Using unsupervised machine learning as a tool for polyp detection in the GI tract

Mathias Kirkerod

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

Using unsupervised machine learning as a tool for polyp detection in the GI tract

Mathias Kirkerod

Using unsupervised machine learning as a tool for polyp detection in the GI tract

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

Topic, challenge, solution
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 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 by radiation or other factors 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 born. In most cases, this is not a problem, as most cells self destruct when they cannot operate. cite

Another factor that increases the risk of cancer is 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 cells that are cancerous. In either external or internal cases, cancer is by definition this uncontrollable multiplication.

Statistics on cancer
Since cancer is, in practice, just a cell division error, it has the opportunity to hit anyone, at any age. Because of this, it is a heavily researched area, both

in Norway, and the rest of the world. In 200X we spent 22Y million dollars worldwide on cancer research. Despite being such a researched area, it is still one of the top causes of human death. Some types of cancer, like cancer, is one of the simpler forms of cancer to treat, and at this point, those kind of cancers are non-fatal. *The most common types of cancer in males are lung cancer, prostate cancer, colorectal cancer and stomach cancer.[2]*

colorectal cancer

Humans can get cancer in every major organ, but some types of cancer are more common than others. For instance cancer in the gastrointestinal tract (GI) is one of the more common places to get cancer. Colorectal cancer is just behind x as the most common cancer for men, and it has a mortality rate of x in the first y years. We often call this 5-year survival rate for z. Z is the standard way to measure the life expectancy of a patient diagnosed with cancer.

polyps

Colorectal cancer often starts in polyps. Polyps are, polyps do.

CAD

Generalisable model is needed

Our solution to the problem

1.2 Problem definition

1.3 Research Method

https://dl.acm.org/citation.cfm?id=63239 https://dl.acm.org/citation.cfm?id=63239 https://dl.acm.org/citation.cfm?id=63239 https://dl.acm.org/citation.cfm?id=63239 https://dl.acm.org/citation.cfm?id=63239 https://dl.acm.org/citation.cfm?id=63239

1.4 Main contributors and related work

The work presented in this thesis builds upon the work done by the following publications.

EIR - A Medical Multimedia System for Efficient Computer Aided Diagnosis by Riegler

cite PhD Michael.

The thesis of Hicks and subsequent publications made by Hicks et al. is the foundation for this thesis. In Hicks's thesis, the author gives an introduction into the 'black box' that is machine learning and tries to decode how the medical images affect the neural network's decisions. The thesis shows how the use of saliency maps can help with the understanding of deep

convolutional networks, and it showed how dataset specific artefacts could affect the classification.

The paper, x by Hicks et al. show how removing the dataset-specific artefacts referenced in his thesis gave a better accuracy when it came to classifying images from the GI tract.

The papers submitted by Pathak et al. and Denton et al. and their work on using Generative Adversarial Networks with regards to inpainting are implemented and improved for this dataset-specific task.

Denton et al. in the paper *SEMI-SUPERVISED LEARNING WITH CONTEXT-CONDITIONAL GENERATIVE ADVERSARIAL NETWORKS* shows in detail how to use neural networks with an adversarial approach to inpaint areas in images. In their paper, they propose an Encoder-Decoder approach with realistic looking inpaining results.

Pathak et al. shows, in the publication *Context Encoders: Feature Learning by Inpainting* a network and results close the work done by Denton et al. Using a similar Encoder-Decoder network as Denton et al. they achieved similar results inpainting areas in images from the Imagenet dataset refimagenet

Our work with Generative Adversarial Networks in this thesis has drawn inspiration from both papers and improved the work to better suit the medical image domain.

1.5 Outline

The thesis is organised as follows:

Chapter 2: Background We give more background information about 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), and the potions that are on the market at the present date. 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 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 datasets we use in this thesis, the methods used when inpainting images, and the design of the experiments and programs. We will first look at the Kvasir

dataset and go into detail about the creation and attributes with the dataset. We will look at the CVC 356 and the CVC 12k dataset and compare them to the Kvasir dataset. After discerning the dataset-specific artefacts from the datasets, we look at the options for inpainting the dataset based on the artefacts we want to remove. Lastly, we look into the whole process from preprocessing to classifying.

Chapter 4: Experiments We give a review of the metrics we use to evaluate our results, followed by the results of the inpainting and classification.

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. Then will then look at what the objective is for the medical staff, with different anomalies in the GI tract. Then our focus is moved 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 common form of machine learning, namely neural networks, and show the structure that is most used in this type of machine learning.

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.

need some reworking

2.1 The Medical Background

In the field of medical diagnosis, there are always new and interesting methods being researched to help the medical staff when it comes to patient rate of survival, and quality of life. Everything from x to y is ways the medical industry has done to improve the survival rates of their patients. In the last decade *comuters and cameras came to help us.* Another example is the invention and usage of gastro-stick-with-camera.

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. [3] 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. [4]

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.

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 [5] works, we need to look at how Machine learning helps with the detection of the anomalies the medical staff are after.

Machine learning is a broad term, but can, in short, be summarised by:

> 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. [6]

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 does 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. With this in mind, we can assume that, given the right amount and type of data, our machine learning algorithms can solve any problems humans can solve, given both the right training environment and a way to train, given the environment.

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 stores the experience gathered.

2.3.1 Machine learning types

With a basis in the quote from the machine learning book from Tom Mitchell [6] , 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 humans, we have multiple ways to gather and retain information.

Figure 2.1 shows a chart over the three most common categories within the field of machine learning.

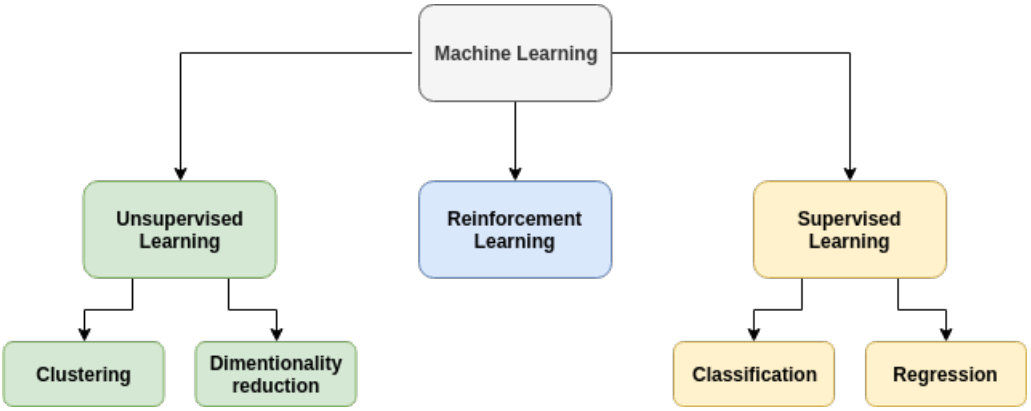


Figure 2.1: The three main types of machine learning and their subtypes

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. [7] 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.

Examples of supervised tasks are to recognise handwritten numbers, or differentiate between different car models. The task is considered 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 supervised learning tasks is regression. Regression is the act of prediction given prior data. Examples of regression are the prediction of stock prices, to house prices in an area, to predicting patterns in the weather.

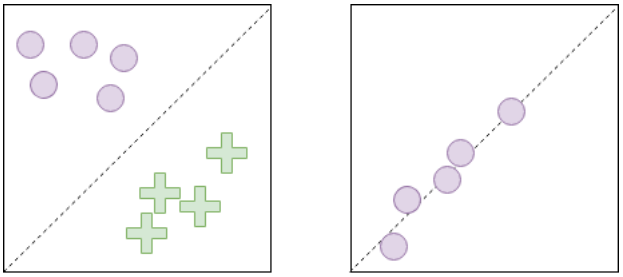


Figure 2.2: Left: Example of binary classification. Right: Example of regression

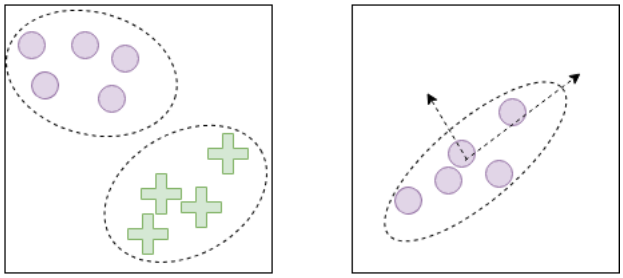


Figure 2.3: Left: Example of binary clustering. Right: Example of principal component analysis

Unsupervised machine learning

Unsupervised learning is the act of training without any supervision, on the sense that we do not give the algorithm the same input-output pair as in supervised 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 dimentionality reduction, the act of simplify or compress the data.

An example of this can be if we want to sort plants based on species, or we are detecting anomalies in a dataset. Unsupervised learning can be used for PCA 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.

In the description of supervised vs unsupervised we looked at a specific branch of machine learning: Classification. Classification is, as the name implies, the task of getting data sorted into groups of similarity.

Reinforcement learning

Is the area in machine learning that is concerned with how a software agent ought to take actions. The actions taken is based on the environment and it is influenced by the objective to get the maximum reward. It is closely influenced by behaviorism, with the fact that the agent want to constantly maximize the reward obtained.

Successful types of reinforcement learning alogrithms are, for instance, Deep Recurrent Q-Learning [8], State–action–reward–state–action (SARSA) [9].

Reinforcement learning algorithms like these can not be categorized under either supervised or unsupervised machine learning.

more?

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 10000 century. The first concept was of the

more

In more recent times, when the first computers came, a common type of machine learning was the K-nearest neighbour algorithm.

fast

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 first 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.

no training

talk about it as a clustering method, classification

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.

proposed by

cite

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.

find out how much I need to write here

cite

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.

perhaps NN

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

might be own chapter

their form of learning, and we have shown how they can be used in the real world. 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 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 a more advanced example utilising a Generative adversarial network as a template.

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 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.

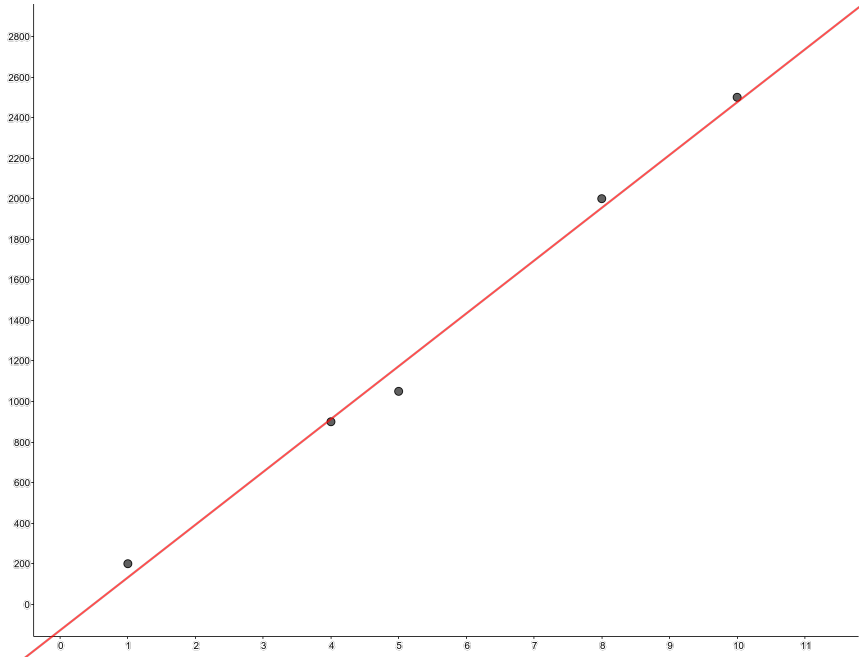


Figure 2.4: Example of linear regression in Geogebra Here the red line is the best approximation of a y value, given an x value.

Figure 2.4 shows an ideal example of linear regression. 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 simple machine learning.

We can recall from the quote 2.3 by [6] 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.

$$y = W^{(1)}x + W^{(2)} \tag{2.2}$$

Here, like before, our output is y, and our input is x. We have replaced a and b with new placeholders $W^{(1)}$ and $W^{(2)}$. Now our goal is to, given a task T, gain experience E and store it in $W^{(1)}$ and $W^{(2)}$. With our values for $W^{(1)}$ and $W^{(2)}$ 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 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 $W^{(1)}$ and $W^{(2)}$), 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, our ideal model would lie at $y = 0.5x + 1$. This means that our ideal weights would be $W^{(1)} = 0.5$ and $W^{(2)} = 1$ In our initial formula we set the the weights $W^{(1)} = 1$ and $W^{(2)} = 0$

Using the formula 2.2 with the input x values we can calculate $\hat{y}$ to be:

$$\hat{y} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \tag{2.5}$$

We can now calculate the performance by applying the mean square error function. Using 2.3 gives us a loss of:

$$\frac{1}{2*3}\left((1.5-1)^2 + (2-1)^2 + (2.5-1)^2\right) = 3.29 \tag{2.6}$$

With our new found error, we need a way to use this to update our weights $W^{(1)}$ and $W^{(2)}$ 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 [10]. In our case, we are looking for the minimum value of the MSE function. Gradient descent is defined as (simplified for our example):

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

Where ∇F is the derivative of function in question, a is the input at step n, and γ is a learning rate set to a small number.

Derivating 2.3 and using a learning rate of 0.5 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 $W^{(1)}$ and $W^{(2)}$

$$\begin{aligned} W^{(0)} &= W^{(0)} - \gamma \frac{1}{m} \sum_i (w^{(0)} \cdot x_i + w^{(1)} - y)_i \cdot x_i \\ &= 1 - 0.5 \cdot \frac{1}{3} \sum (-0.5 + 1 + 1.5) \\ &= 0.583 \end{aligned} \tag{2.10}$$

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

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.

We looked at an example using the formula for a linear approximation. In the real world there are only a handful problems that can be solved by doing a linear approximation.

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 chapter, 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. 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 classification.

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. The fist perceptron was introduced by in , and it was made as an attempt to mimic the human neuron.

who

find when

non-linear problems

Figure 2.5 shows how a human neuron looks like, and in which direction the signal goes. Each neuron is connected to multiple other neurons by connecting

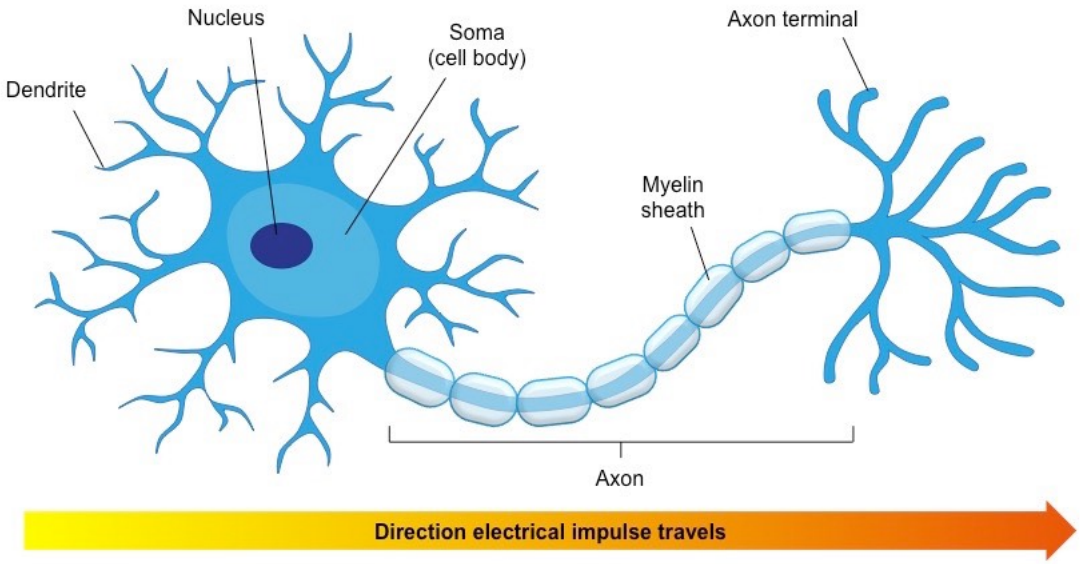


Figure 2.5: https://socratic.org/questions/how-is-a-neuron-adapted-to-perform-its-function

the dendrites to other neighbouring neurons. 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.

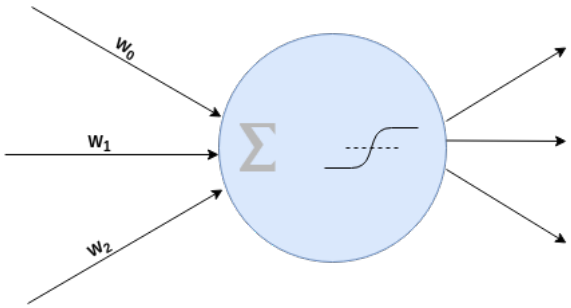


Figure 2.6: Simple perceptron

We can now look at the perceptron in figure 2.6, as we can discern, this mathematical model has the same structure, and we have the same flow as we would in the human brain. First, we get numerical data from, either other perceptrons

or from an input. We add it together and send it out on the other side. We will now look more in depth how this is done, and show how the mathematics behind it works.

Step 1: our perceptron gets signals $a_{(i,0)} - a_{(i,n)}$ where n is the number of inputs to the perceptron.

Step 2: each of our signals $a_{(i,0)} - a_{(i,n)}$ is multiplied by a weight $W^{(0)}$ - $W^{(n)}$.

Step 3: we sum every input to a scalar.

Step 4: We apply an activation function. This can be a simple sigmoid function or tanh function, but the most common is the Rectified linear unit (ReLu)

cite

Step 5: The output of the activation function $a_{(j,0)} - a_{(j,n)}$ is sent to the next perceptron.

talk about why we use an activation function

2.4.2 Multilayer perceptrons

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

cite

The first multilayer network at the time used backpropagation in the same way described in .

talk about the single neuron made by that dude

Figure 2.7 shows the basic structure of a multilayer network. In this figure, we have four things we need to look into.

find formula

2.4.3 Feed forward and backpropagation through a neural network

looked at how MLP works, let us look at back-and-forward prop

2.4.4 Convolutional neural networks

The multilayer perceptron we have discussed is a strong 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 chapter that the number of weights between neurons is . As the number of perceptrons per layer in our neural networks increases, the total amount of storage space increases by a factor of 4

ref chapter on MLP

insert math

fix this

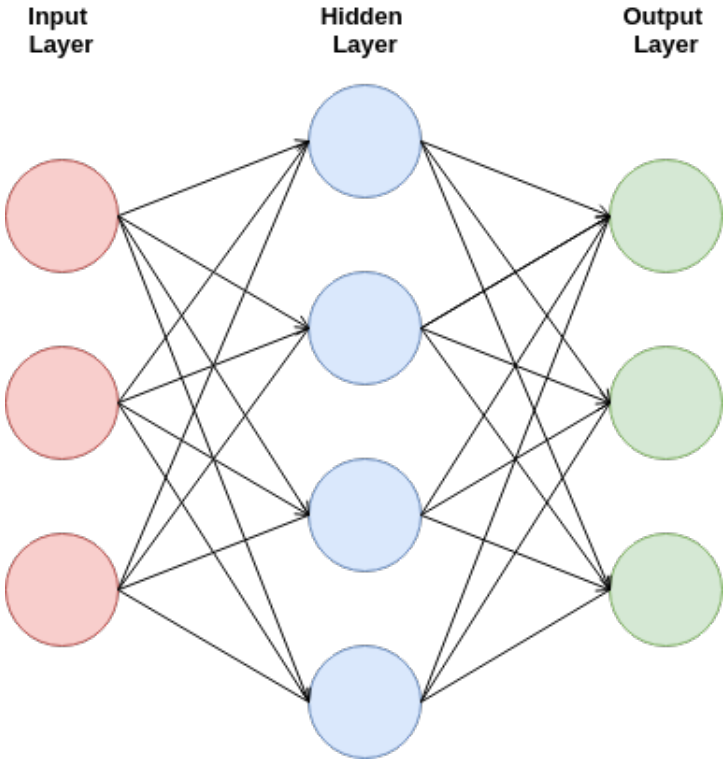


Figure 2.7: Simple illustration of a multilayer perceptron with three inputs, one hidden layer with four nodes, and one output layer with three nodes.

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 solution to solve both complications.

cite

Convolutional neural networks are the most popular form for image recognition, segmentation, and classification

cite, IEEE paper got cites

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.

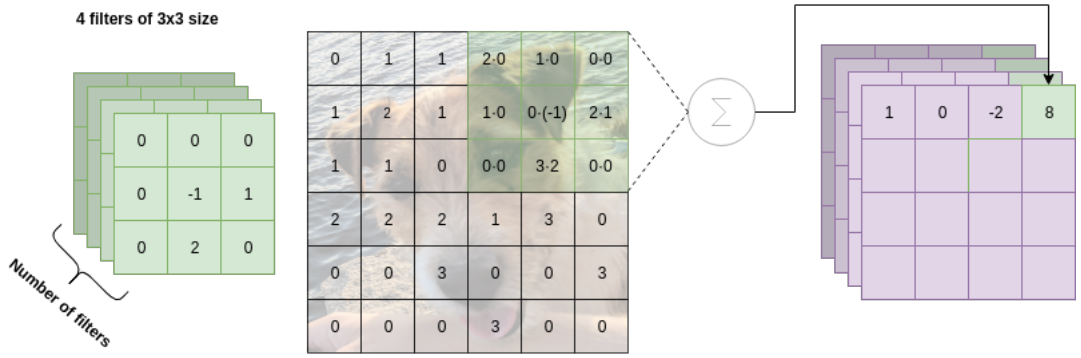


Figure 2.8: The values calculated when a convolutional filter after 4 sliding window operations, here the number of inputs does not represent how many inputs there usually is in an image

Convolutional layers

Convolutional networks work with filters as opposed to perceptrons with weights assigned before and after the input in the vertices between perceptron. 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 2.8 shows the this convolution process after 3 sliding operations with a (3×3) filter.

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 chapter 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.

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. 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

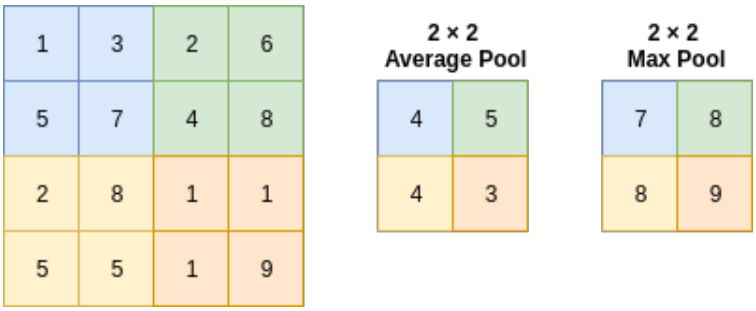


Figure 2.9: Both max and average pooling done on a 4×4 matrix

Just like in Figure 2.9 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

Normalisation layers are one of the newer addition to the CNN models. The normalisation layers, as with the pooling layers does not contain any weights, ad subsequently only change the data based on the statistics of the data.

In CNN models the most used type of normalisation is batch normalisation.

- Learning faster

- Increase 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.

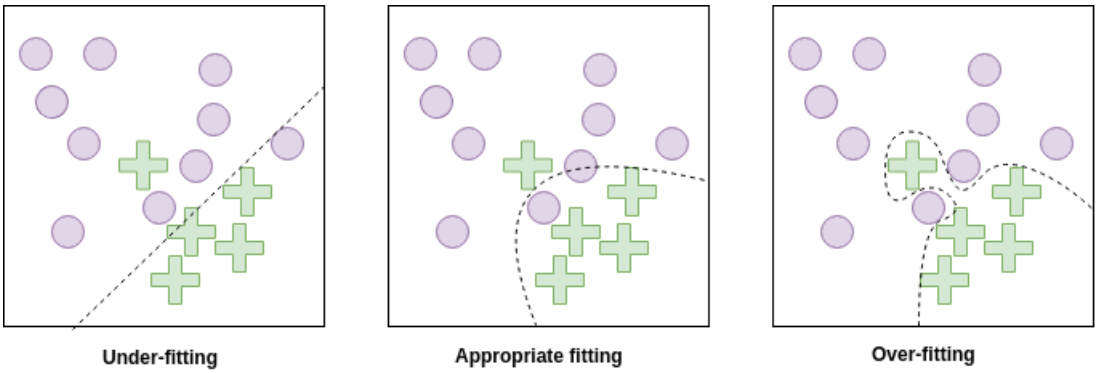


Figure 2.10: Three cases of data boundary prediction. In most cases we desire appropriate amount of fitting to our dataset to keep generalisation

Dropout, first proposed by Srivastava et al.[11], in 2014 was used to prevent overfitting of neural networks (As seen in figure 2.10). Dropout layers in the CNN will randomly cut a certain amount of connections in a layer. The amount is usually between 25% and 50% of the connections. 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 eavenly distributed throughout the network. Dropout does, in practice, the same as an L1 or L2 regularizer.

2.5 Complex Neural Network models

We have 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,

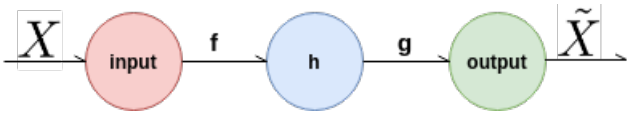


Figure 2.11: The general structure of an autoencoder, encoding **x** with **f**, then decoding **h** with **g** to an output **x**.

we need to present two other types of network, namely the Autoencoder [12] and the generative adversarial network [13].

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 **??**. 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. Figure 2.11 shows the general structure of the autoencoder.

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 between the encoder and the decoder. The reguliser might be that the output of the latent space layer should follow a Gaussian noise 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 2.12 shows the general concept of an inpainting Autoencoder.

As with supervised classifiers, autoencoders use gradient descent to

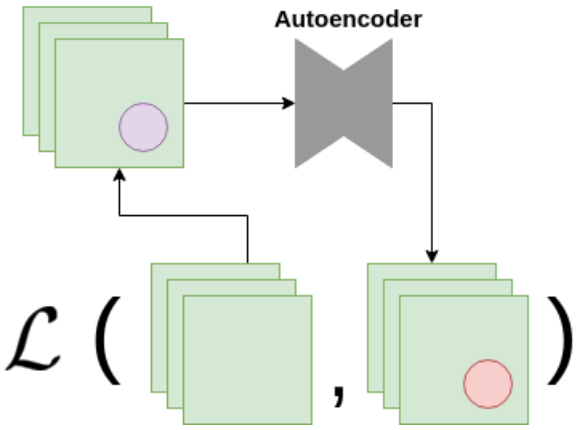


Figure 2.12: Autoencoder where the goal is to inpaint the masked area

optimise the result, as the model work to recreate the input $\mathbf{x}$ from out output $\tilde{\mathbf{x}}$ We achieve this by minimising the loss function:

$$\mathcal{L}(\mathbf{x}, g(f(\mathbf{x}))) \tag{2.12}$$

As this is a problem where the goal is to minimise error, autoencoders often use mean square error or mean absolute error as the optimiser for the gradient descent. Figure 2.12 show a visual example of equation 2.12.

2.5.2 Advaserial neural networks

This is explaining GANS, put me in the right place Now that we have looked at autoencoders we can take it a step further. generative adversarial models can be used as a generator of new data and can have some resemblance to autoencoders 2.5.1, especially variational autoencoders. [14]

The difference lies in that adversarial networks is based on game theoretic scenarios in which a generator network is competing against an advasery. 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 from the generator. More spessific, 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 $-v(\theta^{(g)}, \theta^{(d)})$. *$v$ is here a function that is determined by both the success rate of the discriminator and the generator, the most*

commonly used is

$$v(\theta^{(g)}, \theta^{(d)}) = \mathbb{E}_{x \sim p_{data}} \log d(x) + \mathbb{E}_{x \sim p_{model}} \log (1 - d(x)) \tag{2.13}$$

as derived from Goodfellow et al.

Lets look at a gan in detail.

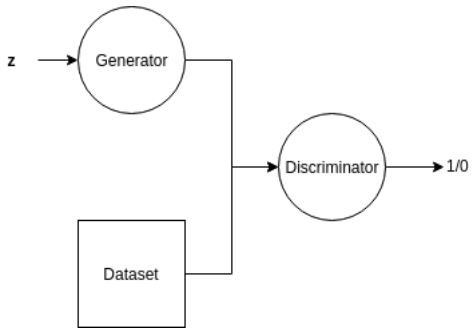


Figure 2.13: The idea behind a GAN. Here the generator samples from a random (Gaussian) distribution and generates samples that the discriminator classifies as real or fake

Contextencoders

Inpainting can also be done with adversarial models, and using a network trained to do the task of inpainting can be a lot more powerful than using just an autoencoder or the naive methods. A contextencoder is building on the adversarial principle by using a generator/discriminator pair to fill in masked areas in an image.

The concept behind a Contextencoder is to take the whole image as input to an encoder/decoder pair and

ref

ref

finish

CC-GANS

2.6 The problem at hand / Summary

Explain how the ML-methods can be used with the polyps

When you work with machine learning a lot of the job is to make the data as clear as possible.
Imagine that you want to do something as simple as reading an analogue clock.

The s̈traight forwardẅay to do it is to make a convolutional neural network to look at the dials. This will require a much more complex network compared to if you could convert the angle of the dials to degrees and have that as an input to your model.

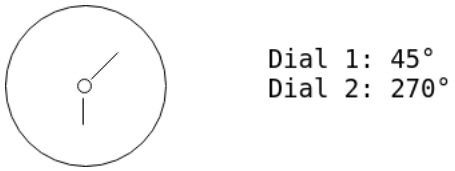


Figure 2.14: A clock needs a more complex network compared to just the degrees

Here I need to talk about how preprocessing might help the problem, and make this a springboard into the next section: the problem at hand

adress the problem with lack of data, and talk about how machine learning can help, machine learning.

The trick is often to make the data as refined as possible. Further, some of the techniques used are described.

- Stevens paper about inpainting
- Konstantins paper about the cross dataset evaluation

Now that we have the definition of machine learning and the current task, we can focus on the task at hand; finding polyps. In an ideal world[1] we have a Classification problem with only two classes: Non-polyp and polyp.

- SVM
- CNN
- random forests
- knn

[1] Ideal as in the only disease, we could get in the GI tract was cancer originating from polyps which looked exactly the same

Chapter 3

Methodology

With our background in both machine learning and we can now look at how we want to solve the problems associated with setting up a system for medical diagnosis. We will first look at the language and packages used in the creation of this thesis. We will go in depth into the reasoning behind why we chose the tools and packages that became the foundation of the programs.

medical background

Then we will look at the setup of the complete program. Here we will go in-depth into both the different preprocessing algorithms, and take a look at the transfer learning network used during classifying.

3.1 Birds eye view (Chapter removed when written)

3.2 Libraries

In this chapter, we will discuss the foundation of our code, important external libraries, and the setup and execution of our project. We will 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 will look into the wrapper we use to get a higher level of abstraction over our code, together with custom wrapper functions that are used by our wrapper.

3.2.1 Python

When doing machine learning, the most popular languages, in no particular order, are Python, Java, R, C++, and C . Some of these languages, like C and C++, are chosen for their speed, which is often a significant factor in Machine

cite

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.

3.2.2 Tensorflow

cite

year

cote

something about projects with python, and how may uses

Arguably the biggest reason for the success of machine learning in python lies in Tensorflow. Tensorflow is a machine learning package developed by Google in and has since then become the leading framework for machine learning worldwide . Tensorflow is in use by companies like AMD, Nvidia, 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.

cpu vs gpu

, CNTK, or Theano

3.2.3 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 was made as 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. The most successful library for on top of Tensorflow is Keras . Just as TFlearn, Keras is a high-level package written in python. It is capable of running on top of TensorFlow, CNTK, or Theano, which is the tree most popular machine learning libraries at this time. From their website they state that their four goals when creating Kears were: **User friendliness.**
Modularity.
Easy extensibility.
Work with Python.

cite TFLEARN

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.

more

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. *Since one of our primary goal is to see how well our datasets generalise to the real world, transfer-learning will be a great tool to forgo unnecessary training*

3.3 Custom functions for Keras, tensorflow and python

3.3.1 CWFC layer

A problem often encountered when working with autoencoders which are not undercomplete is the fact that they learn to represent the data flawlessly. [15] 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 content encoders[16].

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) [16]

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 1 shows the source code used for the channel-wise fully-connected layer.

3.3.2 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 generative adversarial networks has been a challenge that has only recently been cracked. [17] [18] As an early measure to fix this problem, we experimented with the use of a Sub-pixel layer as presented by Shi et al. [19] to give a more realistic output compared to just a standard conv-tanh layer.

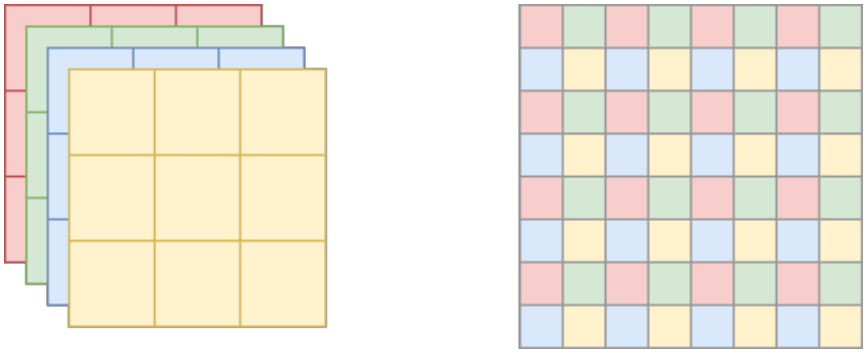


Figure 3.1: How the layers in the sub pixel layer is stacked. Recreated from the SubPixel paper by Shi et al. [19]

```
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 1: The channel-wise fully-connected layer source code

3.3.3 Masklaod

3.3.4 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 a semantically meaningful image, we often want to differentiate between background and the mucosa. To see if the network can learn the features needed we are Introducing the Self Attention layer to help with this.

3.3.5 Masked loss

3.4 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.

```
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 2: The self attention layer source code

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 [1].

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.

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

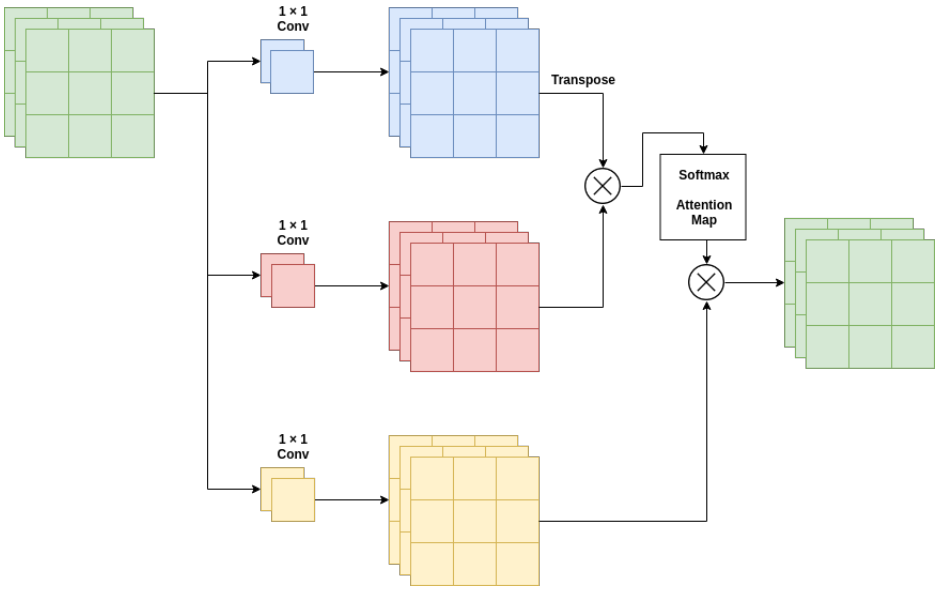


Figure 3.2: How the layers in the Self-Attention layer is stacked. Recreated from the Self-Attention paper by Zhang et al. [20]

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[21] activation function [22] [23] [24]. We saw the best result when we used non-sparse gradients during training. Instead of using ReLu we used the slightly modified LeakyReLu [25].

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

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

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.

3.5 Design of the inpainting algorithms

When inpainting we have multiple hypotheses we want to test to see how it affects the results. 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. Based on the hardware limitations

more

When generating the datasets, we use the Kvasir dataset as a base exclusively. When using only the Kvasir dataset, we have the CVC sets 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 **??** shows two different image types from the Kvasir dataset. Figure 3.3a shows the image class esophagitis. This image shows one of the main problems with the Kvasir dataset when it comes to dataset specific 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. We can recall from chapter that the area to the left does not contain any relevant information for the classification and subsequently can only give false information during classification.

find where

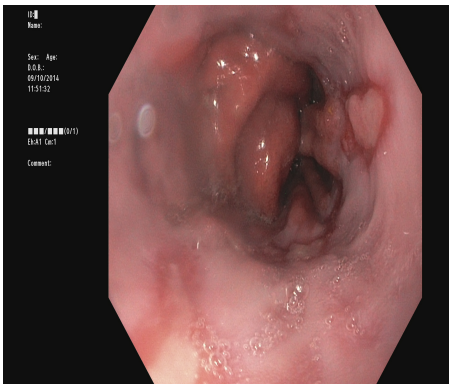
Compared to the CVC images we have images in the Kvasir set with this square and with additional text, both outside the image, and on top of the image.

We propose different types of inpainting to prove or disprove our two hypothesises.

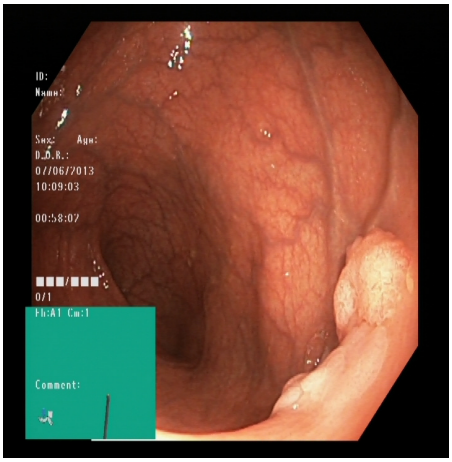
3.5.1 Removing black corners

The most straightforward experiment to conduct is to test how the removal of the black corners will affect the result. The black edges around the images in the Kvasir set is also present in the CVC set, and the Nerthus set too. 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.

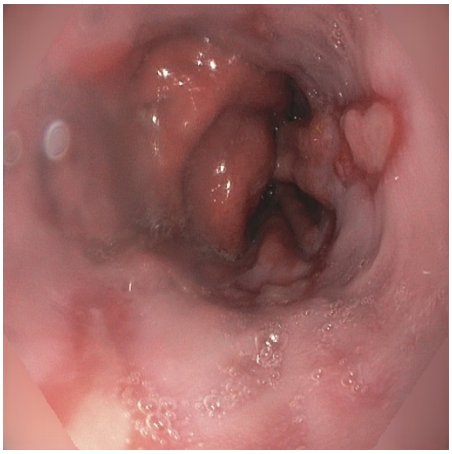
When classifying images during the testing stage, we need to take this mask in to account regarding the removal of black corners in the test set.



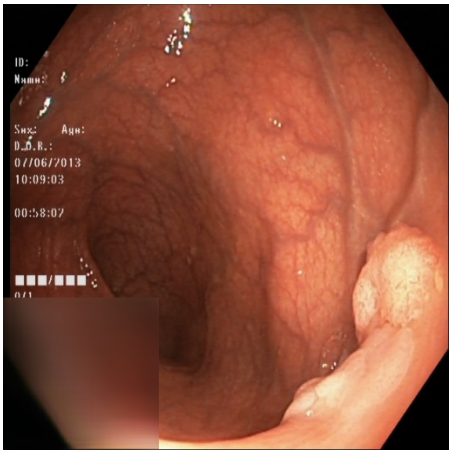
(a) Example of an image with a large area without relevant medical information



(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

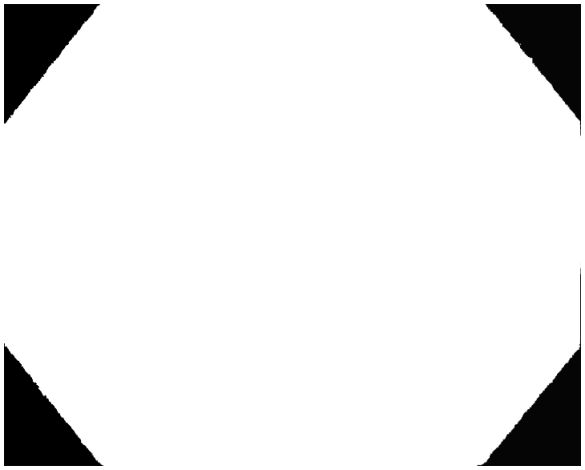


Figure 3.4: The automatially generated mask used to remove the black corners

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 black corners without inpainting.
3. We can forgo modifying the test data and just run the test set as is.

Method 1) was tested in the paper by Kirkerød et al. in the Mediaeval 2018 conference, where both the training and test data were augmented. From the experiments run at the mediaeval conference, we have some essential information we can take into account. Training data used to train the preprocessing tools were data from the same dataset as Kvasir, giving the project and thesis an overlap in the data distribution during training. The evaluation of the test set was done in beforehand, and not live. Also, there was no focus on evaluation speed when running the classification on the test set. The Medico test set used in the Mediaeval conference was also gathered from the same distribution as the Medico test set. This use of the same distribution means that if we are going to compare methods of masking the test set, we should only look at the Kvasir dataset when comparing the masking methods, given the CVC dataset will not give sufficient comparability. Also, given that our goal with the medical classification is, in the future, to have the opportunity for live classification. We have chosen in this thesis not to convert medical images during testing given the time used during a live evaluation would not be suitable.

more about the result of testing inpainting.

Method number 2) was the proposal of cropping the images during evaluation. Thambawita et al. did similar methods in the Mediaeval 2018 conference, but had, in addition, the same cropping during training. In the

paper 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.

find Vajiras paper

The last proposed method is not to augment the images during testing. This method, since we are not preprocessing the test data at all, the fastest when it comes to live classification. Without the augmentation, we risk getting a lower classification score.

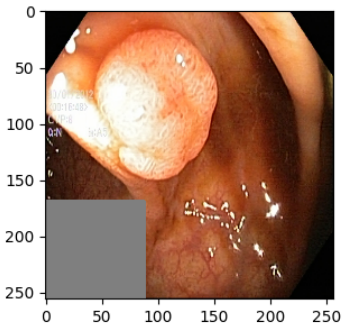
In this thesis we will use method number 3). The decision is based on the fact that the final product would most likely be used in cooperation with medical staff, and hence the image should be as close to the feed from the pillcam/colonoscope. Method 2) might also work, but we remove data from the image that the medical staff might need for their classification.

3.5.2 Removing green squares

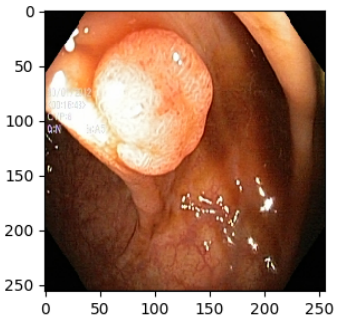
The next major area in question is the removal of the green squares. The experiment would see if the removal of the green ḿinimapẃould affect the classification score.

Figure 3.5: The automatially generated mask used to remove the black corners

The removal of the green squares hypothesis 2



(a) Image the autoencoder receives as an input



(b) The missing part the autoencoder tries to replicate

Figure 3.6: A standard image taken in by the autoencoder

3.5.3 Removing black corners

3.5.4 Removing black corners

3.6 Design of the transfer learning experiments

3.7 Describe code

3.7.1 autoencoder

The autoencoder we used in this thesis bears some resemblance to the standard autoencoder by . 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.

x et al

The next step before training is to cut the images according to the mask provided. Figure 3.6a shows what the finished masking looks like, and 3.6b shows what we want to achieve after training.

We feed 3.6b 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, but does not take in noise

the best version does

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.

Table 3.1: Datasets provided by keras

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 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 Describe project

Here i am going to explain the projects

3.9 Summary

At this point, we have described the reasoning behind using python, tensorflow and Keras for our machine learning. After this we looked into the filters used for training our models, and what type of preprocessing we wanted to do in addition to the inpainting. We looked at the advantages and disadvantages of preprocessing the Trest data, as opposed to just preprocessing the training data.

We then went more in-depth into how the three main programs were built up, and how they differ from their original sources. First, we looked at the preprocessing methods, AE and GAN. We looked at how they learned from the samples, and we went more in-depth talking about the crucial layers for each model.

At last, we ended up giving a summary of the project as a whole, following a the Kvasir dataset from its original source into the generation of the six new datasets, and how each one of them was classified with all the different parameters to show the dataset-specific rate of success.

Chapter 4

Experiments

In the previous chapter, we described our methodology. 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 four datasets to train and evaluate our results.

something about Simula already researching on this

4.2 Metrics

To discern the results of our experiments we introduce multiple metrics and tables to get an indication of our success. The main 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 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.

img of k fold

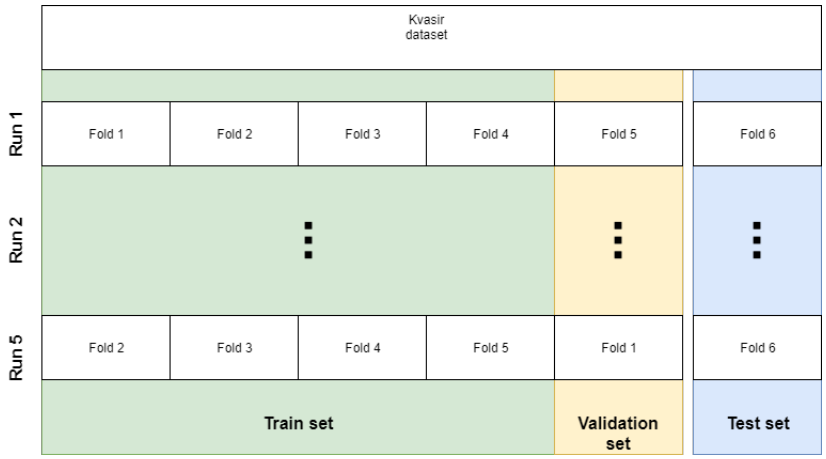


Figure 4.1: The Kvasir divided in to 6 folds

The advantage of this method 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 Presentation of data, 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 4.2

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

Figure 4.2: An empty confusion matrix

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

Figure 4.3: The confusion matrix with [3 5] and [0 0] inserted

After we have initialised the confusion matrix, we add each value pair as to the matrix at is 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). Witch 2 examples we get the following confusion matrix shown in Figure 4.3

As we fill in the matrix with more predictions we can start to draw conclusions from it. After approximately 1600 evaluations, our result might at the end look like this. 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 specific class is when it is predicted

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

Figure 4.4: The confusion matrix with almost 1600 predictions

positive, and the True label is also positive. In the Kvasir dataset, we have a True positive result 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 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 given no polyp present.

False Negative (FN): False positive is the case where we fail to predict the class when it is True. This type of error is often called "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.

For our multiclass confusion table, we can look at the four metrics like this

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 simply 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 accuracy is the lack of comprehensive

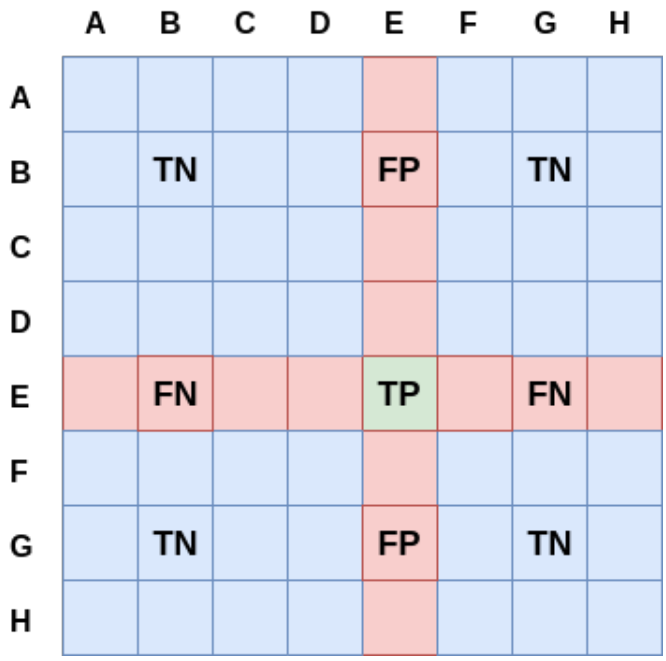


Figure 4.5: Confusion matrix with eight classes, here True positive is marked in green, False Negative and False positive marked in red, and True negative in blue.

overview of the data, as it is just a summation without respect to classes involved. In this project, we use accuracy during the training step as a metric of success.

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

Recall: Recall is the probability of detection, or 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. For our medical image classification, this metric can help us understand how our algorithms work by looking at classes where we have small anomalies in the images, like the case with polyps.

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

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. It is one of the most common metric to compare the performance of statistical models. Equation 4.5 shows the F1 score as it is most commonly used.

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

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.

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

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 shown in fig

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

Here we can calculate the weighted average recall to be **x**. 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 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.

4.3 Setup of experiments

We propose 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.

and

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 will set up our experiments to test our two hypothesises. 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. In addition to just comparing accuracy, we will also look especially at Recall of the polyp class, and the MCC score. As mentioned in chapter and the , we have datasets which are unbalanced, and subsequently, need a more representative score compared to F1 or accuracy. We can also recall that the recall score gives us a probability of detection in a binary class situation. We will use this to see if any one of the generated datasets modifies this score in any manner.

find MCC chapter

find dataset chapter

2x recall

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.

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 can 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 assume 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.

Table 4.1: Details of all datasets we generate in the experiments.

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

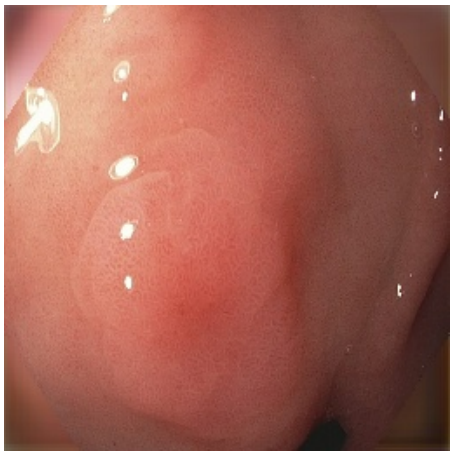
Dataset labels	Size	Inpainted area	Generator network used
I - Base Case	256x256 px	-	-
II - Autoencoder with black corner	256x256 px	BC	Autoencoder
III - Autoencoder with green square	256x256 px	GS	Autoencoder
IV - Autoencoder with both	256x256 px	BC+GS	Autoencoder
V - GAN with black corner	256x256 px	BC	GAN
VI - GAN with green square	256x256 px	GS	GAN
VII - GAN with both	256x256 px	BC+GS	GAN

4.4.1 Black corners

When generating the two first datasets, we used the mask shown in Figure 3.5. Here we did two operations, first cropping, then inpainting. The result is shown in Figure 4.6 and Figure 4.7.

Here we have presented representative images from both datasets inpainting the four edges around the image. Figure 4.6 show images that, for both dataset are well inpainted, and 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.

4.7 show images that are more problematic. In Figure 4.7a the image generated by the Autoencoder has drawn its colour from the nearby white



(a) Image from the polyp class at 256x256 px, generated by an autoencoder

(b) Image from the polyp class at 256x256 px, generated by a GAN .

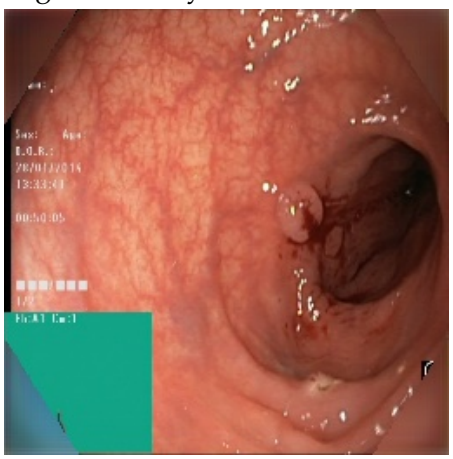
(c) Image from the normal-z-line class at 256x256 px, generated by an autoencoder

(d) Image from the normal-z-line class at 256x256 px, generated by a GAN

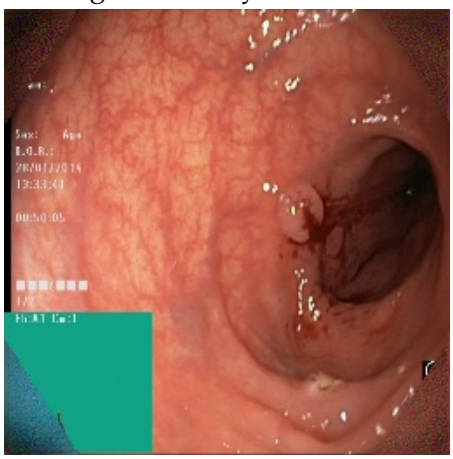
Figure 4.6: Two images from the polyp class and the z-line class. Images on the left were inpained with an Autoencoder, and the images on the right were inpainted with a GAN

(a) Image from the ulcerative colitis class at 256x256 px, generated by an autoencoder

(b) Image from the ulcerative colitis class at 256x256 px, generated by a GAN



(c) Image from the polyp class at 256x256 px, generated by an autoencoder



(d) Image from the polyp class at 256x256 px, generated by a GAN

.

Figure 4.7: Two 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.

oversaturated area, and subsequently, it has misdrawn the corner. In Figure 4.7b 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.

4.4.2 Green square

The next two datasets were generated from the mask in Figure **??**. The goal of the datasets generated by the Autoencoder and the GAN is to remove the dataset-specific green square.

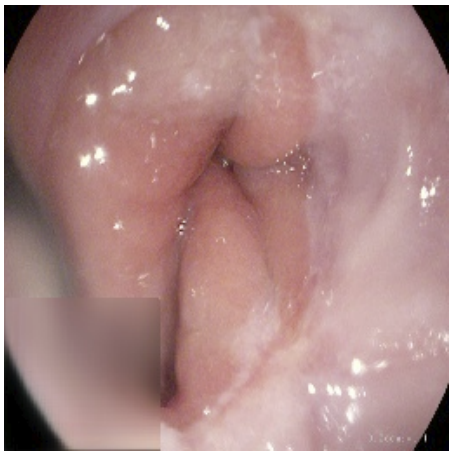
Figure 4.8 and Figure 4.9 shows two exaples from the datasets with the green square inpainted. The images in both figures showcase in detail how the different algorithms inpaints larger areas. rewrite

Figure 4.9d and Figure 4.8c shows examples where the GAN has made a better prediction of the complex structures that lie behind the masked area. We also have an example where the network gets an unconventional input, resulting in discolouration of the target area.

4.4.3 Combination

(a) Image from the polyp class at 256x256 px, generated by an autoencoder

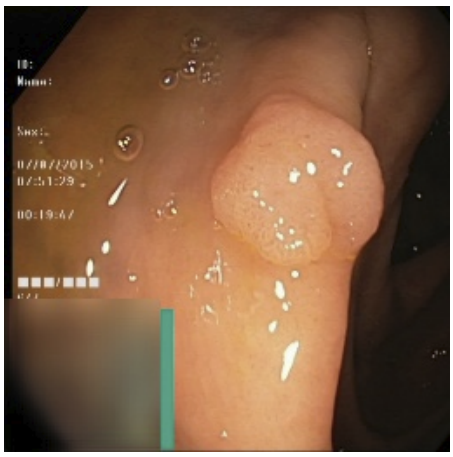
(b) Image from the polyp class at 256x256 px, generated by a GAN .



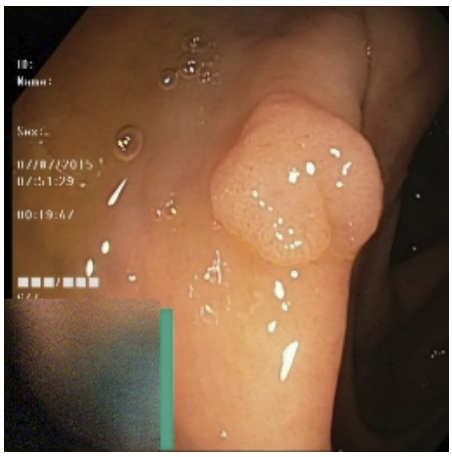
(c) Image from the normal-z-line class at 256x256 px, generated by an autoencoder

(d) Image from the normal-z-line class at 256x256 px, generated by a GAN

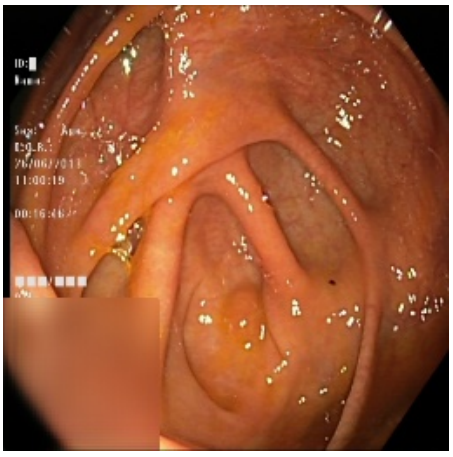
Figure 4.8: Two images from the polyp class and the normal-z-line class. Here we see results that needed finer detail when inpainting.



(a) Image from the polyp class at 256x256 px, generated by an autoencoder



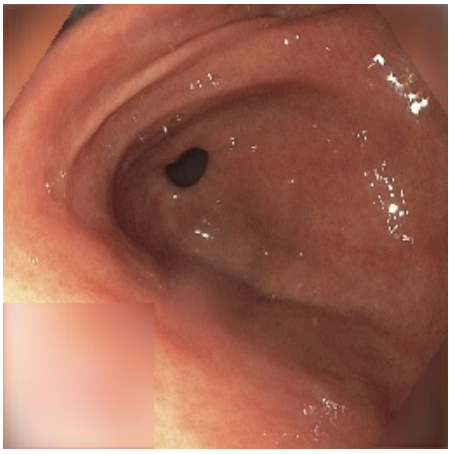
(b) Image from the polyp class at 256x256 px, generated by a GAN
.



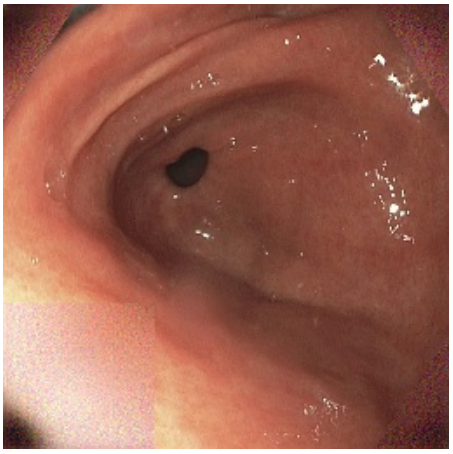
(c) Image from the normal-cecum class at 256x256 px, generated by an autoencoder

(d) Image from the normal-cecum class at 256x256 px, generated by a GAN

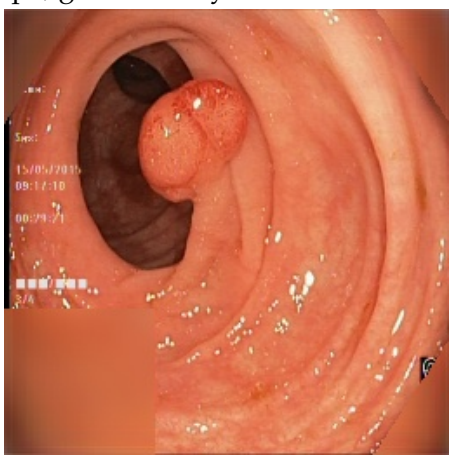
Figure 4.9: Two images from the polyp class and the normal-cecum class. Here we have images with a problematic green square, and an image with non-essential details.



(a) Image from the normal-pylorus class at 256x256 px, generated by an autoencoder



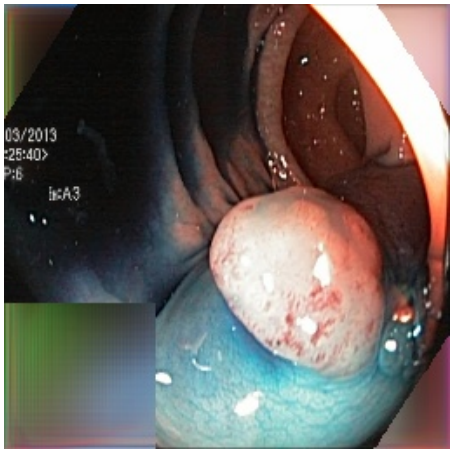
(b) Image from the normal-pylorus class at 256x256 px, generated by a GAN



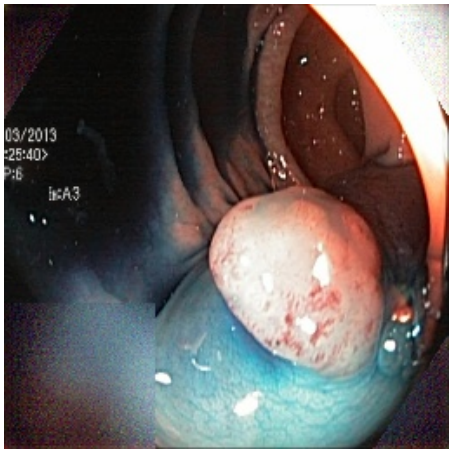
(c) Image from the polyp class at 256x256 px, generated by an autoencoder

(d) Image from the polyp class at 256x256 px, generated by a GAN .

Figure 4.10: New text here



(a) Image from the dye lifted polyp class at 256x256 px, generated by an autoencoder



(b) Image from the dye lifted polyp class at 256x256 px, generated by a GAN

(c) Image from the polyp class at 256x256 px, generated by an autoencoder

(d) Image from the polyp class at 256x256 px, generated by a GAN

Figure 4.11: New text here

4.5 Results of the transfer learning experiments

In the last chapter, we looked at the generation of the new datasets. *we looked at the parameters behind the creation of each of them*, and we discussed downfall and advantages with the autoencoder method versus the GAN method.

We are now looking 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 clssification models was as follows. First, we supply info about the model. This information is for instance batch size, type of network, and choice of an optimiser.

The network chosen is loaded with the imagenet weights. With the model loaded into memory, a 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.

here we talk about K-fold

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.

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.

Table 4.2: Attributes for the training

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

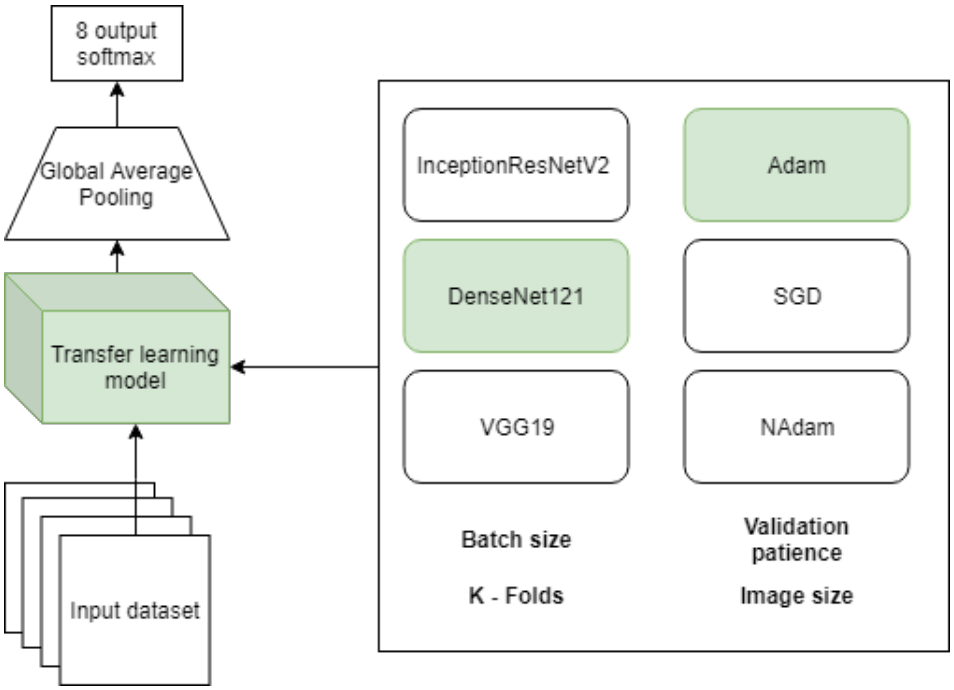


Figure 4.12: The model we use for classifying with the most important options for the learning process.

4.6 Densenet121

ref saga/rune

Based on the hyperparameter-optimiser SAGA we have the optimal network to achieve the highest score when both training and evaluating on the Kvasir dataset. The Densenet121 model gets its name from the use of fully connected blocks between the convolutional layers.

Based on the SAGA 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 4.3.

Table 4.3: 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

The parameters used are to get the optimal amount of training without overfitting to the data. 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.

4.6.1 Densenet121 base model

We present the results with Densenet121 in table 4.36. This Table shows the base case result with the Densenet121 model.

Figure 4.13: Densenet121 Base 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	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

Figure 4.14: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.15: The baseline confusion matrixes

Matrix 4.14 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.

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.

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 an near perfect accuracy given the complexity of the dataset.

CVC 12k The CVC 356 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_1 we would expect the act of inpainting the corner in the images to give Kvasir a higher score complared to the baseline. In addition, we could see improvement in both CVC 356 and CVC 12k. We present the GAN followed by the Autoencoder and evaluate them together.

Figure 4.16: Densenet121 Inpainted corners 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	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

Figure 4.17: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.18: The inpainted corner with GAN confusion matrixes

Figure 4.19: Densenet121 Inpainted corners with the AE 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	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

Figure 4.20: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.21: The inpainted corner with AE confusion matrixes

Figure 4.21 and 4.18 shows the evaluation of the test set on each of the three datasets used both for 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.

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.

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 Square inpainted result

We can recall from Hypothesis H_0 that removing dataset specific artefacts we will achieve a higher classification score when evaluation our models on previously unseen datasets. We expect the Kvasir dataset not show any improvements, though both the CVC 356 and the CVC 12k dataset we expect to improve compared to the base case. We present the GAN followed by the Autoencoder and evaluate them together.

Figure 4.22: Densenet121 Inpainted green square 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	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

Figure 4.23: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.24: The inpainted green square with GAN confusion matrixes

Figure 4.25: Densenet121 Inpainted green square with the AE 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	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

Figure 4.26: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.27: The inpainted green square with AE confusion matrixes

Figure 4.24 and 4.27 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_0 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 a 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.

This 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 evaluation on the Kvasir dataset. Here both the GAN and the Autoencoder got a lower score compared to the base, though only with a small margin.

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 evaluation on the dataset.

This disproves hyp:a

4.6.4 Both inpainted result

In addition to test the two hypothesises separately we, as we recall, want to test both types of inpainting at the same time. When testing both inpainting types we can not draw any predictions reliably, 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.

Figure 4.28: Densenet121 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	1916	129
G	12	227

	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

	I	G
I	1917	7851
G	12	2174

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

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

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

Figure 4.29: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.30: The inpainted both areas with GAN confusion matrixes

Figure 4.31: Densenet121 Inpainted both areas with the AE 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	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

Figure 4.32: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.33: The inpainted both areas with AE confusion matrixes

Figure 4.30 and 4.33 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_0 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 a 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.

This highly suggest that our hypothesis about removing dataset-specific artefacts to improve accuracy is indeed correct.

this text is not done yet!

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

this text is not done yet!

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 evaluation on the dataset.

this text is not done yet!

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 earlier, Densenet 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 4.4.

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 **??**. With the patience of four we, on average, train longer and hence get a more stable result without overfitting.

Table 4.4: 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

4.7.1 Inceptionresnetv2 base model

We present the results with InceptionResNetV2 in table 4.36. This Table shows the base case result with the larger Inceptionresnetv2 model.

Figure 4.34: InceptionResNetV2 Base 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	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

Figure 4.35: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.36: The baseline confusion matrixes

From the results we see the general score compared to a Densenet model as being lower by a reasonable margin. We can also discern that the base case has an MCC value close to 0 when evaluating on the CVC12k dataset.

better comparison beteween this and base DN121.

4.7.2 Inceptionresnetv2 Inpainted Corner

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 4.37: InceptionResNetV2 Inpainted corners 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	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

Figure 4.38: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.39: The inpainted corner with GAN confusion matrixes

Our results from figure 4.39 shows a clear advantage when evaluating the results on the CVC 12K dataset while keeping the result the same on the CVC 256 and Kvasir dataset. We can not draw any clear conclutions.

Figure 4.40: InceptionResNetV2 Inpainted corners with the AE 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	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

Figure 4.41: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.42: The inpainted corner with AE confusion matrixes

Our results from figure 4.42 shows the same advantage when evaluating the results on the CVC 12K dataset. In we got a lower score when evaluating our data on the CVC 356 dataset compared to the base case.

We can conclude that the act of inpainting the corners to remove sparseness gives the same result with IRV2 as with Densenet. We see no clear advantage to inpaint the corners when using either method or model for inpainting.

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 4.43: InceptionResNetV2 Inpainted square 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	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

Figure 4.44: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.45: The inpainted square with GAN confusion matrixes

Our results from figure 4.39 shows a clear advantage when evaluating the results on the CVC 12K dataset while keeping the result the same on the CVC 256 and Kvasir dataset. We can not draw any clear conclutions.

Figure 4.46: InceptionResNetV2 Inpainted square with the AE 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	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

Figure 4.47: Confusion matrices for the three datasets

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

(a) The CVC 356 dataset Metrics

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

(b) The Kvasir dataset Metrics

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

(c) The CVC 12k dataset Metrics

Figure 4.48: The inpainted square with AE confusion matrixes

Figure 4.45 shows a definite improvement compared to the base case IRV2 model when using the inpainted dataset generated by the GAN. Both for the CVC 356 dataset as well for the CVC 12k dataset we have genuine improvements.

The Autoencoder shows similar good results in the same datasets as with the GAN in figure 4.48.

From our results, we can conclude with the fact that the IRV2 model manages to generalise itself better when presented with the inpainted data, just as we concluded with the Densenet model.

4.8 Classification results based on the Densenet model

4.9 Classification results based on the InceptionResnetV2 model

4.10 Classification results

4.11 Classification results

4.12 Classification results

4.13 Classification results

4.14 Classification results

From the results, we see that inpainting the green square gives a better MCC score, compared to the base case When dealing with the images at 256 resolution.

4.15 summary

Chapter 5

Result and Discussion

5.1 How to view the result?

5.1.1 The rate of success

What is a good result, how to measure?
FP,TN,FN,TP

Chapter 6

Conclusion

Chapter 7

Future Work

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

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. The option to remove text using an Optical Character Recognition (OCR) has the foundation it needs, but the time used by OCR algorithms are to slow to work in real-time. Combining this system with a system like might give a speedup, but at this point requires a multi-GPU setup to work. OCRMODELS

Another possible future branch is to look at entirely generated medical images as training data of a classifier.

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] B. Stewart and C. Wild, *World Cancer Report 2014*, ser. International Agency for Research on Cancer. International Agency for Research on Cancer, 2014, ISBN: 9789283204299. [Online]. Available: https://books.google.no/books?id=0QHbngEACAAJ.

[3] 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.

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

[5] 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.

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

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

[8] 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.

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

[10] 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".

[11] 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.

[12] 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.

[13] 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.

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

[15] ——, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

Chapter 8

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
Simula Metropolitan Center for Digital Engineering, Norway
University of Oslo, 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).

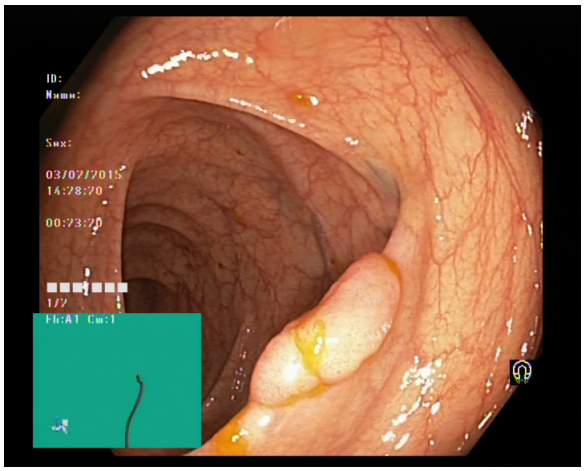


Fig. 1: Example image from the Kvasir dataset with included overlays and black borders.

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.

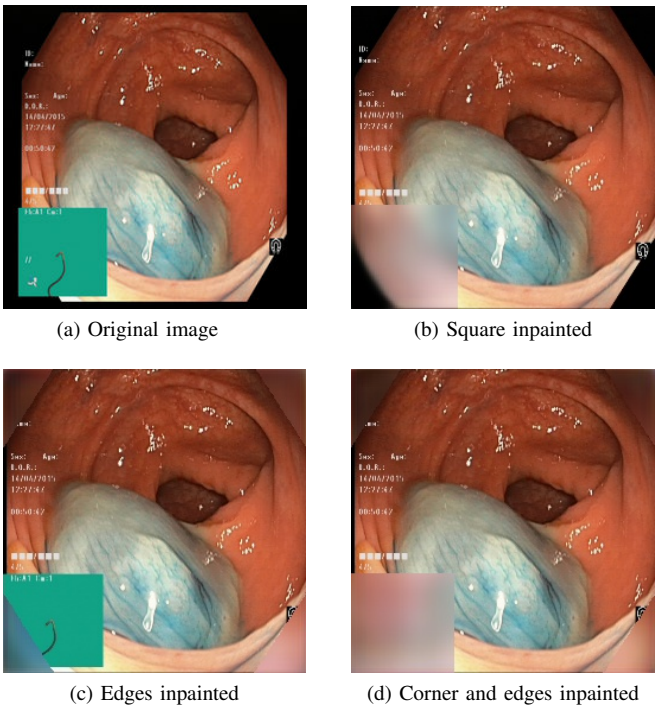
Previously, Hicks et al. [5] applied various preprocessing steps to Kvasir based on analysis conducted on common CNN architectures. Using heat maps and saliency maps, they discovered a common issue where artefacts such as text, black borders, and green navigation boxes were directly correlated to the 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 the best result. However, we expect improvement if we instead

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

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

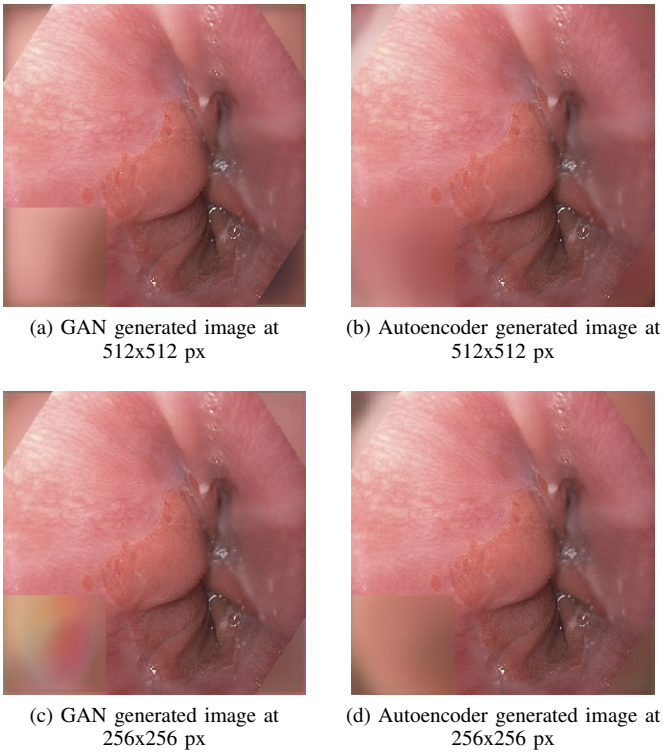
We divide our results into two sections, preprocessing and classification. In our preprocessing section, we discuss the appearance of the dataset, and how close the results are to the ground truth. In our classification section, we discuss the rate of generalisation and rate of success.

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 inpainting the black corner. The results do not show a clear

TABLE III: Test T1, Kvasir dataset on DenseNet121

Dataset	MCC	F1	Precision	Recall	Average MCC
D-I	0.9307	0.9394	0.9396	0.9394	0.9163
D-II	0.9150	0.9254	0.9303	0.9250	0.9053
D-III	0.9212	0.9310	0.9347	0.9306	0.9040
D-IV	0.9187	0.9287	0.9298	0.9288	0.9105
D-V	0.9208	0.9308	0.9316	0.9306	0.9108
D-VI	0.9096	0.9204	0.9226	0.9206	0.9055
D-VII	0.8960	0.9094	0.9174	0.9081	0.8926

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

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

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.

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

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

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

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

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 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

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

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

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