorelov, Kristin Ranheim Randel, Carsten Griwodz, Sigrun Losada Eskeland, Thomas de Lange, Dag Johansen, Concetto Spampinato, Duc-Tien Dang-Nguyen, Mathias Lux, Peter Thelin Schmidt, Michael Riegler, and Pål Halvorsen. 2017. KVASIR: A Multi-Class Image Dataset for Computer Aided Gastrointestinal Disease Detection. In *Proceedings of the 8th ACM on Multimedia Systems Conference (MMSys'17)*. ACM, New York, NY, USA, 164–169. https://doi.org/10.1145/3083187.3083212

[8] Konstantin Pogorelov, Michael Riegler, Pål Halvorsen, Thomas De Lange, Kristin Ranheim Randel, Duc-Tien Dang-Nguyen, Mathias Lux, and Olga Ostroukhova. 2018. Medico Multimedia Task at MediaEval 2018. (2018).

[9] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. 2016. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *CoRR* abs/1602.07261 (2016). arXiv:1602.07261 http://arxiv.org/abs/1602.07261

Unsupervised preprocessing to improve generalisation for medical image classification

Mathias Kirkerød, Rune Johan Borgli
Simula Research Laboratory, Norway
University of Oslo, Norway
mathiaki@ifi.uio.no, rune@simula.no

Vajira Thambawita, Steven Hicks, Michael Alexander Riegler, Pål Halvorsen
SimulaMet - Simula Metropolitan Center for Digital Engineering, Norway
{vajira, steven, michael, paalh}@simula.no

***Abstract*—Automated disease detection in videos and images from the gastrointestinal (GI) tract has received much attention in the last years. However, the quality of image data is often reduced due to overlays of text and positional data. In this paper, we present different methods of preprocessing such images and we describe our approach to GI disease classification for the Kvasir v2 dataset. We propose multiple approaches to inpaint problematic areas in the images to improve the anomaly classification, and we discuss the effect that such preprocessing does to the input data. In short, our experiments show that the proposed methods improve the Matthews correlation coefficient by approximately 7% in terms of better classification of GI anomalies.**

***Index Terms*—Machine learning, GAN, Autoencoder, Inpainting**

I. Introduction

In the field of computer vision, image-based disease detection has become a popular area of research. For example, algorithms based on deep neural networks have been used to automatically analyse the human digestive system for anomalies such as polyps, lesions and other common illnesses. This is important as the detection and removal of colon polyps is the main prevention method of colorectal cancer, which ranks within the top-three terminal cancer types for both men and woman [1]. Automatically detecting this disease goes a long way of aiding doctors to perform a more thorough analysis of their patients, and has the potential of saving lives. In addition to gastroenterology, we continue to see machine learning based classification systems appear in nearly every branch of medicine.

In recent years, deep learning based algorithms have become a popular method for solving these problems. Aided by the rapid advancement of computational power due to the efficiency of GPUs, deep learning has shown state-of-the-art performance across numerous fields, including medicine. However, deep neural networks are only as good as the data used to train them. Thus, data which contains artefacts such as text and overlays may negatively impact the performance of models trained on this data. This is particularly problematic in medicine, as the selection of datasets is often limited, and the datasets available may include artefacts from the software the doctors use to analyse the images/videos (e.g., overlays, text, and other information).

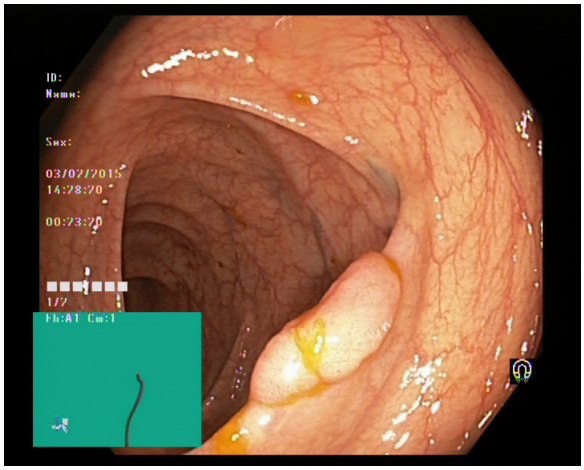


Fig. 1: Example image from the Kvasir dataset with included overlays and black borders.

In this work, we look at improving the quality of a publicly available endoscopy dataset called Kvasir [2], which contains several of the artefacts previously mentioned (example shown in Figure 1). We hope that this shows that there are more ways of improving the performance of a deep neural network than increasing its number of training samples. This work can be seen as an extension of our approach to this years MediaEval Medico task [3], where we presented a similar technique, albeit to a much lesser extent [4]. Additionally, a recent study using Kvasir for training deep learning based models showed that these artefacts directly impacted the classification performance of said models, showing that there is potential room for improvement [5].

The main contributions of this paper are (i) we present different methods for preprocessing data to be able to create better generalisable models, (ii) a detailed cross-dataset evaluation of the methods used and (iii) we report classification performance across different datasets.

II. Related Work

As mentioned in the introduction, medical image classification has been a heavily researched area. Research gathered by Lu and Weng [6] give current practices, problems, and prospects of image classification.

Our methods for inpainting bears a resemblance to context-encoder made by Pathak et al. [7] who introduce an encoder-decoder network in style close to our proposed generative adversarial network. However, a big difference is the use of a channel-wise fully connected layer in their model to share information around in the image space. This part was not necessary for us, given the homogeneity of the medical images coupled with the use of a non-random filter for inpainting.

Denton et al. [8] presented a model for inpainting close to the context-conditional adversarial network presented by Pathak et al. that is also trained on non-medical images, with random filter placement during training and evaluation. Their results showed that their generative adversarial network (GAN) model was capable of producing semantically meaningful inpaintings in a diverse set of images.

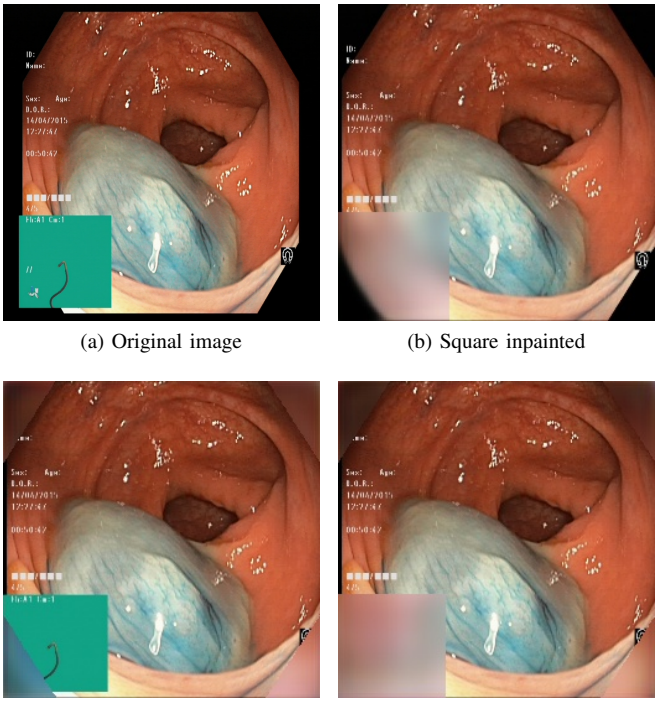
Previously, Hicks et al. [5] applied various preprocessing steps to Kvasir based on analysis conducted on common CNN architectures. Using heat maps and saliency maps, they discovered a common issue where artefacts such as text, black borders, and green navigation boxes were directly correlated to the misclassification of some images. In an attempt to correct this issue, they applied various preprocessing steps to the training data, namely cropping black borders and blacking out the green navigation box. Their results revealed improvement in all cases of data preprocessing, and in the best case, they achieved an increase of Matthews correlation coefficient (MCC) by approximately 3%.

In this paper, we aim to improve on this work by not simply removing borders and green navigation boxes. We also try to replace the artefacts using ideas from GAN inpainting to generate an automatically generated mask which attempts to replicate what would have been there if not for said overlay artefacts.

III. Approach

By using machine learning, we aim to classify medical images from the gastrointestinal (GI) tract correctly. With this approach, it is common to use a dataset for training and validation, with a separate set for testing. In practice the dataset we test on is never seen by the model before its evaluation. This is the main reason why we often struggle to get the same level of accuracy when evaluating our model if the data originates from different sources. In our case, the test data from the CVC dataset differs from the training data in both the image content and size. When this problem arises, it is practice to use domain-specific knowledge to help training, and if the amount of training data is small, methods like K-fold cross-validation [9] can also be used to improve the results.

For this paper, we focus on inpainting as a form of generalised preprocessing. We do this to remove dataset specific overlays for better classification on new datasets no matter the source of the dataset. Furthermore, we have also chosen to use the same Bayesian optimisation techniques as in the Borgli et al. paper [10] to find the optimal network for classification. With both hyperparameter optimisation and inpainting, our goal is to get the highest classification score on the CVC datasets.



(a) Original image (b) Square inpainted

(c) Edges inpainted (d) Corner and edges inpainted

Fig. 2: Here we have a sample of what we want to achieve. (a) Original from the Kvasir dataset. Here we also see extended edges that we can cut away without any machine learning. (b) Same image without edges and the green square. (c) Same image with new corners, (d) Same image with both new corners and new area for green square

A. Preprocessing

As discussed, the Kvasir dataset has some unwanted artefacts that are present in a good portion of the data. Some of the unwanted artefacts are Kvasir specific, and some are general artefacts when capturing images from the colon. First, the camera used in colonoscopies has an exceptionally wide lens. This setup takes good medical images but comes with the drawback that the images are not rectangular. Because of this, the camera needs to add black corners and borders to save the images. Another unwanted artefact that is Kvasir specific is an unwanted additional overlay added to the images. They are added post-image-capture by the medical staff, and they show essential information about the patient. As we can see from this, we have multiple areas in the images with pixels not originating from the patient, and subsequently contains no information relevant for classification.

A neural network will also often struggle with areas with really sparse information. Because of this, we believe that just replacing the green area with a similar black area will not yield

TABLE I: Details of all datasets used in the experiments.

BC: Black corner. **GS**: Green square. **BC+GS**: Black corner and Green square

Dataset labels	Size	Inpainted area	Generator network used
D-I	256x256 px	-	-
D-II	256x256 px	BC	Autoencoder
D-III	256x256 px	GS	Autoencoder
D-IV	256x256 px	BC+GS	Autoencoder
D-V	256x256 px	BC	GAN
D-VI	256x256 px	GS	GAN
D-VII	256x256 px	BC+GS	GAN
D-VIII	512x512 px	-	-
D-IX	512x512 px	BC	Autoencoder
D-X	512x512 px	GS	Autoencoder
D-XI	512x512 px	BC+GS	Autoencoder
D-XII	512x512 px	BC	GAN
D-XIII	512x512 px	GS	GAN
D-XIV	512x512 px	BC+GS	GAN

the best result. However, we expect improvement if we instead try to inpaint both the green corner and the black edges with data gathered from similar images. Furthermore, by removing areas that are specific for that dataset, we believe the model will be far better at generalising to other datasets within the same domain. In our case, the area we will be inpainting is the green area, since it is not present in the CVC datasets, and most other medical datasets are also without it.

With our two hypotheses, we have two different features that we believe will make the classification harder. We first aim to inpaint both areas separately to see how each of them affects classification. We also want to try to collectively remove both areas to see if a combined mask will yield a better or worse result.

With this in mind, we use two different methods for inpainting the desired areas. First, an autoencoder (AE) [11] as a lightweight way to generate new data, and second we use a GAN [12] as a more sophisticated generator. Both methods are unsupervised learning methods to generate new data within the distribution of the original dataset.

For our experiments, we scale our data to a constant resolution. We run four experiments with 256x256 pixels (px) resolution, and four experiments at 512x512 px. Our change in resolution is to compare the effect it has compared to our standard 256x256 px. With this configuration, we end up with 14 augmented datasets shown in table I.

B. Classification

Our research from the 2018 MediaEval workshop showed less desirable result compared to other projects that researched on the same dataset [13] [10]. Therefore one of our goals is to make our model more realistic by using a model that works better on the augmented Kvasir dataset. Using the Bayesian hyperparameter optimiser on our newly created datasets, we choose Densenet121 [14] as our default architecture for training our new datasets. We are also interested in the accuracy compared to a more general classification network. We ran model D-I - D-VII with the pretrained InceptionResNetV2 [15] network. We chose this network because of its high accuracy on the Keras websites [16], and thus we hypothesise that the model will be generally good without hyperparameter optimisation. In both cases, we remove the top layer and replace it with a global average pooling layer and a dense eight layer output to match the number of classes in the training dataset.

TABLE II: Details of experiments.

Test	Training datasets	Testing dataset	Network model
T1	D-I - D-VII	Kvasir V2	DenseNet121
T2	D-I - D-VII	CVC-12k	DenseNet121
T3	D-I - D-VII	CVC-356	DenseNet121
T4	D-I - D-VII	CVC-356	InceptionResnetV2
T5	D-VIII - D-XIV	CVC-356	DenseNet121

Our focus is the comparison between the generated datasets and the baseline; hence we do not change the hyperparameters after they are chosen. We believe this sets up a valid comparison since the only difference in score should come from the differences in the dataset and not the classification model. An overview of our experiments are shown in Table II, where Models T1 - T3 is a direct comparison on how well we have generalised our model, while Models T4 & T5 show how changing models will affect the results. Below, we give brief a description of the three datasets used.

a) The Kvasir V2 dataset [2]: The Kvasir V2 dataset consists of 8,000 images from the GI tract. Several of these images contain artefacts such as navigation boxes (green box as seen Figure 1), overlayed text, black borders, and black edges. With our first hypothesis in mind, we assume that the dataset with the inpainted rounding corners (D-II & D-IV) will do slightly better than the baseline (D-I). This is because the training and test data is from the same set, and subsequently our generalisation will not help. That leaves us with the only way to improve the result is to remove sparseness.

b) The CVC-356 dataset [17]: The CVC-356 dataset consists of 2,285 images from the lower GI tract. CVC-356 does not have images with green boxes. It does have images with black borders, and rounded black edges. As stated in our second hypothesis; the inpainting of the green square will presumably give the best result. This is because, as stated, the CVC-356 images has the same black rounded corners as Kvasir, but lacks the green squares.

c) The CVC 12k dataset [17]: The CVC-12k dataset consists of 11954 images from the lower GI tract, with a resolution of primarily 288x384 px. Given the similarity with the CVC-356 dataset, this will presumably follow our second hypothesis stating that the inpainting of the green square would give the best result. Given that the CVC-12k images has the same black rounded corners as Kvasir, but lacks the green squares.

IV. Preprocessing tools

The networks used for inpainting are based on the network presented in the Mediaeval conference [4]. Both networks are using on masking, where only the parts of the image corresponding to a mask was inpainted.

A. Autoencoder

The first approach we created and trained was a custom autoencoder [11] from scratch. Our autoencoder consists of an encoder-decoder network, with 2D convolutions as well as rectified linear units as activation functions, and a 25% dropout between the encoder and decoder. The network used is a modification of the network presented in [4]. The modifications are a smaller batch size and a more consistent filter size throughout the network. These modifications were made to make more credible results, and to get a lower error during training. The loss function was also modified to solely train on parts of the images that were modified. This lead to a larger and more accurate gradient descent, which also contributed to a better reconstruction.

B. Context conditional generative adversarial network

For the GAN approach, we create a similar structure to the autoencoder. We have a generator-discriminator network that serves much of the same functionality as the encoder-decoder network in the autoencoder. As with the autoencoder, we have the same size input as output, but we only decide to keep the parts we want to inpaint. The model we ended up with is closely inspired to the model made by Denton et al. in [18]. The main differences are the number of layers used, and the lack of a low-resolution image as an extra input.

V. Results

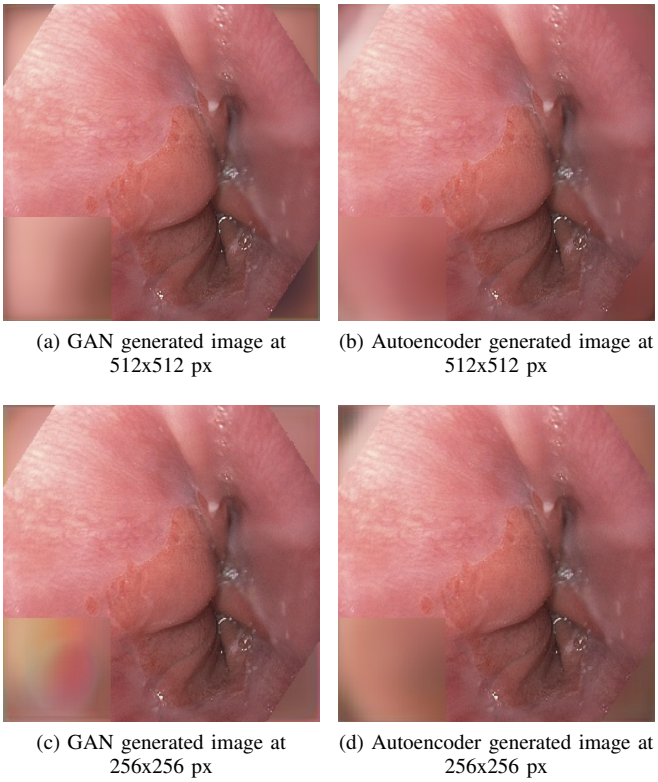
We divide our results into two sections, preprocessing and classification. In our preprocessing section, we discuss the appearance of the dataset, and how close the results are to the ground truth. In our classification section, we discuss the rate of generalisation and rate of success.

A. Preprocessing

Since there are no specific metrics associated with the training of Autoencoders and GANs, we used the mean square error of the ground truth as a metric of our progress. Figure 3 from the z-line shows how the two different models perform on the two different sizes. This is a typical case where both the GAN and the AE are fairly similar, except for more features added by the GAN. The features are most present in the smaller images, as the images are easier to train on, and subsequently easier to add complex local features too.

B. Classification

We evaluated our model on both the Kvasir and the CVC dataset as described in the classification section (III-B). When presenting our results, our main point of comparison is the MCC [19]. In addition to the MCC score, we use F1, precision and recall as metrics when presenting our results. In addition to the best MCC score, we present the average MCC score as an indicator of the general success of the method in question.



(a) GAN generated image at 512x512 px

(b) Autoencoder generated image at 512x512 px

(c) GAN generated image at 256x256 px

(d) Autoencoder generated image at 256x256 px

Fig. 3: Same image from the z-line with four different inpainting attempts. Each image is re-sized to fit in the figure.

Since our task was to improve classification and cross-dataset generalisability through inpainting, each table has its first row as the dataset without any inpainting, followed by the rest of the datasets. The first column is the maximum MCC score of the runs. Then we give the maximum F1 score followed by the maximum precision and recall. The last column gives us the average MCC of all four runs for each model.

First, we evaluated our results on the three datasets: Kvasir, CVC-356 and CVC-12k. Here our goal was to see the general improvement based only on inpainting and dataset. Then we evaluated the InceptionResNetV2 network on the CVC-356 dataset, and lastly, re-evaluated the CVC-356 network, at double image size.

a) Kvasir, Test T1: These are our results from training and evaluating on the Kvasir v2 dataset with the 5,600 image training set, 800 image validation set, and 1,600 image test set split. Table III shows the highest value for each of the six methods compared to the highest baseline.

As we can see in the results shown in Table III, we got the highest MCC score on the baseline dataset. Both the best and average scores were highest for the baseline, but the average was consistently high for all methods. As we recall, we predicted that we expected a higher MCC score for the Autoencoder inpainting the black corner and the GAN

TABLE III: Test T1, Kvasir dataset on DenseNet121

Dataset	MCC	F1	Precision	Recall	MCC (AVG)	MCC (SD)
D-I	0.9307	0.9394	0.9396	0.9394	0.9163	0.0166
D-II	0.9150	0.9254	0.9303	0.9250	0.9053	0.0102
D-III	0.9212	0.9310	0.9347	0.9306	0.9040	0.0167
D-IV	0.9187	0.9287	0.9298	0.9288	0.9105	0.0057
D-V	0.9208	0.9308	0.9316	0.9306	0.9108	0.0067
D-VI	0.9096	0.9204	0.9226	0.9206	0.9055	0.0038
D-VII	0.8960	0.9094	0.9174	0.9081	0.8926	0.0049

TABLE IV: Test T2, CVC-12k dataset on DenseNet121

Dataset	MCC	F1	Precision	Recall	MCC (AVG)	MCC (SD)
D-I	0.2897	0.5558	0.6968	0.6067	0.2723	0.0329
D-II	0.3031	0.5413	0.7148	0.5927	0.2675	0.0250
D-III	0.3197	0.6152	0.7050	0.6600	0.2649	0.0374
D-IV	0.2956	0.4663	0.7632	0.5156	0.2733	0.0225
D-V	0.2967	0.5451	0.7072	0.5965	0.2523	0.0440
D-VI	0.2803	0.4548	0.7571	0.5038	0.2244	0.0410
D-VII	0.2225	0.5740	0.6451	0.6236	0.1984	0.0195

TABLE V: Test T3, CVC-356 dataset on DenseNet121

Dataset	MCC	F1	Precision	Recall	MCC (AVG)	MCC (SD)
D-I	0.7070	0.9137	0.9132	0.9164	0.5904	0.1104
D-II	0.5153	0.7846	0.8153	0.8065	0.4861	0.0307
D-III	0.7325	0.9402	0.9535	0.9348	0.6465	0.0978
D-IV	0.6631	0.9264	0.9410	0.9194	0.5637	0.1011
D-V	0.5714	0.8387	0.8487	0.8516	0.4557	0.1002
D-VI	0.7150	0.9214	0.9206	0.9225	0.6334	0.0819
D-VII	0.7466	0.9370	0.9391	0.9356	0.4576	0.1941

TABLE VI: Test T4, CVC-356 dataset on InceptionResNetV2

Dataset	MCC	F1	Precision	Recall	MCC (AVG)	MCC (SD)
D-I	0.4038	0.8851	0.9130	0.8678	0.2999	0.0841
D-II	0.2221	0.7957	0.7958	0.7955	0.1227	0.0900
D-III	0.0745	0.4489	0.5535	0.5131	0.0299	0.0374
D-IV	0.3147	0.7793	0.7730	0.7916	0.1636	0.1197
D-V	0.1802	0.5434	0.6201	0.5985	0.0446	0.0923
D-VI	0.3276	0.8372	0.8429	0.8323	0.2234	0.0826
D-VII	0.2738	0.6754	0.6938	0.7106	0.1417	0.1230

inpainting the black corner. The results do not show a clear indication that the baseline was the best method, nor that there are any good ways to inpaint this dataset.

b) CVC-12k, Test T2: The T2 test case was trained on the Kvasir v2 dataset with the 5,600 image training set and the 800 image validation set, then evaluating on the CVC-12k dataset. Table IV shows the highest value for the six methods compared to the highest baseline, with four runs each.

As we can see in the results, shown in Table IV, we got the highest MCC score on the dataset with the inpainted green square made by the autoencoder. Also, the average score was consistently higher for the autoencoder datasets compared to the GAN datasets. The results give a small indication that inpainting the green area with an autoencoder might give a better result compared to the baseline.

c) CVC-356, Test T3: The T3 test case was, as test case T2, trained on the Kvasir v2 and evaluated on the CVC-356 dataset. The table V shows the highest value for each of the six methods compared to the highest baseline, with four runs each.

As we can observe in the results shown in Table V, we got the highest MCC score on the dataset with the inpainted green square made by the autoencoder and the GAN. We can also see a constant higher value for both datasets inpainting the green area. The highest value was from the dataset with both corner and square inpainting, but this is most likely just a lucky result, given the low average MCC. The results give a reasonable indication that inpainting the green area will give a better result compared to the baseline.

d) InceptionResNetV2, Test T4: These are our results from training on the Kvasir v2 dataset with the 5,600 image training set and the 800 image validation set, then evaluating on the CVC-365 dataset. The table VI shows the highest value for each of the six methods compared to the highest baseline, with four runs each. In this run we used the InceptionResNetV2 network to train our model.

As we can see from the results shown in Table VI, we got the highest MCC score on the baseline dataset. From our tests, it looked like the overall scores were much lower here compared to our DenseNet121 models, and in general, we got more unpredictable scores.

e) Double image size, Test T5: These are the results from training on the Kvasir v2 dataset with the 5600 image training set and the 800 image validation set, then evaluating on the CVC-365k dataset. The table VII shows the highest value for each of the six methods compared to the highest baseline, with four runs each. Here we have doubled the size of the images for the training and evaluation set to see how size affects the results.

On the CVC-356 dataset at 512x512 px resolution, we see a generally lower MCC score compared to the same dataset at 256x256 px. Our best average results came from the dataset with both inpainted corners and inpainted squares, but it looks like the more inpainting, the better. The results give a small indication that inpainting large areas with sparse information might give a better result compared to the baseline, at least compared to smaller areas.

Overall, we can observe through all experiments that inpainting can both improve and worsen the results. In general, inpainting works best when applied in dataset specific artefacts that are not present in the test set.

VI. Discussion

Our first hypothesis was that removal of the black edges and corners around the images would result in a better classification and better generalisation. Our results also show that training and testing on the same dataset gave approximately

TABLE VII: Test T5, CVC-356 dataset with double resolution

Dataset	MCC	F1	Precision	Recall	MCC (AVG)	MCC (SD)
D-VIII	0.5865	0.8711	0.8702	0.8770	0.4696	0.1560
D-IX	0.6447	0.8992	0.8980	0.9015	0.4775	0.1142
D-X	0.4346	0.8894	0.9157	0.8735	0.3754	0.0709
D-XI	0.6449	0.8998	0.8986	0.9019	0.5935	0.0402
D-XII	0.7189	0.9294	0.9311	0.9282	0.4499	0.2110
D-XIII	0.5956	0.8891	0.8880	0.8905	0.5547	0.0604
D-XIV	0.7234	0.9235	0.9228	0.9247	0.5737	0.1173

the same MCC score, with and without corners. In addition, we observed that the removal of areas within the images with no relevant information did not give any better results, given the same training and test distribution. This was not the case when the images were up-scaled above their original size, as we saw a much better result when the areas were inpainted. We also observed that by removing the corners on the Kvasir set during training, the testing on the CVC-sets we did not get any better results in general. This was as expected since all the images had black edges, and removing them from training would make the datasets less alike. Our second hypothesis was concerning the removal of the green squares in the training set. With this, we wanted to see how the inpainted training sets affected to the test set that did not originate from the original distribution. We observed good results for both the CVC-12k set and the CVC-356 set. For the set, we deemed most realistic, namely the CVC-356 set, we saw that our score consistently was higher both for the average and the max MCC. Lastly, using a non-optimised network gives a lower MCC score when inpainting. In general, we see that inpainting to only remove sparseness will often worsen the results when the test and training set is from different sources. The same goes for excessive inpainting.

VII. Conclusions

Our two main hypotheses regarding types of inpainting for this paper were about how it would affect classification. We tested this on various datasets with different models at different sizes to see how the datasets affected the classification score. From our experiments, we can see that inpainting can help when generalising the training data to other datasets. In our GI anomaly classification experiments, our models show an average increase of at least 7% MCC score when using an optimal network for testing on images that are not from the same domain as the training data, shown in VII. When working with bigger size images, and subsequently larger areas with sparse information, it seems that inpainting does a better job, compared to smaller images. The results coincide with the previous work done [4].

References

[1] B. Stewart and C. Wild, *International Agency for Research on Cancer. World Cancer Report 2014 (International Agency for Research on Cancer)*. World Health Organization, 2014.

[2] K. Pogorelov, K. R. Randel, C. Griwodz, S. L. Eskeland, T. de Lange, D. Johansen, C. Spampinato, D.-T. Dang-Nguyen, M. Lux, P. T. Schmidt, M. Riegler, and P. Halvorsen, "Kvasir: A multi-class image dataset for computer aided gastrointestinal disease detection," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, ser. MMSys'17. New York, NY, USA: ACM, 2017, pp. 164–169. [Online]. Available: http://doi.acm.org/10.1145/3083187.3083212

[3] K. Pogorelov, M. Riegler, P. Halvorsen, S. Hicks, K. Randel, D.-T. Dang-Nguyen, M. Lux, O. Ostroukhova, and T. Lange, "Medico multimedia task at mediaeval 2018." CEUR Workshop Proceedings (CEUR-WS.org), 2018.

[4] M. Kirkerød, V. Thambawita, M. Riegler, and P. Halvorsen, "Using preprocessing as a tool in medical image detection." CEUR Workshop Proceedings (CEUR-WS.org), 2018.

[5] S. Hicks, M. Riegler, P. Konstantin, K. V. nonsen, T. de Lange, D. Johansen, M. Jeppsson, K. R. Randel, S. Eskeland, and P. Halvorsen, "Dissecting deep neural networks for better medical image classification and classification understanding," 2018.

[6] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *International Journal of Remote Sensing*, vol. 28, no. 5, pp. 823–870, 2007. [Online]. Available: https://doi.org/10.1080/01431160600746456

[7] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2016.278

[8] E. Denton, S. Gross, and R. Fergus, "Semi-supervised learning with context-conditional generative adversarial networks," 2016.

[9] S. M., "Cross-validatory choice and assessment of statistical predictions." *Journal of the Royal Statistical Society*, no. 36(2), pp. 111–147, 1974.

[10] R. J. Borgli, P. Halvorsen, M. Riegler, and H. K. Stensland, "Automatic hyperparameter optimization in keras for the mediaeval 2018 medico multimedia task." CEUR Workshop Proceedings (CEUR-WS.org), 2018.

[11] Y. K. H. Bourlard, "Auto-association by multilayer perceptrons and singular value decomposition," 1988. [Online]. Available: http://ace.cs.ohio.edu/ razvan/courses/dl6890/papers/bourlard-kamp88.pdf

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[13] S. Hicks, P. H. Smedsrud, P. Halvorsen, and M. Riegler, "Deep learning based disease detection using domain specific transfer learning." CEUR Workshop Proceedings (CEUR-WS.org), 2018.

[14] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2017.243

[15] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2016.

[16] F. Chollet, "Applications - keras documentation. [online]," https://keras.io/applications/, 2019, accessed: 2019-01-07.

[17] J. Bernal and H. Aymeric, "Miccai endoscopic vision challenge polyp detection and segmentation," 2017, accessed: 2019-01-07. [Online]. Available: https://endovissub2017-giana.grand-challenge.org/home/

[18] E. L. Denton, S. Gross, and R. Fergus, "Semi-supervised learning with context-conditional generative adversarial networks," *CoRR*, vol. abs/1611.06430, 2016. [Online]. Available: http://arxiv.org/abs/1611.06430

[19] B. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA) - Protein Structure*, vol. 405, no. 2, pp. 442 – 451, 1975. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0005279575901099