Synthesizing Skin Lesion Images Using Generative Adversarial Networks

A case study

Sondre Fossen-Romsaas

Adrian Storm-Johannessen

Master's thesis in Software Engineering at

Department of Computing, Mathematics and Physics,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

June 29, 2020

Acknowledgements

We would like to thank the Mohn Medical Imaging and Visualization Centre (MMIV) at Haukeland University Hospital for giving us the best work environment possible, while working with our master thesis. A special thanks to the ML group at MMIV for hosting meetings with many interesting and exciting discussions.

We would also like to thank PhD candidate Sathiesh Kaliyugarasan at MMIV and our co-student Sivert Stavland for many educational discussions around the topic Deep Learning, and making this period a lot of fun.

Finally, a special thanks to our supervisor Dr. Alexander Selvikvåg Lundervold for introducing us to all of this. His great commitment and enthusiastic help have given us a lot of motivation to give our best during this period.

Abstract

According to the World Health Organization cancer is the second leading cause of death globally [1], and the most common cancer in the world is skin cancer [2].

To get a skin cancer diagnosis, a patient will usually mention their concern to a doctor. The doctor will then refer the patient to a dermatologist, who takes a closer look to determine whether the lesion is abnormal and in need of further inspection. If it's abnormal, the next step is usually a biopsy of the lesion for further testing in a laboratory. The test results determine the final diagnosis.

The dermatologist uses an array of different techniques and tools to examine the lesion, one of which is often a dermoscope. A dermoscope is a handheld device that makes it possible to see structures in the skin that normally are invisible to the naked eye. Nowadays dermoscopes are usually equipped with a camera, allowing the physician to digitally store images of the lesion. This makes it easier to get a second opinion. Either from another physician, or in the interest of this thesis, from some state-of-the-art automated skin cancer detection system, i.e. computer-aided diagnosis (CAD).

The idea of having CAD tools act as second opinions has become more plausible than ever, with the recent improvements in machine learning (ML) seen in the last decade. Most importantly, the last years developments within the sub-fields of neural network (NN), deep learning (DL), and computer vision. This have caused plenty of new startups, companies, software, and applications for medical image analysis. In the subject of skin cancer, there has been a great deal of activity around the field of machine learning, ranging from simple mobile applications to comprehensive research into clinical utility.

In 2017, Andre Esteva et al. published a paper which showed that machine learning models can perform at the level of dermatologists [3]. In 2018, H. A. Haenssel et al. constructed a convolutional neural network that in most cases outperformed 58 dermatologist. They claimed that any physicians, regardless of experience, may benefit from including a machine learning model in their evaluation [4].

A main reason for the success of neural networks is their ability to automatically extract useful features from data (*representation learning*). Data can also be said to be the methods greatest weakness: the performance of the model becomes extremely dependent on the quantity and quality of the data its being fed. The problem of acquiring high-quality data is particularly challenging when working with medical data. Because of the highly restrictive privacy protection of sensitive patient data, the resources needed for correctly labeling it, and the highly imbalanced nature of medical data, the process of collecting medical data is both difficult and expensive.

In this thesis, we will try to reduce the problem of low amounts of data

in machine learning by artificially creating more data using generative models, more specifically Generative Adversarial Network (GAN). We will create synthetic images of skin lesions, expanding the original data set. In addition to generating visually accurate synthetic data, our goal is to improve machine learning models by adding synthetic training data.

Figure 1: Examples of synthetic images of skin lesions generated by our models.

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Machine learning and generative adversarial networks in medicine

All the breakthroughs in machine learning, and deep learning the last years, have resulted in a lot of work for developing CAD tools in medicine, and medical imaging in particular [5]. Take for example the field of dermatology, a lot of scientific work have shown that a state-of-the art machine learning model is at least at the same level as a dermatologist [3, 4, 6, 7]. Another example of this is detection of diabetic retinopathy in photographs of retinal fundus [8, 9, 10]. If this is discovered, it is possible to reduce vision loss and prevent blindness for diabetic patients. The research within this domain have come far, and the tool IDx-DR is developed and able to detect diabetic retinopathy without human interaction. This is actually FDA-approved and used at hospitals in USA.

In recent years generative adversarial networks (GAN) have shown impressive work in generating realistic looking images e.g. BigGAN [11], StyleGAN2 [12] and LOGAN [13]. Today, generative models can produce synthetic images that are almost impossible to differentiate from real images (Fig. 1.1).

Figure 1.1: Images of generated faces produced with NVIDIA's state-of-the-art GAN model StyleGAN2 [12]. This figure is published under Nvidia Source Code License-NC. For more information see: `https://github.com/NVlabs/stylegan2/blob/master/LICENSE.txt`

Yann LeCun, the director of Facebook AI, professor at NYU, recipient of the 2018 Turing Award, and a prominent figure in deep learning, has called GANs and adversarial training *"the most interesting idea in the last 10 years in machine learning"*.

The extreme popularity of GANs is reflected in the exponential growth of published papers about GANs, see Figure 1.2. A major reason for the interest in synthesizing images using GANs is the never ending challenge of data hungry models in machine learning. By generating your own synthetic data, one can potentially partially solve this problem in many cases.

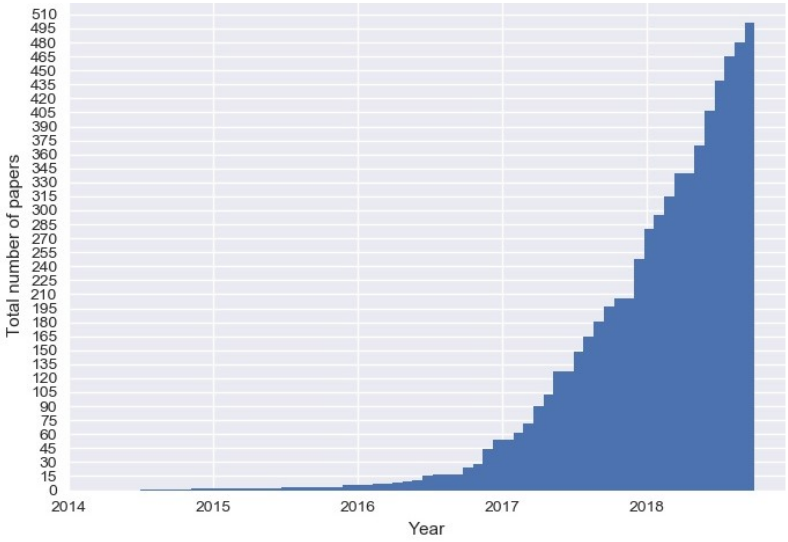

Figure 1.2: Cumulative number of named GAN papers by month from 2014 to September of 2018. This figure is published under the MIT License. For more information see: `https://github.com/hindupuravinash/the-gan-zoo/blob/master/LICENSE`

This is particularly relevant for the field of medical imaging, where synthetic images could help mitigate some of the distinct problems they are faced with. One set of such problems are those related to the protections of patient privacy. Privacy protection is a real concern when medical images are distributed, but synthetically generated images are not related to any particular person, meaning that there are no privacy to protect. Another problem is data imbalance. Which is to say that it can be difficult to collect a data set where all diseases are equally represented. This happens because some diseases are rarer then others. This would no longer be as problematic if one could generate an unlimited amount of synthetic samples for those cases that are poorly represented.

The task of classifying skin lesions is a case in point. To use supervised machine learning models, sufficient high-quality training data is needed, but the issue of privacy protected data and data imbalance makes successful development of high-performing models challenging. In our work, we have taken a deeper look into skin cancer, and tried to generate realistic looking images of skin lesions.

1.2 Skin Cancer

The growth of abnormal cells in the outer layer of the skin (epidermis) causes skin cancer. The underlying mutation of cells is in most cases triggered by DNA damage to the skin, caused by UV rays from the sun or from tanning machines. The type of skin cancer is determined by the type of cell the cancer evolves from [14, 15].

1.2.1 Common Types

The three most common types of skin cancer are basal cell carcinoma (BCC), squamous cell carcinoma (SCC), and melanoma [14]. The BCC and SCC is often named with the common term nonmelanoma

Cancer that starts from the basal cell is BCC, the most common of all types of skin cancer. This type of cancer does not have the ability to spread to other parts of the body, making it harmless in most cases. The SCC, on the other hand, has the ability to arise from the squamous cell and spread through the body. If it is not treated correctly, this type of tumor can be life threatening [15].

Melanoma only occurs in 4% of the cases where skin cancer is diagnosed, but it is the most aggressive type. It is responsible for 75% of all deaths related to skin cancer [16]. In 2015 it was over 350 000 cases of melanoma, where almost 60 000 resulted in deaths [17]. Additionally, reports are suggesting that the number of cases involving melanoma of the skin is increasing. In the report "Cancer in Norway 2016", published in 2017, they reported a 20% increase in melanoma skin cancer in both men and women when comparing the data collected from 2012-2016 with data from 2007-2011 [18]. Although, if melanoma is detected in an early stage, when there are no signs that the cancer has spread beyond its starting point, the 5-year survival rate is as much as 98%. That is why detecting melanoma in an early stage is crucial for the patients chance of survival.

1.2.2 Diagnosis

To get a diagnosis for melanoma a clinical assessment is made to conclude whether the nevus is suspect, and if removal is warranted [19]. The assessment is often done using the ABCDE-criteria in combination with Dermatoscopy.

The ABCDE's

The ABCDE-criteria is a guide for finding common features in melanoma [20]. Before the introduction of the ABCD(E)-criteria in the 1980s (the E was not added until 2004), the assessment was made by looking at macroscopic features, such as ulceration and bleeding, which usually appears in the later stages of melanoma [21]. The use of ABCDE-criteria made it easier to detect melanoma earlier [21]. This was an important development for reducing the risk of the cancer spreading to other parts of the body and for reducing the death rate [22].

A	Asymmetry. One half of the lesion is unlike the other half.
B	Border. Border irregularity
C	Color. Varying colors.
D	Diameter. Melanomas usually have a diameter larger than 6mm
E	Evolution. The lesion is evolving or changing.

Dermatoscopy

Dermatoscopy is a diagnostic aid used to evaluate skin lesions and to investigate possible melanoma. It involves using a dermatoscope, which makes it possible to see skin structures that are hard to see with the naked eye [19]. A dermatoscope is a handheld microscope (usually with a 10x zoom) and a light source, being either polarized or unpolarized. The different polarizations are used to highlight different information in the skin structure. When using polarized light the dermatoscope does not have to be in contact with the patient, reducing the risk for the infection to spread. Some dermatoscopes can be connected to a camera for documentation.

The technique is not used to diagnose melanoma, but is rather one of many ways of determining whether melanoma is plausible, thus deciding whether or not further testing is needed. The next step usually is a histopathological examination, based on which diagnosis is made.

The added use of dermatoscopy, in comparison to using the naked eye, has increased the detection of melanoma from 60% to 80% [23]. Several studies have also shown an increase in accuracy after just one day of dermatoscopy training [24].

1.3 Machine Learning and Skin Cancer

The production of images using dermoscopy has opened a path for computer vision as a tool to classify an image to a specific type of skin lesion. There has been a lot of research on this topic, especially after deep learning made its huge impact on computer vision in the early 2010's. Since 2016, The International Skin Imaging Collaboration (ISIC) has organized a yearly competition for segmentation and classification on their dermoscopy image data set [17]. Today the classification performance of state-of-the-art machine learning models trained on dermoscopic images is at the level of dermatologists [3**?** , 7].

A main driver for the use of machine learning is the potential to improve efficiency in the work of a dermatologist, resulting in reduced costs, and to create diagnostic support systems that can be used in situations where expert dermatologists are in short supply. In general, the high number of patients with skin lesions are too much for certified dermatologists to handle. The increasing case-load of non-melanoma skin cancer has led to an increase in the responsibility of the primary care physicians (PCP) to implement the skin cancer screening [25]. However, many studies have shown that dermoscopy depends on experienced examiners to increase the diagnostic accuracy [23, 26, 27, 28]. In a study done on dermatologists (MDs) with 9,5 years of median experience and PCP's back in January and February 2010 and August to December 2012, the number needed to biopsy was found to be lower for the experienced dermatologists. When a PCP's needed 4,55 biopsied lesions to find a person with skin cancer, the MD's only needed 2.82 [25]. Another study from 2018 showed that dermatologists and general physicians agreed on the diagnosis in only 38,9% of the patients [29]. Since many studies have concluded that a state-of-the-art machine learning model performs at least at the same level as a dermatologist, an automated classification model have the potential to be a useful aid for physicians tasked to do skin lesion evaluations.

1.4 Research question and main contributions of thesis

Our main aim in this thesis is to use Generative Adversarial Networks to synthesise realistic-looking dermoscopic images of skin lesions. using the ISIC data set. Ideally, these generated images are of such a good quality that they can be used in the training of neural networks. More precisely, we investigate whether these *fake* images can be used alongside the original training data to increase the performance of classification models. This can be done by generating data to enlarge smaller data sets, or by generating more samples of some specific class to even out unbalanced data sets.

The hope is to use these enlarged data sets to train more robust and stable models, that are less prone to errors and are in general more reliable. Having a reliable model opens the door to create even better automated systems for skin cancer detection.

In this work we have explored different approaches to generate as realistic looking images as possible. This has led us through an array of different technologies within deep learning and generative adversarial networks. The thesis is focused on the investigation of the methods we have concluded to be the most promising for this task, and the technologies behind them.

The product of this work is a result of exploration in new approaches and inspiration we've gathered from other researcher's work. We end the introduction with a quick overview of related work inspiring our approach, and also related work in the broader field of generative adversarial networks and medical image synthesis.

1.5 Related work

Generative adversarial networks have many potential practical uses in medical imaging. The paper [30] provides a review of generative adversarial networks in medical imaging, and give a run-down of the main applications, some of which are:

- Reconstruction: Here GAN models where used to remove unwanted noise and artifacts from images. It is for example used to remove the noise in low-dose CT acquisition in [31] or to improve the quality and speed for MRI in [32, 33].
- Segmentation: Here GANs are used to improve segmentation scores by applying the discriminator as a sort of shape regulator [34].
- Classification: Here components from the generator and discriminator in a GAN model can be used for feature extraction, or one can use the discriminator as a classifier directly.
- Abnormality detection: GANs can be used to detect abnormality by utilizing the discriminator. During training the discriminator learns the probability distributions of the images. For example, if the images used during training is of normal pathology, an abnormal image will fall out of this distribution and be flagged as abnormal.

The applications above shows some of the different use cases for GANs in medical imaging. But for the purposes of this thesis the most relevant applications are those dealing with image synthesis, especially in the context of data augmentation. Traditionally, data augmentation is based on simple transformations of the images, e.g. scaling, rotations, etc [35]. But GAN models are also being used for these purposes. For example, in [36] the authors used a GAN to generate images of the eye fundus. The model was an image-to-image network trained on pair images of vessle trees and its corresponding eye fundus. They found that the synthesized eye fundus images were different from the real images in their global appearance, even though they where generated from the same vessle tree, while still retaining a high proportion of the true image quality.

The work reported in [37] also used image-to-image GANs for data augmentation. They used cycle-consistent generative adversarial networks (CycleGANs) in an unpaired image-to-image translation technique, transforming normal colonic mucosa images (the innermost layer of the colan), to synthetic colonic mucosa images containing an uncommon class of colorectal polyp (a abnormal tissue growth that can lead to colon cancer). The images that where generated was of such a good quality that two out of four gastrointestinal pathologists could not tell the synthesized images apart from the real ones. Additionally they found that the generated images were useful for data augmentation, leading to an improved classification model.

GANs can also be used to mitigate data imbalances, through generating additional images for the classes with low representation. In [38] different GAN models where used to generate realistic high quality images of melanoma lesions. The models where used in an experiment for skin lesion classification. The experiment where conducted by training a classifier on three different classes, it was shown that generated images helped by improving accuracy for cases with high class imbalance. As we shall see, our first experiment reported in Part II of this thesis resembles their approach, as we attempt to generate high quality images of skin lesions from random noise and testing the images as additional data for a classifier.

Before GANs where introduced by Ian Goodfellow in 2014 [39], there have been many other attempts at generating synthetic images. For example, in the 1990s, some work was done on texture synthesis of images, e.g. [40, 41, 42]. Here the goal was to capture the textures in an image, and generate a new synthetic image with the same texture. The work of Efros et. al. [40] starts with a seed (a small part of a sample image). Then they synthesize one pixel at the time around this seed by looking at the neighbours of the pixel, and creates a neighbourhood with width and height w, which contains pixels from the seed and empty pixels. With a similarity measure they find similar neighbourhoods in the sample image, and the center of a random chosen neighbourhood is the new pixel around the seed. This work motivated Rose et. al. to later synthesize images of mammograms [43].

In 2013, Jog et. al. [44] generated MRI images using a regression tree. They did an image-to-image translation by generating T_2 weighted MRI images from T_1. This was done to tackle the missing sequence problem when taking a MRI scan. This method was much faster than the state-of-the-art approaches at that time, and generated, in most cases, images of a higher quality.

In this thesis, we have investigated multiple different approaches to generate synthetic skin lesion images. As discussed earlier, recent years have shown that

GANs provide the most promising approach to image synthesis. We therefore focused our work on GAN-based approaches. In our first experiment, the goal was to generate as realistic looking skin lesion images as possible from random noise, much like in the work of [38]. To achieve this, we defined a novel pipeline, a pipeline that we have not seen anyone else using earlier. Unlike [38], our approach generates images of more than one class. In our second experiment, we wanted to balance a data set to improve the classification of a critical class in that data set. This approach was heavily inspired of the work in [37], but instead of generating synthetic colorectal polyp images from colonic mucosa, we generate images of melanoma from nevus.

Part I

Background

Chapter 2

Artificial Neural Networks

Artificial neural networks (ANNs) is a class of machine learning models, inspired by the biological brain. They have been around for a long time (arguably since the 1940s), but have seen a lot of traction over the last decade. It is the workhorse behind a lot of new technologies, and is behind all impressive results across all sort of fields, ranging from self driving cars, to targeted advertisement, and medicine.

These networks are examples of what is referred to as *self-learning algorithms*, being part of the larger field of machine learning (ML), aiming to create algorithms able to learn from data. In other words, ANNs are able to learn how to preform a task on their own, learning from examples of intended behaviour or from patterns extracted from data. In a style of training called Supervised learning, the network learns how to preform task by viewing examples of the task they are training on. I.e. if one wants to use a neural network to detect if a x-ray image contains a fracture, one would give the network x-ray images, some containing fractures, and some not. After seeing many of these images, the network can learn what images contains fractures and which does not. The network can then be shown a x-ray image it has never seen before, and make a prediction on whether or not the image contains a fracture.

This style of writing algorithms is different from the traditional approach to programming, where an algorithm is written as step by step instructions on how to behave. This can be very difficultly to do in many situations, confer the example of fractions in x-rays above. Humans can relatively easily be trained to tell the difference between a broken and healthy bone, but how would one describe that logic in a way that a computer would understand? It is for this types of situations the practical use of artificial neural networks become apparent.

In this chapter we try to give an introduction on how basic neural networks work, how they are structured and how they learn. Hopefully getting a basic understanding on how the basic works will make the more specified topics in the later chapters more digestible. The chapter is inspired by Michael Nielsen's book "*Neural Networks and Deep Learning*" [45]. Which is highly recommended if one wants to learn more about the subjects touched upon in this chapter.

2.1 Building a Neural Network

In order to understand how a neural network actually function, it is helpful to get an understanding of the different components it is made from.

2.1.1 The Artificial Neuron

An artificial neural network is a collection of connected artificial neurons.The idea behind artificial neurons goes back to the 40's when Warren McCulloch and Walter Pitts published their paper [46] about a mathematical representation of a simplified model of the biological neuron.

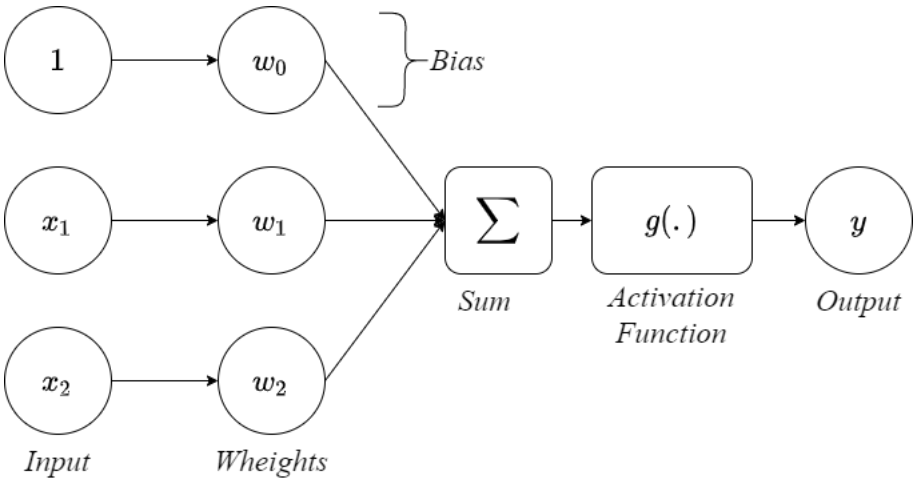

Figure 2.1: An illustration of an Artificial Neuron

The neurons used in artificial neural networks today consist of a few main components: an input, some weights and an activation function. The neuron receives some input ($[x_1, ..., x_i]$). This input is then multiplied with its matching weight ($[w_1, ..., w_i]$), a value contained inside the neuron. Then the sum is taken for all the input weight pairs, including a special pair (x_0, w_0) called the bias. The bias is an input/weight pair where the input is active (value of 1) at all times, meaning that its weight is always in use. This addition is used to push the value passed to the activation function in a positive or negative direction, independent of the input. This allows the weights connected to the actual input values to be more precise since they don't have to work as hard to bring the sum-value to a meaningful range for the activation function.

Beside this, is the bias regarded as all other input/weight pairs, and is used in the same fashion when calculating the sum.

The sum is then passed to a activation function. It is the activation function that defines what the final output from the neuron is going to look like.

2.1.2 Activation Functions

Step Function

The earliest implementation of the artificial neuron was the perceptron, created in the 60's by Frank Rosenblatt [47]. The perceptron described in the original paper is structured some what differently from the artificial neuron described above, but to keep things from getting to confusing we will describe similarly to the modern artificial neurons.

The perceptron is a neuron that receives multiple boolean-inputs and returns one boolean-output based on its internal logic. It receives some values $[x_1, ..., x_i]$, where each value is either 0 or 1. These inputs are then multiplied with the internal integer-value with the matching index ($[w_1, ..., w_i]$). The total sum is then calculated for all these values ($\sum_{i=0}^{n} x_i w_i$), including the bias, $x_0 w_0$, where x_0 is always equal to 1. As mention earlier the neuron is supposed to output a boolean value, but the sum of the weights and bias can be any integer value, large or small, negative or positive, it is not limited to 0 or 1. It becomes the job of the activation function to transform the sum to a boolean-value. The function used for this in the perceptron is the Step Function. This is a function that return 1 if it's given a positive value, and 0 if its input is negative. A problem with the step function is its sensitivity to small changes. Since the output from the step function always is either 0 or 1, small changes in its input can cause the output to completely flip from 0 to 1, and vice versa. This behavior is unwanted because it implies that a small adjustment made to increase performance in one area, can result in performance decrease in others.

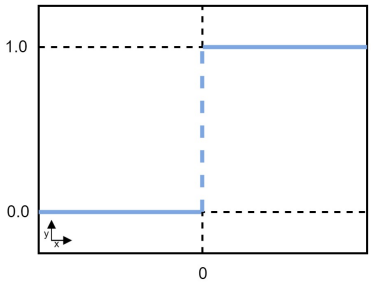

Figure 2.2: A graph illustrating the step function

Linear Activation Functions

One naive way of solving the problem related to sensitivity with the step function would be to use a linear activation function. Then the neurons output would be in some range rather then binary, meaning that changes to the input would not cause such drastic changes in the output.

Using linear activations to reduce the sensitivity sounds nice, but unfortunately this would drastically reduce the amount of problems that the network can solve. All a ANN actually does is to find patterns in the input data so that it can map them to specific outputs. When solving complex problems, this entails that the network has to find a complex pattern. If all activations in the network are linear, then the final function mapping inputs to outputs given by the network will also be linear. In other word, a ANN that uses linear activations can only map the input linearly, reducing the problems it can solve to those that are approximated well by a linear function.

Sigmoid

The Sigmoid Function, or logistic function, is a frequently used non-linear activation function. It works by squishing the neurons output into to a range between 0 and 1:

$$\sigma(x) = \frac{e^x}{1 + e^x}.$$

The characteristic "S" shape of the function is caused by the steep middle part of the curve. Having such a steep curve means that small changes in the input causes big changes in the output. Meaning that the function has a tendency to push all values towards 0 or 1.

A problem with the sigmoid function is what's at the edges of the graph. The bigger the input values get, the smaller difference it makes in the output. In other words, the gradient at these regions becomes small. This is problematic while training a neural network, since, as we shall see below, the changes (learning) to the network is based on the gradient. When the gradient becomes small, the changes in the network are also small resulting in a slow learning process. This problem is often referred to as the "vanishing gradients"-problem.

Tanh

Another "S"-shaped activation function is Tanh, the hyperbolic tangent function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

It looks and behaves similarly to the sigmoid function. It is in fact just a scaled sigmoid function, where the range is from -1 to 1 instead of from 0 to 1.

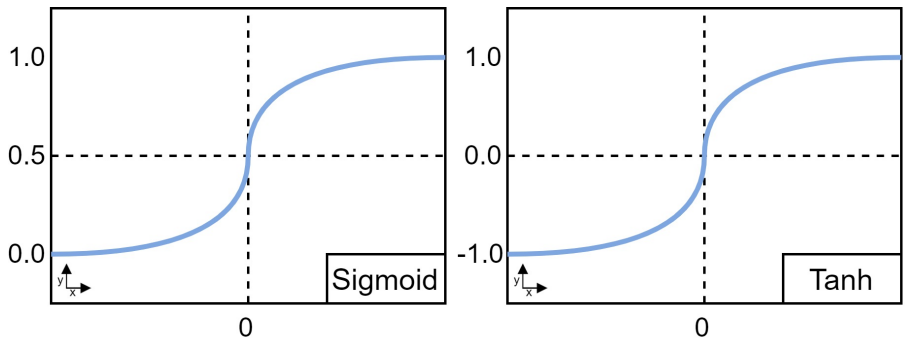

Figure 2.3: A graph illustrating the sigmoid function *(left)* and the tanh function *(right)*

ReLU

Another popular non-linear activation function is the ReLU, or rectified linear unit. The output for this function is the input as long as the input is positive,

if its negative the output is 0:

$$ReLu(x) = max(0, x).$$

It ranges from 0 to infinity.

Since the function excludes all negative values it results in networks that are relatively fast to train (see the section 2.2 for more details on training). If one uses Tanh or sigmoid, all inputs will give an output within some range, meaning that all the activations play a role in the final output. This makes the network dense and slow to train. But with ReLU negative values will cause neurons not to fire (i.e. output zero). The ReLU networks will be faster to train from scratch, since approximately half of the neuron will be inactive after initialization.

Neglecting negative values can also cause some problems, as the gradient will be zero for these neurons. This means that during training they wont be adjusted, becoming so-called dead neurons, permanently inactive. This is refereed to as the "dying ReLU problem".

Leaky ReLU

The leaky ReLU is a variation of ReLU that attempts to solve the problem of dead neurons. Here negative values are multiplied with some small number α (often 0.001):

$$LeakyReLU(x) = \begin{cases} x, & x > 0 \\ \alpha \cdot x, & x \leq 0 \end{cases}$$

This results in a slow decline below zero rather than an horizontal line, causing the values to be non-zero in other to help them recover during training.

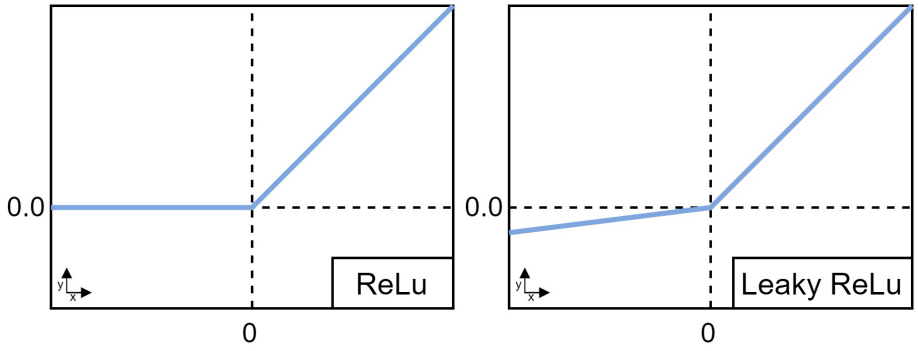

Figure 2.4: A graph illustrating the ReLU function *(left)* and the Leaky ReLU function *(right)*

2.1.3 Network Architecture

In the graphical representation 2.5 the information flows from left to right. Passing from one column of neurons to the next. These columns are called layers. The number of layers, and how they are connected will vary from network to network, but the layers can be broken down to three main types; input-, output- and hidden-layers.

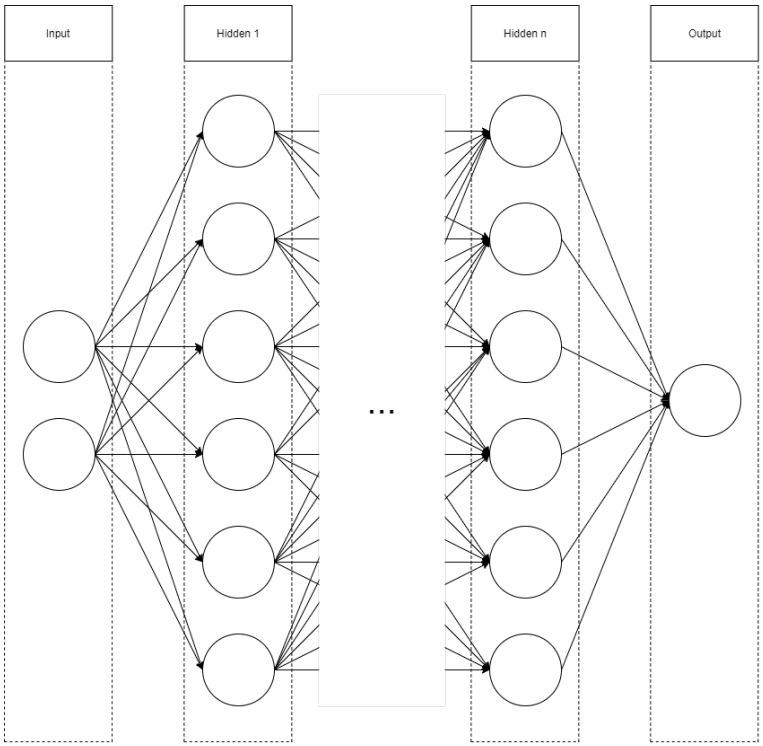

Figure 2.5: A graphical representation of a ANN, showing the different layers

The input layer is where the network gets its information. The nodes at this layer represents the values in one data sample. If the input to a network is an image, there would be an input node representing each pixel in that image. The nodes in this layer are connected to the hidden layer.

The hidden-layer is in the middle of the network and is the part that transforms the input to an output. The number of hidden layers denotes the networks dept. Having a large number of hidden-layers increases the complexity of the network in that it allows for more computations.

After the input has passed trough the hidden layers it reaches the output-layer. This is the last and final layer representing the networks output. Similarly to the input-layer the nodes here can take many forms, depending on what the task is.

2.2 Training a Neural Network

Now we have looked at how an input flows through a neural network producing a specific output, by transforming it in the hidden layers. This process of information flow from the start of the network to the end is called a forward-pass.

When a neural network is first created, its weights and biases are set to some random values by what is called initialization. When the input first passes through the network, the output will likely be wrong. The network isn't trained yet. That is because the weights and biases has not been adjusted for the data in order for it to return any meaningful output.

2.2.1 Cost Function

To give feedback to the network regarding its output a cost function is used. The cost function is also referred to as the loss function or the objective function, but they are the same thing which is, in essence, a function that measures the distance between two values.

In a typical neural network setting this would be how far off the predicted output $\hat{y}$ given x is from the actual answer y. This tells the network if its predictions is wrong and by how much. If the cost function returns a large number, i.e. the distance is far, the network is doing a bad job. The networks job is to reduce the distance between its predictions and the actual answer. In other words, to minimize the cost function.

2.2.2 The idea of Gradient Descent

To minimize the cost function gradient descent is often used. Gradient descent is an incremental optimization algorithm used to find the local minimum of a function. The idea is to figure out how to adjusts the functions variables in order to reduce the value of the functions output. The algorithm starts out at some arbitrary position. From that position it finds what direction it needs to move (what values to change) in order to move towards the (global or local) minimum of the function. The direction is determined by calculating the slope (or the gradient) of the function at its current position. Having that information it changes its position by stepping in the negative direction of the slope. The size of that step is determined by η the step size, later referred to as the learning rate. From the new position the process is repeated, incrementally moving towards a minimum.

$$Position \rightarrow Position' = Position - \eta * Slope$$

This process keeps iterating until the slope equals 0 (or is approximately equal to 0), or until some max number of iterations have been completed.

Univariate Gradient Descent

An example of gradient descent applied on a univariative function to illustrate the concept.

The function we want to minimize is a univariative function $f(x)$, the goal is then to minimize the output from $f(x)$ by making small incremental changes to x.

First we find a starting position by selecting a value for x. Then we calculate the slope at that position by taking the derivative of the function at x, $\Delta f(x)$. Using that slope we update at the current position x to a new position x', by taking a η size step in the negative direction of the slope:

$$x \to x' = x - \eta \Delta f(x)$$

Then repeat that process from the new point, then the next, etc, until we have reached our conditions for determination.

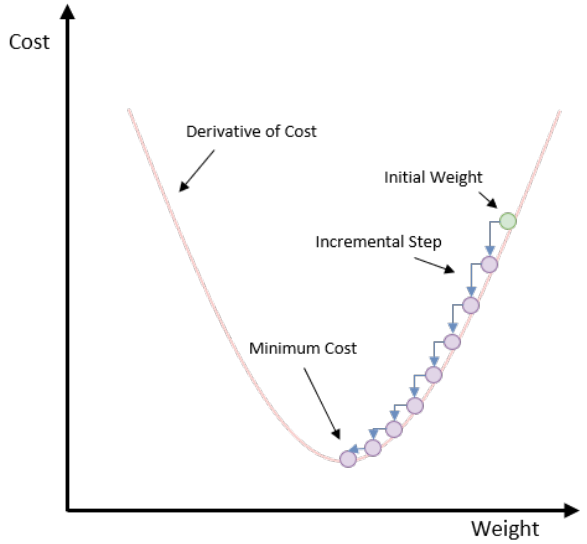

Figure 2.6: An illustration of gradient descent on a univariative function

Multivariate Gradient Descent

A neural network is a function with a large amount of variables (weights and biases), so it can be useful to give an example of how gradient descent work on a function containing multiple variables.

C is a function for two values v_1 and v_2, and what we want is to minimize that function $C(v)$. (If this was the cost function for a neural network it would contain many more variables, but the approach is the same for two variables which is much easier to illustrate.)

Same as before we use gradient descent to make negative changes in C by making small adjustments to its variables v_1 and v_2.

The difference is that earlier when the function only contained one variable, we could calculate the slope by taking the derivative of the function, but now we are working with multiple variables.

In order to find the gradient (slope) for C we set it to be the vector of partial derivatives, the gradient vector ∇C:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)^T .$$

We also define Δv as the vector of changes in v, $\Delta v \equiv (\Delta v_1, \Delta v_2)^T$. The iterative adjustments to the variables v_1 and v_2 is then:

$$\Delta v = -\eta \nabla C,$$

Giving us a similar update rule as earlier: $v \to v' = v - \eta \nabla C$.

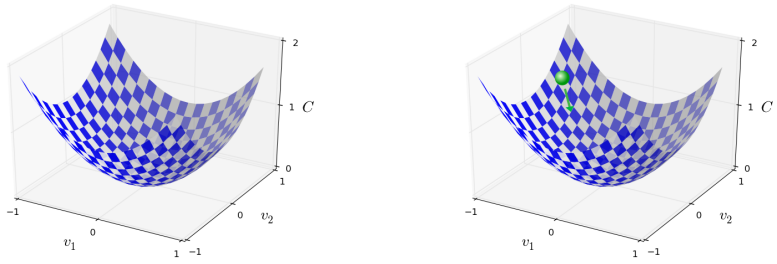

Figure 2.7: An illustration of gradient descent on a function with two variables. This figure is from `http://neuralnetworksanddeeplearning.com/chap1.html` which is licensed under a Creative Commons Attribution-NonCommercial 3.0 Unported License.

2.2.3 Backpropagation

In order to minimize the cost we use gradient decent. And in order to do that we need to calculate the partial derivatives for all the weights and biases in the network with respect to the cost. Backpropagation is an algorithm that allows us to do just that.

The cost function is a function of the neural networks output a^L:

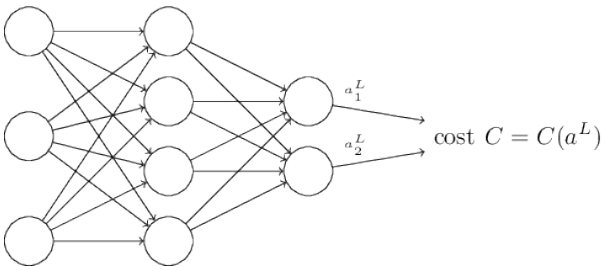

Figure 2.8: An illustration of how the cost function is a function of the networks output. This figure is from `http://neuralnetworksanddeeplearning.com/chap2.html` which is licensed under a Creative Commons Attribution-NonCommercial 3.0 Unported License.

a^L is just a function of all the weights, biases and activations being past from the previous layer:

$$a^L = \sigma(w^L a^{L-1} + b^L)$$

The same goes for that layer:

$$a^{L-1} = \sigma(w^{L-1} a^{L-2} + b^{L-1}),$$

and the next, continues all the way back to the first layers in the network.

In order to compute the partial derivatives for all the weights and biases in the network it has to start at the end of the network, and then compute its way back to the input-layer.

So in order to compute the partial derivatives for all the weights and biases in the network it has to start by computing the partial derivatives of the cost function with respect to the weights and biases, and the activation function from the previous layer. This computation results in an intermediate quantity called the error δ^l, where l denotes the layer. The error in one layer δ^l is then used to compute the error in the previous layer δ^{l-1}, and so on, backwards trough the network. The errors are then used to compute the partial derivatives for each individual weight, $\frac{\partial C}{\partial w^l_{jk}}$ and bias, $\frac{\partial C}{\partial b^l_j}$. The gradient of the cost function is described by combining those partial derivatives, allowing the weights and biases to be updated with gradient descent.

Equations

The equations used to compute backprobagation as described by Michael Nielsen [45], see book for more detailed deception and proof for the equations.

- An equation for the error in the output layer, δ^L:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

- An equation for the error δ^l in terms of the error in the next layer, $l+1$:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

- An equation for the rate of change of the cost with respect to any bias in the network:

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

- An equation for the rate of change of the cost with respect to any weight in the network:

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

2.2.4 Learning Rates

As mentioned above the learning rate (step size) determines how much the parameters are adjusted in the negative direction of the gradient. The learning rate is a hyper-parameter, one chooses its value before running the algorithm. Setting the right value for the learning rate can be crucial for model performance. Choosing a learning rate that is too high can lead to overshooting the minima, and in some cases lead to divergent behavior; where it moves in the positive direction of the gradient.

Using a learning rate that is to small can lead to slow learning. Meaning that it takes a insufficient amounts of steps to reach minima, because the the changes are so small per iteration. The main idea is to find a optimal learning rate, that doses not over shoot the minima and that isn't to slow.

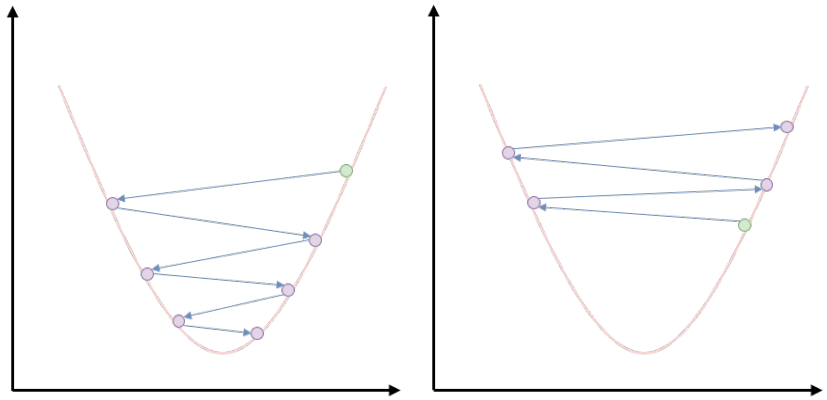

Figure 2.9: An illustration of how the learning rate impacts gradient descent. *Left:* shows when the learning rate is to high. *Right:* shows a learning rate that is way too high, causing it to diverge

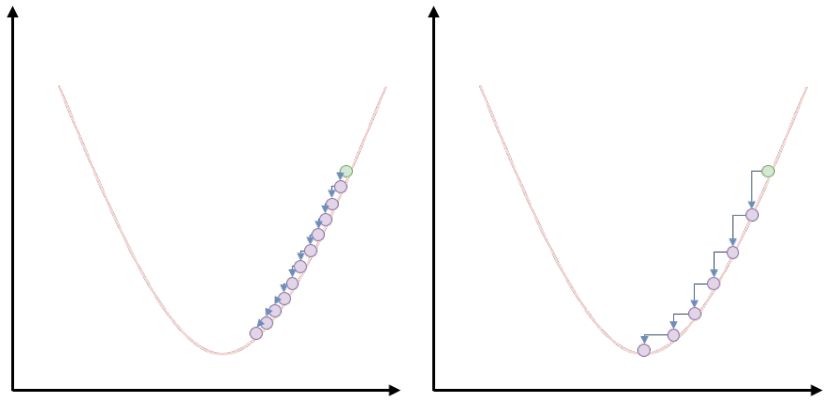

Figure 2.10: An illustration of how the learning rate impacts gradient descent. *Left:* illustrates how a low learning rate can result in slow learning. *Right:* illustrates how a "optimal" learning rate allows for a smooth descent with few steps.

2.2.5 Feature Scaling

Another way of making gradient descent faster is by scale the features (data). Having un-scaled data is slow since this means that some parameters has to be adjusted by a larger amount then others. When the data is normalized the parameters can be adjusted by a some what similar amount, making the process of gradient descent smoother and faster.

I.e. if we have some cost function $F(x_1, x_2)$. And the features are not scaled the gradient descent algorithm has to adjust x_1 and x_2 by varying amounts, leading to a lot of jumping back and forth, taking a long time to reach minima. While if the features are scaled the parameters can be adjusted by a similar amount, giving a smother descent that requires fewer iterations.

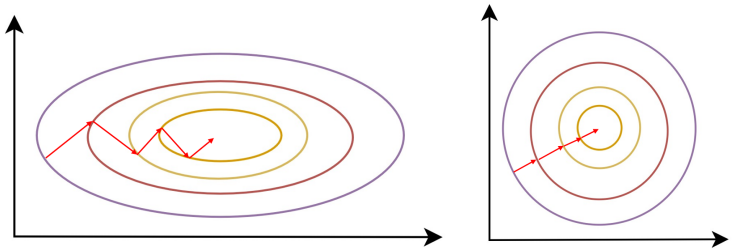

Figure 2.11: Contour plot of $F(.)$ showing the effect feature scaling has on gradient descent. With unscaled data on the *left*, and scaled data on the *right*.

Standardization

Standardization (also called Z-score normalization) scales the features to a standard normal distribution with the mean μ of 0 and a standard deviation σ of 1.

$$x' = \frac{x - \mu}{\sigma}$$

Normalization

Normalization (often refereed to as Min-max scaling) scales the features in to a fixed range, $X_{min} < - > X_{max}$. The range is often 0 to 1, or 0 to 255 on images.

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

$$x' = \frac{x - X_{min}}{X_{max} - X_{min}}$$

2.2.6 Overfitting and Underfitting

A common problem, and often the reason for bad performance of a machine learning model, is overfitting or underfitting. If a model is very complex it has the tendency to fit the training data extremely well, but produce bad predictions on unseen data. This is called *overfitting*, characterized by high generalization error. If the model is too simple it outputs bad predictions on both training data and unseen data. This is called *underfitting*. We want to find the golden middle way, i.e. a model that has low training error and low generalization error, i.e. that predicts well on unseen data.

In practice, when using models of high capacity, i.e. complex models that can reach very low training errors, overfitting is often caused by overtraining: during training the model becomes too adapted to the training data, including its noise. This results in a minimal training error, but the model will make unreliable predictions of new instances. On the other hand, if the model hasn't learned enough from the training data, it will typically also be unreliable when producing predictions on new input.

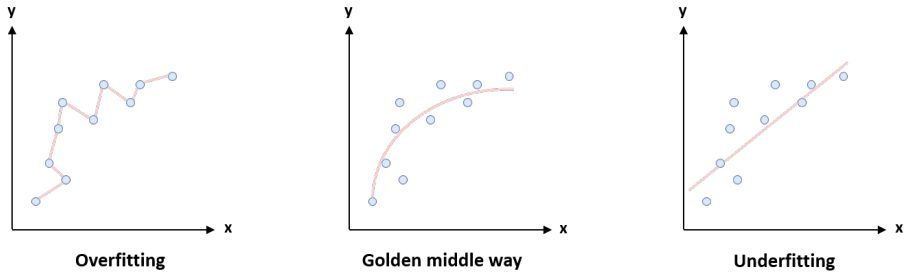

Figure 2.12: Visualizing overfitting and underfitting.

2.2.7 Bias and variance

There are different sources of generalization errors: irreducible, bias and variance.

The irreducible error have nothing to do with the trained model, but inherent features of the data, for example noisy or uncertain measurements from sensors. The only way to reduce this error is by cleaning the data set.

The bias is high when the training error is high. When this is the case it is often a simple model that is underfitting.

The variety in the performance when constructing models from different training data is called the model variance. A complex model that is overfitting tends to have high variance. It adapts too closely to very specific features of the training data and not to the underlying structures common to all possible sets of training instances.

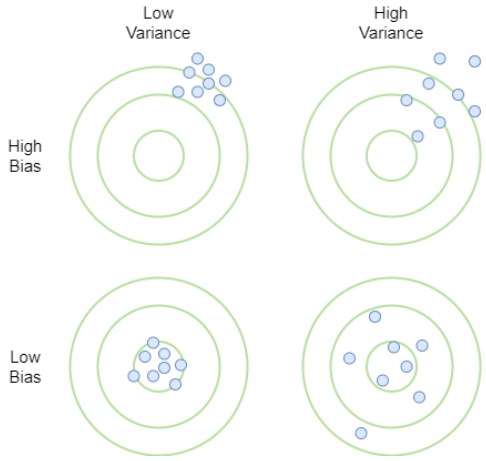

Figure 2.13: Visualizing the models that are high or low in bias or variance.

Bias-variance tradeoff

To find the sweet spot for a model that generalizes well, you need to discover the optimal balance in the bias-variance tradeoff. The optimal solution is to have both low bias and variance, but these two will fight against each other. If

the bias is decreased the variance will increase, and vice versa. This is because when a model is simple and underfitting the training error is high, which means the bias is high, and the variance is low. By training the model further, it becomes more complex, which reduces the training error and bias, and increase the variance. When the model is overfitting the variance is high, and the training error and bias is low. The challenge is to find the correct balance such that the model neither overfits nor underfits. See figure 2.14

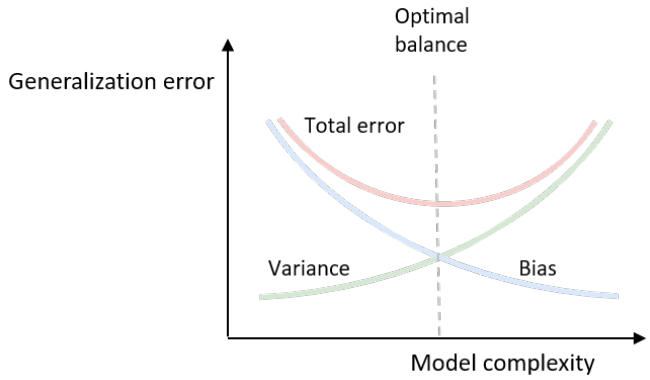

Figure 2.14: Illustration of the bias-variance tradeoff showing the optimal balance. With a simple model the bias gives a high error and the variance a low error, the more complex the model gets the error because of the bias is reduced, but increased because of the variance.

2.2.8 Optimizing Gradient Descent

There are multiple approaches to gradient descent, differing in their usage of training data when computing gradients. Which one to choose depends on the amount and character of the training data, and it becomes a trade-off between the accuracy of the weight updates, memory usage and training time. In this section, we follow the notation in [48]. The goal of gradient descent is to minimize the objective function $J(\theta)$, where θ are the weights. We write the learning rate as η.

Stochastic Gradient Descent

In Stochastic Gradient Descent or SGD, $J(\theta)$ is calculated and gradient descent is applied for each data-sample. It can be expressed as follows:

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

where $x^{(i)}$ and $y^{(i)}$ is a specific training sample. This approach can be time consuming for large collections of data, since the calculations have to be done often. It can also result in very unstable gradient updates. It is rarely used in practice.

Batch Gradient Descent

In Batch Gradient Descent, the gradient is calculated based on the entire training data set:

$$\theta = \theta - \eta * \nabla_\theta J(\theta)$$

In this case, the gradient will definitely point in the direction of steepest descent of $J(\theta)$ (given the training data), and convergence can therefore be faster. But depending on the size of the training data set, BGD can have prohibitively large memory requirements. Batch gradient descent is often completely infeasible to use in practice.

Mini-Batch Gradient Descent

Mini-Batch Gradient Descent (Mini-BGD) provides a middle-ground between SGD and BGD by introducing a parameter called the batch size. The training data is separated into multiple batches, each of the same size (mini-batches). The errors and updates is then calculated and done for each mini-batch:

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

Batch size equal to 1 recovers SGD, batch size set to the size of the training set is identical to BGD. The adjustable batch size provides a way to strike a balance among gradient stability, speed and memory requirements. It is also highly parallelizable and can therefore make efficient use of the high number of cores in GPUs.

Additional tweaks to gradient descent

These were the basic forms of gradient descent. There are many further variations that result in optimization algorithms that can perform significantly better in practice. We mention two such tweaks below. See [48] for a nice overview of gradient descent optimization algorithms.

Momentum. With the above gradient descent approaches, the weights are updated usign the gradient at the current step. With momentum, the weighted moving average of the previous steps are included in the equation. This results in movements analogous to rolling a ball down a hill, making it possible to pass over small valleys (i.e. local minima). This is done by adding a term to the gradient update, controlled by a hyperparameter γ between 0 and 1 (most cases as 0.9) . γ is multiplied by the result from the previous step and added to the current step length:

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

Adam optimizer. Adaptive Moment Estimation (Adam) is one of the most frequently used optimization algorithms. It is a combination of momentum and what is called Root Mean Square Propagation, or RMSProp. RMSProp divide the learning rate by the exponentially decaying average of squared gradients (v_t), to adjust the learning rate automatically for each update of a weight. The hyperparameter β_2 is in most cases between 0.9 and 0.99.

The weighted moving average is included with the formula v_t, like momentum:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

With m_t and v_t, the weight can be updated by using the Adam optimization algorithm:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

ϵ is just a small number to make sure we do not divide by 0.

2.2.9 Regularization

Regularization is a collective term for tecniques used to reduce generalization error. In practice, when a model is underfitting the solution often is just to continue training the model.[1] On the other hand there are many techniques to prevent overfitting. In this section we present some of the most common regularisation techniques for deep learning.

Data augmentation

Collecting more data is the best way to prevent overfitting, but in many cases this is extremely difficult or prohibitively expensive. An approach for expanding an existing data set, without gathering more data, is data augmentation. In computer vision this is an easy and efficient way to, in effect, obtain more training data. Operations such as flipping, rotating, zooming and changing contrast can be applied to the images in the data set, creating several versions of the same images. See figure 2.15 for an example.

Note that one can also use data augmentation to make models more invariant to the transformations used to expand the data set, e.g. better able to detect upside-down cats.

Early stopping

When training a neural network you can often get better generalization performance by stopping the training process when the model starts overfitting. Signs of overfitting can be detected by looking at the training and validation

[1] This is true if the underfitting is not caused by using a model that's too simple, i.e. of too low capacity. In that case, switching to a more complex model or class of models may be necessary.

Figure 2.15: Data augmentation applied to an image of a cat from [49]. For a computer vision model this can, in practice, have an effect similar to expanding the data set with completely new images.

loss during training. When the validation loss is significantly higher than the training loss, the model will likely be overfitted when evaluated on the test data. By stopping the training before this happens, or reverting to an earlier, stored model checkpoint, one can end up with a model that generalize well.

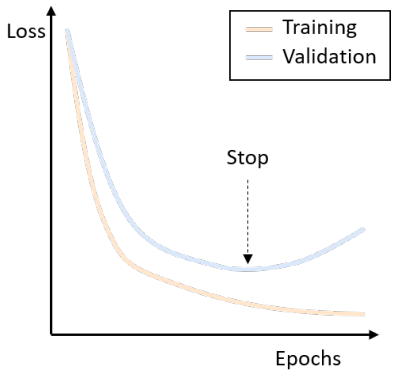

Figure 2.16: Shows the learning curves of training a neural network. Early stopping is when you stop training while the validation loss is at its lowest before it start increasing.

Dropout

Another approach to avoid overfitting artificial neural networks is dropout, introduced by Srivastava et. al. in 2012 [50]. When applying dropout, each unit (neuron) and its connected weights is removed with a probability p (a typical choice is $p = 0.5$), independently for each neuron and for each training iteration, resulting in a sparser version of the neural network (See figure 2.17). At test

time, the full neural network is used.[2]. This tends to lead to better generalization performance as there's a lower chance of each individual neuron being assigned specific, important tasks during training.

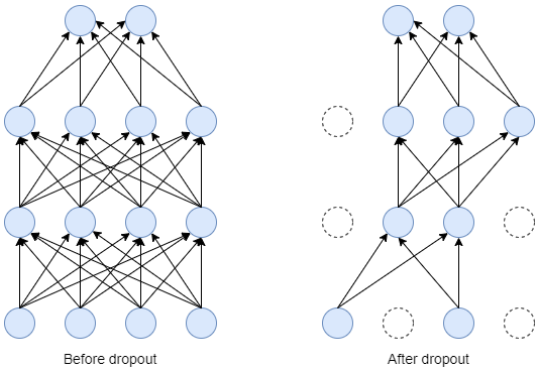

Figure 2.17: An example of how a neural network looks before and after dropout. For each batch of training data flowing through the network neurons are disabled with a certain probability, resulting in network's that tend to overfit less.

L1 and L2 regularization

A sign of a complex model is if the neural network contains large weights, signifying high reliance on certain neurons. As discussed above, this can lead to overfitting. To help avoid one can add a penalty to the loss function during optimization, e.g. an L1 or L2 regularisation term. L1 regularization penalize the weights by pushing them to become zero:

$$\text{Loss} = \text{Error}(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

The L2 regularization focus more on penalizing the large weights, and does not force their values to become zero:

$$\text{Loss} = \text{Error}(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

In both cases, λ is a hyperparameter with value between 0 and 1, controlling the amount of weight magnitude penalty. The higher the number, the harder the weights are penalized.

2.2.10 1cycle policy and cyclic momentum

In the paper [53], Leslie Smith present different approaches for tuning your hyperparameters. To optimize the learning rate a test is run where it starts

[2]Leaving dropout turned on also at test time and feeding the test data through the models multiple times leads to an interesting approach to getting an estimate of model uncertainty associated to model predictions. This was introduced in [51] and further explored in the MSc thesis of Sean Murray from our research group [52]

with a small learning rate which is increased for each mini-batch until the loss starts increasing drastically. With this test it is possible to find a good learning rate. Then during training the 1cycle policy is applied. The chosen learning rate is set as the maximum learning rate, and the minimum learning rate is set to $\frac{1}{5}$ or $\frac{1}{10}$ of the maximum learning rate. While training there is two steps with equal length in the 1cycle policy. The first step starts with the minimum learning rate and increases to its maximum, step two starts with the maximum learning rate and decrease until it reach its minimum. The cycle length is set to be a little less then the total number of epochs the training of the network is running. The remaining epochs, the learning rate is even lower than the minimum learning rate.

The cycle momentum is an approach to set the hyperparameter momentum during training. The approach is to adjust the momentum in the exact opposite direction of the learning rate, so when the learning rate is increasing the momentum is decreasing and the other way around. See Fig. 2.18

Combining these two approaches have shown to greatly decrease the training time.

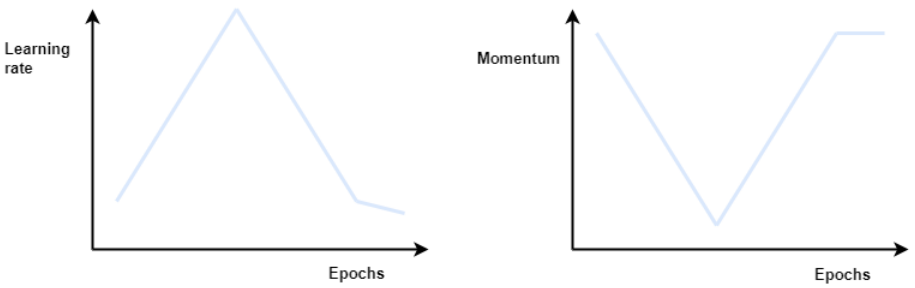

Figure 2.18: First graph shows the 1cycle policy, the second shows the cycle momentum.

Further unpublished work have shown even more improved results, if you use the same step one, but in the second step cosine annealing from the maximum learning rate to zero. The momentum also follows step one from earlier, and follows a symmetric cosine in the second step. This is shown in figure 2.19

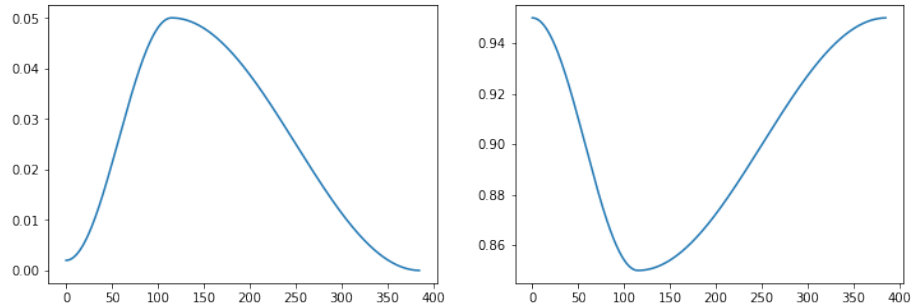

Figure 2.19: Shows the improved 1cycle policy, and cyclic momentum. Fig. 8 from [54]

2.3 Evaluate a Neural Network

It is important to point out that this section is about metrics that can be used for evaluating all kinds of machine learning models used for classification tasks. But in this work the metrics have been used to evaluate neural network specifically.

When evaluating a model there is many different metrics that can be used to discover if the model does a good job or not. But depending on what data set that is used, it is important to know which metrics to use to assess the quality of the model. If the data set is highly unbalanced containing 90% of data with the same label, the accuracy score will be 90% if the model always predict that label, which makes the metric accuracy misleading in evaluating the model. In this section we will explain some of these metrics. But first, to understand the concepts of true positive (TP), true negative (TN), false positive (FP) and false negative (FN) look at the illustration in figure 2.20

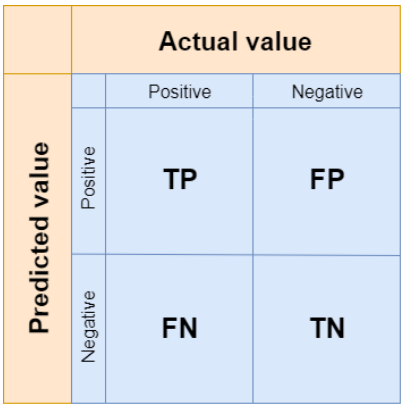

Figure 2.20: Shows the concepts of true positive, true negative, false positive and false negative

2.3.1 Accuracy

Accuracy evaluates how accurate your model is, by looking at how many inputs have been predicted correctly, and works great when dealing with a balanced data set.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

2.3.2 Recall

Recall evaluates how precise the model is on predicting positive labeled data. This metric is good to use with a unbalanced data set and when it is important to avoid the false negative cases.

$$Recall = \frac{TP}{TP + FN}$$

2.3.3 Precision

Precision looks at the positive predicted data and evaluates how precise the model is in positive predictions. This is a good measure when the data set is unbalanced and the importance of discovering false positive cases are very high.

$$Precision = \frac{TP}{TP + FP}$$

2.3.4 Model Evaluation in Real World

While these measurements mentioned above are useful for evaluating the quality of you model they should be viewed as indications and not be interpreted to strictly. For a model that preforms well on the data its trained and evaluated on, can in many cases have problems when applied in new situations, where features can vary from in the lab setting.

Google felt the impact on this when they applied a deep learning system for the detection of diabetic retinopathy (DR), for real life use in Thailand [55]. There model was built with the intention to reduce the amount of time it takes to process screening of DR, which can take 2 to 10 weeks, but with the use of there algorithm they meant it could be reduced down to 10 minutes. The algorithm had shown promising result in detecting referable cases of DR, scoring over 90% on sensitivity and specificity. To achieve these results the model had a feature where it discarded images if the quality wasn't good enough for it to make a accurate prediction. This feature cause problems in the real life usage, where poor lighting conditions resulted in that a large portion of the images where deemed ungradable by the model. This meant that multiple images had to be taken in order for the patient to be processed. This caused some patients to spend more time getting processed that it normally would using the normal approach.

Chapter 3

Convolutional Neural Networks

Neural networks really started to outperform other methods within the field of computer vision when Convolutional Neural Networks (CNN) were used in the winning entry of the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), significantly outperforming all competitors using their AlexNet CNN model [56]. CNN is not a new technology, it was already introduced by Fukushima [57] as early as 1980, and used together with backpropagation in 1989 [58]. But it was not until the recent breakthrough performances that CNNs became the go-to method for most of computer vision tasks.

The CNN takes a matrix of pixels as input, instead of a vector of pixels, like in a regular neural network. This matrix is sent through a series of filters (layers), and, for image classification, in the end outputs one single label for the input image. A CNN network do a better job processing more complex images, and have improved results in a lot of image classification problems [56, 59]. It is also used for other computer vision tasks beyond classification, e.g. semantic segmentation and various other image-to-image settings, and to solve non-computer vision problems [60, 61, 62].

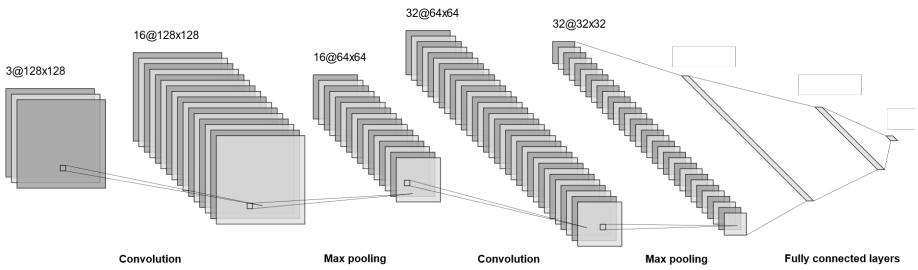

Figure 3.1: An example of a CNN architecture, showing some of the fundamental building blocks for constructing a CNN, including convolutional layer, pooling layer and fully connected layer. Made with [63]

3.1 Layers

In this section, we will go further into CNN architectures by looking at some of the different layers commonly used when constructing a CNN. Note that modern CNNs combine these and other building blocks in (increasingly) complex ways. This is exemplified in Section 3.2 below.

3.1.1 Convolutional layer

The convolutional layer is the main component in a CNN. A fully-connected, non-convolutional neural network for image processing takes an image as input, treating every single pixel from all its color channels individually. This is a lot of information, and probably a lot of useless or at least redundant information. The core idea of a convolutional layer is to focus on the important parts, by extracting the features in the image. This is done by applying a mathematical operation with a set of matrices called kernels, over the image. These kernels makes it less redundant by: Having kernels that are smaller then the input (*sparse connectivity* [64]), reduce the amount of parameters needed for matrix multiplication. And reusing the same weights in multiple functions (*weight sharing* [64]), reducing the workload for updating weights. In the first few layers the CNN can focus on features like corners and horizontal lines, deeper into the network more complex shapes are detected.

A convolution takes a small part of the image in the first layer, or the output of the previous layer and transforms the pixels within this region into one single pixel, with the assistance of a kernel. The kernel is a matrix with the same number of dimensions as the input to that layer, but with a smaller width and height (the depth is the same). The kernel is applied to this smaller region and does an element-wise multiplication and add them together into a single number. This operation continues one step to the side, when the stride value is 1, and slides over all the input values. Stride is a way to control the output width and height, by defining the number of steps the kernel is going to jump after each convolution. With a stride of 2 the output width and height will be half of its input. With this approach there will be a reduction in the output dimensions, even with the stride as 1, because the kernel will not be able to preserve the outer pixels. A normal solution to this is to add a padding around the image with zeros. This is explained with an illustration in Figure 3.2.

3.1.2 Pooling Layer

Another approach for down scaling the width and height instead of increasing the convolution stride is pooling. The two most common functions is max pooling and average pooling. To get half the width and height from the input you use a 2x2 frame and slide over the input with a stride of 2. For max pooling the output is the highest value in the frame, see Figure 3.3 for average pooling you take the average of all the values in the frame.

This is done to compress the input and reduce the number of parameters, and opens the opportunity for creating more depth in the output in the next convolutional layer.

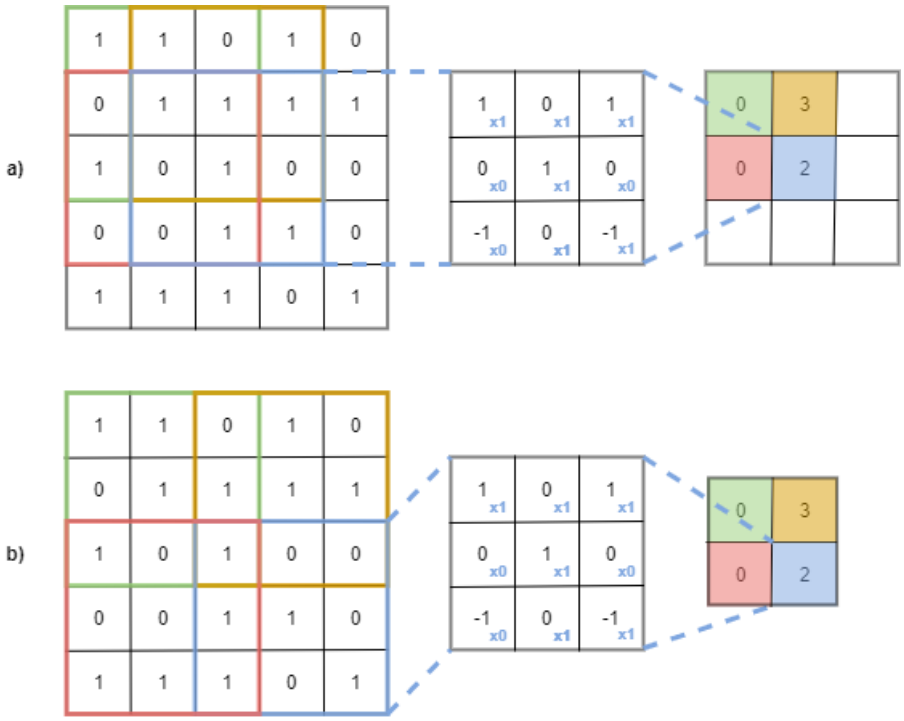

Figure 3.2: a) A convolutional operation with kernel size 3x3x1, and stride 1. b) A convolutional operation with kernel size 3x3x1, and stride 2.

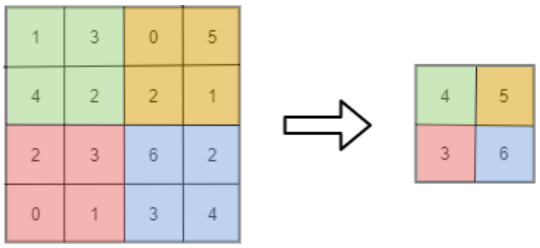

Figure 3.3: Max pooling with a 2x2 frame

3.1.3 Fully Connected Layer

In the last step of the network, the matrix is flatten into a vector where each neuron is connected to each neuron in the next layer, like in a normal feed forward network. This is to map from the deep feature maps into the desired output.

3.2 Three illustrative examples of CNN architectures

3.2.1 AlexNet

CNN got its breakthrough when Krizhevsky et. al. participated in the ImageNet competition with their AlexNet back in 2012 [56]. They crushed their opponents with an error rate of 15.3% compared to 26.2% from the nearest competitor. This was done with a pretty standard CNN, containing five convolutional layers, each followed by ReLU activations, three max pooling layers, and ends up with three fully connected layers.

3.2.2 ResNet

There has been a lot of different improvements from the basic idea of AlexNet. One of the most impactful is ResNet, which was published by Kaiming He et. al. from Microsoft Research in 2015 [59], and minor tweaks of ResNets are still today state-of-the-art approaches for deep learning-based image processing.

The origin of ResNets is based on an observation of Kaiming He [59]. He compared the training error from a 20-layer and 54-layer CNN, expecting the 54-layer network to be at least as good as the 20-layer, and that it would likely overfit. That did not happen. The 54-layer network had a higher training error than the 20-layer network. He then got the idea to introduce *residual blocks* to the network. For every second convolutional layer the input x of the first layer was added to the output of the second layer with an identity connection. This is often called a skip connection, see Figure 3.4.

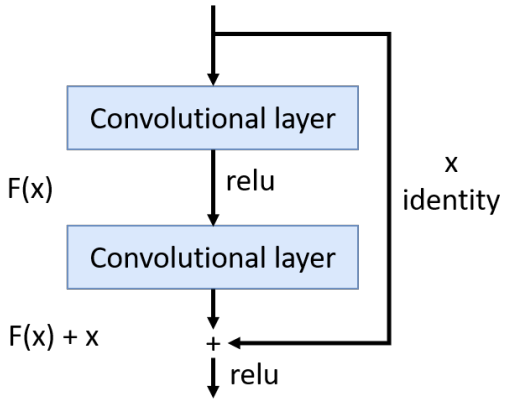

Figure 3.4: A Residual Block inspired by Fig. 2 in the paper *Deep Residual Learning for Image Recognition* [59]. Note that adding paths that bypass weight layers allows the network to learn to, in effect, use fewer layers for certain inputs.

The idea of this is that if in worst case there is no need for the extra layers, they can learn to output the identity, and only the parameters from earlier layers in the network is retained. This should make a deep network at least as good as a shallower one. This was a great success and they easily won the ImageNet competition that year, with a 152-layer network, which was a lot deeper than

any of its opponents. ResNet made it possible to create much deeper networks, but why did it work? The answer to this came three years later when Hao Li et. al. discovered how to visualize the loss surface of a neural network [65]. We were then able to see that with skip connections the loss landscape was much smoother, which makes it easier to find the global minimum. Without skip connections a deep network created such a complex loss landscape that it would get stuck in a local minimum. See Figure 3.5

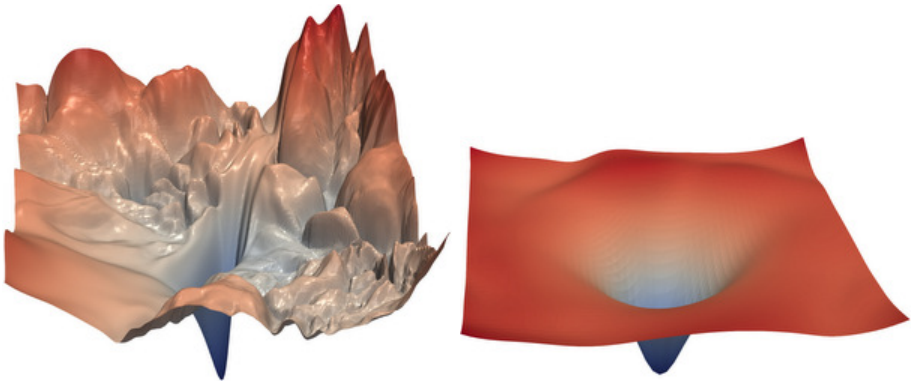

Figure 3.5: The loss surface of a 56-layer ResNet, to the left without skip connections and to the right with skip connections. This figure is published under the MIT License, for more information see: *https://github.com/tomgoldstein/loss-landscape/blob/master/LICENSE*

3.2.3 Unet and image segmenation

Until now we have talked about image classification where the whole image have been predicted to a specific class label. Another task, which is especially interesting within the field of medicine is to locate different objects in an image with segmentation. This is done by classifying each pixel into what kind of object in the image it is a part of. Instead of outputting one label class, it outputs a label for each pixel in the input image, shown in Figure 3.6

The U-net architecture (Figure 3.7) is still a state-of-the-art approach for image segmentation, and was published in 2015 [67]. A U-net can be split into two parts, the down-sampling part called the encoder, and a up-sampling part called the decoder. The encoder is typically CNN without the fully connected layers. Since the output is suppose to be the same size as the input we need an up-sampling part which is the decoder. There is many approaches to increase the image size e.g. transposed convolution (deconvolution), nearest neighbor interpolation, bi-linear interpolation and bi-cubic interpolation. The most common technique for a U-net is the transposed convolution with a 2x2 kernel, which transform input pixel with all its depth into 4 pixels (2x2), and for one up-convolution several kernels is used to output half the depth of the input. For further details see [68]. But from this approach a lot of details from the input images is lost in the encoder part, and it is difficult to rebuild the image with the small amount of information after the encoding. That is why skip connection is introduced. After each up-sampling layer, the output from the same level in the

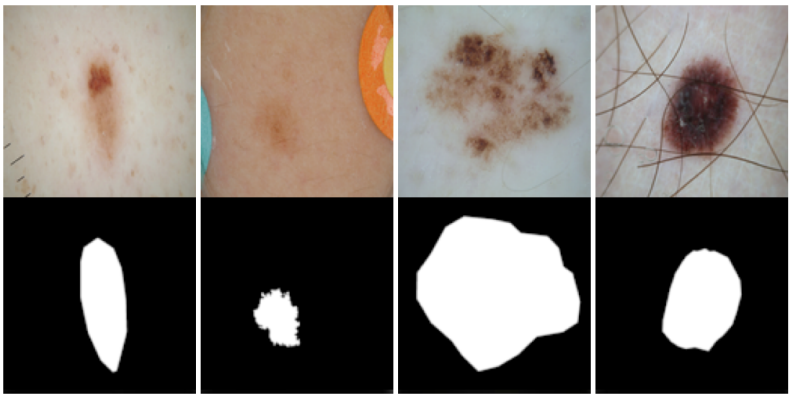

Figure 3.6: Examples of training data from the ISIC 2018 competition. Top: Input data. Bottom: How output data is suppose to look like [66]. The ISIC data set is also used for Part II in this thesis

encoder is concatenated to the up-sampling output. And this is how the details from the input image is kept throughout the network.

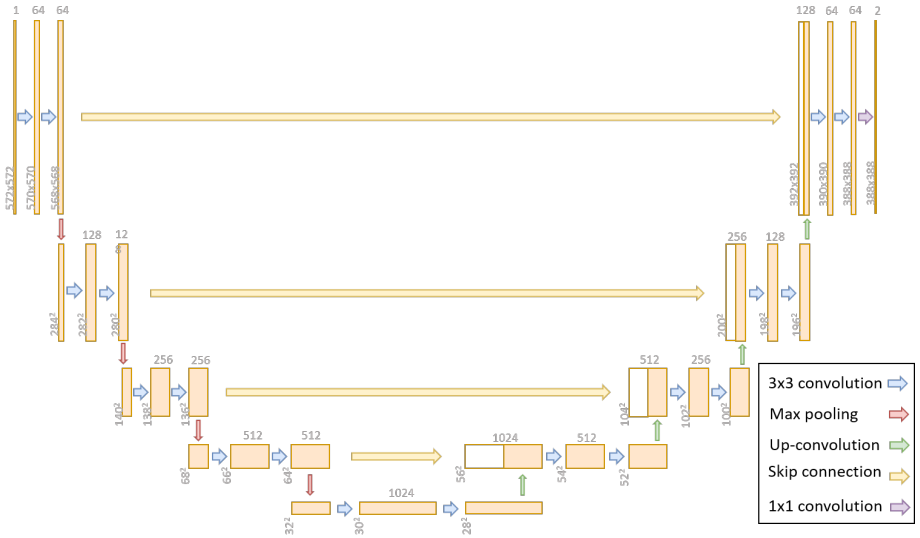

Figure 3.7: The U-Net architecture from the paper *U-Net: Convolutional Networks for Biomedical Image Segmentation* [67]

Chapter 4

Generative Adversarial Networks

The last couple of years generative models have shown great potential in generating realistic looking synthetic images, e.g. BigGAN [11], StyleGAN2 [12] and pix2pix [69]. After Ian Goodfellow came up with the idea of generative adversarial networks (GAN) [39], this have become the standard approach for generating synthetic images. There have been a lot of development within the field of GANs, and many different architecture have been explored, but with the same fundamental ideas that Goodfellow introduced back in 2014. In this chapter we are going to look deeper into the GAN models that we have found most suitable for our project, where we aim to produce as realistic looking synthetic images as possible.

4.1 Generative adversarial networks

The traditional GAN model of [39] includes two deep neural networks, whose combined purpose is to synthesize new "fake" data. The two deep neural networks in the GAN are called the generator(G) and the Discriminator(D). The generator is given vector z, often consisting of random noise, and attempts to produce "fake" data G(z) by sampling from the data distribution $p_{\text{data}}(x)$ of the training data, attempting to fool the discriminator D, whose aim is to distinguish real samples y from those produced by G. During training, the discriminator provides guidance to the generator, making each other better. See Fig. 4.2 for an illustration. If the training process is successful, the generator can produce synthetic samples with similar properties as the training data. More precisely, the objective of the basic GAN models can be expressed as follows:

$$\begin{aligned}\mathcal{L}_{\text{GAN}}(G, D, Z, X) = E_{x\sim p_{\text{data}}(x)}\big[\log(D(x))\big] \\ + E_{z\sim p_{\text{data}}(z)}\big[\log(1 - D(G(z))\big])),\end{aligned}$$

and G and D together tries to solve the minimax problem

$$\min_G \max_D \mathcal{L}_{\text{GAN}}(G, D, Z, X),$$

aiming for a Nash equilibrium for this two-player non-cooperative game. In other word, the minimax solution is reached when the discriminator can't differentiate between the generated images ($G(z)$) and the real images, i.e. $D(x) = \frac{1}{2}$.

The two models are trained simultaneously where they play the two-player minimax game presented above. G tries to minimize $\log(1 - D(G(x)))$, and D maximize $\log(D(x))$. In the inner loop of the algorithm, D is updated k times, before G is updated. For the end product the optimal solution is when G generates samples from the training data distribution, and D equal $\frac{1}{2}$ all over. See Fig. 4.1.

for number of iterations **do**:

  for k steps **do**:

    - Sample m noise samples $(z^1, ..., z^m)$ from noise prior $p_g(z)$
    - Sample m examples $(x^1, ..., x^m)$ from data generating distribution $p_{data}(x)$.
    - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(x^i) + \log(1 - D(G(z^i)) \right]$$

  end for

  - Sample m noise samples $z^1, ..., z^m$ from noise prior p_g^z
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^i))$$

end for

Figure 4.1: Algorithm inspired from the original paper [39], describing how to train a GAN using minibatch stochastic gradient descent. The number of steps to apply to the discriminator is given by k and the size of each minibatch is denoted by m. Note that the notation is kept as in the original paper.

Multiple modifications and extensions to this basic setup have been proposed, aimed at providing more stable training and making the generators produce higher quality and more diverse samples. See [70] for a recent overview.

4.2 Conditional Image Synthesis with Auxiliary Classifier GANs

A generator trained using the traditional GAN setup can only produce images from one specific domain. The generator is never told what to generate. It is simply trying to produce an image from noise that is good enough to fool the discriminator.

If one wanted to use GANs to make an image gallery of cats and dogs, one would have to train multiple models. One model trained on images of dogs, and another on images of cats. To add other animals to the gallery, new models

would have to be trained. What would be better is a single GAN that can generate images from multiple classes.

4.2.1 Conditional generative adversarial networks

Conditional generative adversarial networks [71] (CGAN) is a modification of GAN that is capable of producing class-specific images. Meaning that a single CGAN model can be used to synthesize images from multiple domains.

The basic idea is to feed both the generator and the discriminator the class label as additional side information. The generator is given a class label specifying what to generate in addition to the noise, providing a mechanism to guide the image synthesis process. The discriminator is also given a class label. Its task changes from classifying if an image is real or fake, to classify if an image from a specific domain is real or fake.

To get the networks to do this we have to modify the objective function in the original GAN to include the the label information:

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}\big[\log(D(x))\big] + E_{z\sim p_z(z)}\big[\log(1-D(G(z)))\big].$$

Giving CGAN the objective function:

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}\big[\log(D(x|y))\big] + E_{z\sim p_z(z)}\big[\log(1-D(G(z|y)))\big],$$

where y is the additional side information. In the explanation above, y is the image label, but it could be any type auxiliary information.

4.2.2 Auxiliary Classifier GAN

The Auxiliary Classifier GAN [72] (ACGAN) further modifies the changes done in CGAN in order to produce an image synthesis model that could produce 128x128 resolution image samples exhibiting global coherence for all 1000 ImageNet classes.

Here the generator remains the same. It synthesises an image of a specific class based on the input label and the noise its given.

What separates this model from CGAN is that the discriminator is not getting the label-specifying side information. Instead they get the discriminator to figure out the image classes on its own. This is done by implementing what they call an "auxiliary classifier network" into the discriminator, which is a auxiliary decoder that is assigned with the task to correctly reconstruct the class labels in the images its presented. Now the discriminator has to classify what the image is and if it is artificially generated or a sample from the training data.

The modification made to the model results in changes in the objective function. Following the notation in [72], in the standard GAN model:

$$L = E[log\ P(S = real \mid X_{real})] + E[log\ P(S = fake \mid X_{fake})],$$

where $X_{fake} = G(z)$, z is the random noise feed to G. Here the discriminator D is trying to maximize the log-likelihood of L, and the generator G is aiming to minimize the second term in L.

In the ACGAN model:

$$L_S = E[log\ P(S = real \mid X_{real})] + E[log\ P(S = fake \mid X_{fake})],$$

$$L_C = E[log\ P(C = c \mid X_{real})] + E[log\ P(C = c \mid X_{fake})],$$

where $X_{fake} = G(c, z)$, c is the class-label and z is some noise as in the traditional GAN. L_S is the log-likelihood for the image source (generated or real), same as in L, while L_C is the log-likelihood of the correct class. D is trained to maximize $L_S + L_C$, while G is attempting to maximize $L_C - L_S$

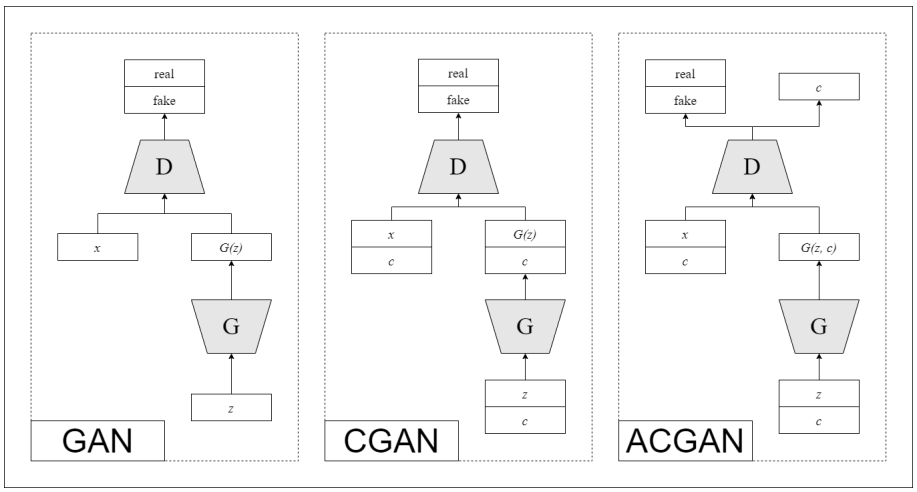

Figure 4.2: An illustration of the architectural differences between the original GAN(*left*), the CGAN(*middle*), and the ACGAN(*right*).

4.3 Image-to-image translation with Cycle-Consistent Generative Adversarial Networks

Image-to-image translation has recently been successfully approached using generative adversarial networks, e.g. pix2pix [69] and SRGAN [73].

A requirement for setting up most image-to-image models is to have a set of image pairs to be used as training data. For example, in an assignment where the generator is suppose to colorize black and white images, the training set need to contain pairs of black and white images and the same images with colors (X and Y). The generator is then given a black and white input image x, and the output is compared with the colorized image y during training. In this case one can create grayscale images from their colored counterparts, but there are many cases where the data sets does not have these image pairs, and it is extremely difficult to obtain (e.g. horse $\leftrightarrow$ zebra). One solution for this is to use the cycle-consistent generative adversarial networks (CycleGAN) model [74]

In a CycleGAN there are two generators G and F, where G transform X $\rightarrow$ Y, and F transform Y $\rightarrow$ X. While training the model, two loss functions are used to update the G and F. The Adversarial Loss, the same as for the

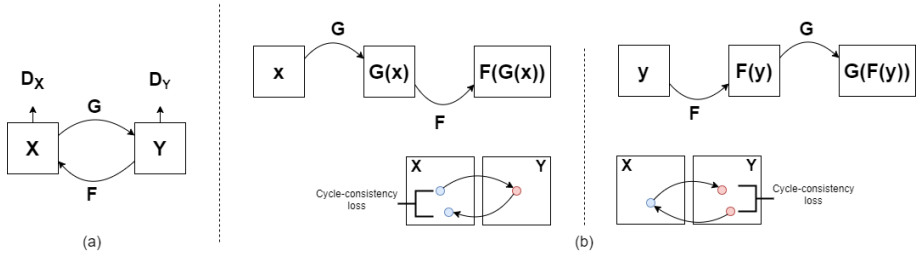

Figure 4.3: (a) shows Adversarial Loss, and (b) shows Cycle-Consistency Loss. This figure is inspired by Fig. 3 in the paper *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks* [74]

traditional GAN, introduced in 4.1, and the so-called Cycle-Consistency Loss, explained below.

It is possible for the adversarial training to generate output of both G and F that is equally distributed to the target Y and X. However, there are endless ways for G to map from X $\rightarrow$ Y. This approach individually have shown problems with reaching its potential, and often end up with the *mode collapse* problem, where all input x maps to (essentially) the same output y. This is why the Cycle Consistency Loss is introduced. It forces more structure in the mapping of X $\leftrightarrow$ Y, such that G(F(y)) $\approx$ y and F(G(x)) $\approx$ x, with the objective function:

$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) = E_{x\sim p_{\text{data}}(x)}\big[\,\|F(G(x)) - x))\|_1\big] \\ + E_{x\sim p_{\text{data}}(y)}\big[\,\|G(F(y)) - y))\|_1\big].\end{aligned}$$

In this cycle, the image x should be able to go through the cycle and end up looking like x again, and same for y. This means that x $\rightarrow$ G(x) $\rightarrow$ F(G(x)) $\approx$ x, and this results in cycle consistency (illustrated in 4.3).

4.4 Other GAN models

The huge interest around GAN have resulted in a lot of research. This have resulted in many different approaches and architectures for generating synthetic images. Table 4.1 shows some of the most influential recent GAN models.

Model	Title	Author	Year	Reference
GAN	Generative Adversarial Networks	Goodfellow et. al.	2014	[39]
DCGAN	Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks	Radford et. al.	2015	[75]
LAPGAN	Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks	Denton et. al.	2015	[76]
WGAN	Wasserstein GAN	Arjovsky et. al.	2017	[77]
SRGAN	Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network	Ledig et. al.	2017	[73]
pix2pix	Image-to-Image Translation with Conditional Adversarial Networks	Isola et. al.	2018	[69]
PGGAN	Progressive Growing of GANs for Improved Quality, Stability, and Variation	Karras et. al.	2018	[78]
BigGAN	Large Scale GAN Training for High Fidelity Natural Image Synthesis	Brock et. al.	2019	[11]
StyleGAN	A Style-Based Generator Architecture for Generative Adversarial Networks	Karras et. al.	2019	[79]
LOGAN	LOGAN: Latent Optimisation for Generative Adversarial Networks	Wu et. al.	2019	[13]
StyleGAN2	Analyzing and Improving the Image Quality of StyleGAN	Karras et. al.	2020	[12]

Table 4.1: Some of the most influential recent GAN models

Part II

Experiments

Chapter 5

Introduction to Experiments

5.1 Introduction

One of the more important aspects of machine learning is data. Data is so essential that a lot of the time, simpler models will outperform more complex ones if you simply feed them more data. Therefore, spending time on prepossessing and gathering sufficient amounts of data is crucially important when creating a machine learning model. For supervised learning, the data needs to be labeled. In many cases it is also important that the data is well balanced in order to get good results.

Acquiring well balanced data is a common hurdle when developing machine learning models. This is especially difficult when working with medical data since the data is often very unbalanced. The reason for this is that getting data for healthy patients is easy, but since such a small percentage of the population is sick it is difficult to get enough data to represent the group to a comparable extent. A possible solution to this problem is to artificially create more data by using generative adversarial networks (GANs, Chapter 4).

For this thesis the idea is to use GANs to generate realistic medical images from random noise. Ideally, we want to synthesize images in cases where the availability of real images are lacking (i.e. generate images with abnormalities, since images without are typically easier to come bye). Then use the fake generated data to train deep neural networks capable of, for example, separating the sick patients from the healthy ones. We also investigate the use GANs to generate fake abnormalities in the images from the healthy patients, creating more data representing sick patients.

5.2 Methods and Materials

5.2.1 Data

ISIC

The International Skin Imaging Collaboration (or ISIC) is a collaboration between academia and industry with the goal to reduce the number of mortalities related to melanoma by building the necessary infrastructures needed for the application of digital skin imaging. In order to reach their goal they are developing standards for what technologies, techniques, and terminology should be used for skin imaging, with attention to privacy and the ability to share the data on different platforms. To test these standards they are developing an open source public access archive of skin images. In addition to this, the archive can also be used for other purposes such as teaching or, for the development of automated diagnostic systems.

Data set

We used the training data set from the International Skin Imaging Collaboration (ISIC) Challenge 2019 [66, 80, 81], consisting of 25.331 images, each classified as either Melanoma (MEL), Melanocytic nevus (NV), Basal cell carcinoma (BCC), Actinic keratosis (AK), Benign keratosis (solar lentigo / seborrheic keratosis / lichen planus-like keratosis) (BKL), Dermatofibroma (DF), Vascular lesion (VASC) or Squamous cell carcinoma (SCC). See Fig. 6.4 for some examples.

Data preprocessing

The preprocessing were done using the pytorch transforms library. The images where resized to 128x128 in the first experiment for training the ACGAN model. For training the CycleGAN models in both experiments the images where resized to 256x256. Bicubic interpolation where used for resampling the images. The images where then converted to tensors, before being normalized with mean and standard deviation set to 0.5 for all three color channels.

5.2.2 Frameworks

We used a number of Python-based libraries in our work. In particular Pytorch and the Pytorch-based fastai library.

Pytorch

Pytorch is a open source scientific computing software written in Python, developed with two aims:

1. *"A replacement for NumPy to use the power of GPUs"* by using Tensors. These provide a lot of the same functionality as NumPy's N-dimensional arrays, but are compatible with CPUs as well as with GPUs, making computations much faster.

2. *"A deep learning research platform that provides maximum flexibility and speed"*. They build their framework to accelerates the path from research prototyping to production deployment.

Fastai

Fastai is an open source library for deep learning built on top of Pytorch, developed by Jeremy Howard et. al. [82]. The goal is to make it as easy and effective as possible to create a state-of-the-art deep learning models, such that anyone can do it. This combined with the opportunity to configure your own complex adjustments, makes this library very powerful.

5.2.3 Models

Skin lesion classifier using `fastai`

To test if our generated images could improve a classifier we needed to make this classifier, and an easy way to create a state-of-the-art image classifier, is to work with the fastai library. In this project the resnet-50 model (50 layers residual network) pretraind on the ImageNet in fastai is used to create the classifiers for testing. The residual network was introduced in 2015 by Kaiming He et. al. [59] and where explained in 3.2.2. The resnet model run the Adam optimization algorithm [83] to update its weights (see 2.2.8), and to find the optimal learning rate fastai have a learning rate finder that explore different learning rates. To speed up the training fastai gives the opportunity to train with the 1cycle policy [53] which we talked about in 2.2.10. In the first part of the training only the weights in the last layers that are added to fit this classification task is tuned, and to fine-tune the model the weights in all the layers are updated for optimal results.

5.3 Experiments

The main goal of our thesis was to generated realistic looking images of skin lesions. In the first experiment we explore different GAN models to find the optimal approach for generating these realistic looking images from random noise. In the second experiment we wanted to use GAN to see if image-to-image translation could balance the the data set, and by this improve a classifier. This work was presented at the MMIV Conference in December 2019, where it won the “Best Poster Award”.

Chapter 6

Generating Images From Noise

6.1 Introduction

The main goal was to produce images that looked visually pleasing directly from noise.

While working toward that goal we experimented with multiple GAN models with varying results. We tried unconditional GANs, where we would train one model for each class in the skin lesion data set. These images would vary a lot depending on the amount of images that where available from each class. This problem occurs because each model had to learn how to generate the skin lesion from scratch. That can be an unnecessary process since the lesions are visually similar, with small variations separating the classes. Therefore we tried using a conditional GAN model. This meant that the the same network could use all of the images. Meaning that in theory it could learn the main features from all lesions, then find what separates them in order to produce images for the specific classes.

First, we tried to use a CGAN for this. The images were very noisy and it was difficult to see if the different classes got separated. Later we tried using an ACGAN. The images produced by this model had a larger visual difference for the classes. However, the images were limited to being of size 128x128 pixels. Additionally to being in a small resolution, they contained quite a lot of artifacts, resulting in them being easily separable from the real skin lesion images.

In order to improve the visual quality of the images we decided to use CycleGAN as an image-to-image technique to make the synthetic images resemble the real ones of the same lesion class. We used CycleGAN because the model is designed to be used on image-to-image translation tasks where the images aren't paired. This was perfect for our case since the generated images where created from random noise and there where no structural match in the real data set that were easy to find. The use of the CycleGAN model for quality improvement on the ACGAN images resulted in a large improvement in the visual quality, to the extent where we had a hard time separating between the generated and the real ones.

This lead to some interesting questions. If the images where hard for us to separate from real ones, could them be used as training data for a classification model? And furthermore, are the improvements of the visual quality in the CycleGAN processed images more meaningful for the classifier than the images produced by the ACGAN model alone?

6.2 Methods and materials

6.2.1 Dataset

The data used in this project is from the ISIC 2019 Classification Task. We separated the data randomly into a training, validation and a test set. The Training and Validation data is used during the training of the GANs. The test set is only used to test the quality of the classifiers after they have been trained on the generated images.

Label	Train	Validation	Test
NV	10300	1287	1288
MEL	3618	452	452
BCC	2658	332	333
BKL	2099	262	263
AK	694	87	86
SCC	502	63	63
VASC	202	26	25
DF	191	24	24
Sum	20264	2533	2534

Table 6.1: Table showing the number of images per class for each data set.

6.2.2 Models

ACGAN

ACGAN is a GAN model that can be used to generate label-specific images (Chapter 4). It deviates from a standard GAN model since the generator is trained to generate different types of images based on the label it given. For our project it is trained to generate images based on the 8 different types of lesions in the ISIC data set.

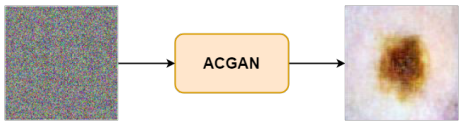

Figure 6.1: An illustration of how ACGAN is utilised in the experiment

CycleGAN

CycleGAN is a image to image style of GAN model, meaning that the generator is given a image instead of random noise. For this project it is used to transform the images generated using the ACGAN in such a way that they will look more closely to the real images in the ISIC data set of the same label.

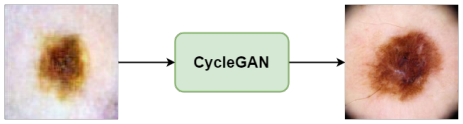

Figure 6.2: An illustration of how CycleGAN is utilised in the experiment

6.2.3 Experiment pipeline

First we use the trained ACGAN model to generate label specific images using the training data. When this model is finished training it can be used to generate an unlimited amount of images given a label and some noise. However, these images where not of the greatest quality, often containing large amount of artifacts and being confined to 128x128 pixel resolution. The second step is to try and improve the quality and image size of these images. Here we train one CycleGAN model for each label in the data sett. We do this by extracting the all the images from on label out of the training set and the ACGAN generated images of the same label type.

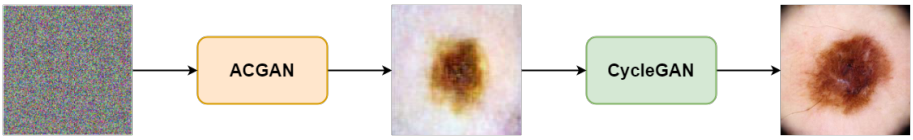

Figure 6.3: The experimental pipeline

6.3 Experimental setup

The experimental setup is composed of to different part, one looking at the visual quality of the images, the second is to test if the images can be used as additional data in the training of a classifier.

Since the syndication pipeline was mainly created with the intention of generating visually realistic skin lesion images. The first experiment is therefor a visual comparison of the generated images in relation the the real images.

To test the generated images potential for improving image classification we composed a classifier designed to correctly label images from the eight different skin lesion classes that where generated.

6.4 Evaluation setup

The testing was done by first training a baseline model. The baseline was set by training a classifier using the training set composed of images form the ISIC data set. The model was then evaluated on a test set, composed of other, unseen images from the ISIC data set. The results from the evaluation is the baseline score, compared to the models trained using additional generated images.

The data containing the synthetic images where created by taking the training set for the baseline model and then adding varying amounts of generated images to all of the classes. The amount of additional data added to each class was: 250, 500, 1000, 2000, 3000. We created training set for the ACGAN images and for the visually improved images generated from the CycleGAN, resulting in ten new training sets. The classifier was trained and evaluated from scratch for each new data set, using the test set that where used for the baseline.

6.5 Experimental results

Fig. 6.4-6.6 are the visual results, where Fig. 6.4 shows real skin lesion images from the ISIC data set, Fig. 6.5 shows images generated form noise/label pairs using the ACGAN model, and Fig. 6.6 shows the generated images in Fig. 6.5 after they have been improved by the CycleGAN model.

Fig. 6.2 shows the results from the image classification test. The **Model** column denotes what data has been used, where *ISIC-0* is the base line model trained using only the real images. In the remaining models the first part of the name (i.e. ***acgan****-250* and ***cyclegan****-250*) denotes what model has been used to generate the images, and the final number (i.e. *acgan-****250*** and *cyclegan-****250***) is the number of generated images that has been added to each class to expand on the training set used in *ISIC-0*. The columns with the class labels show the classification results (*precision/recall*) for each model, and the *Accuracy* for all classes.

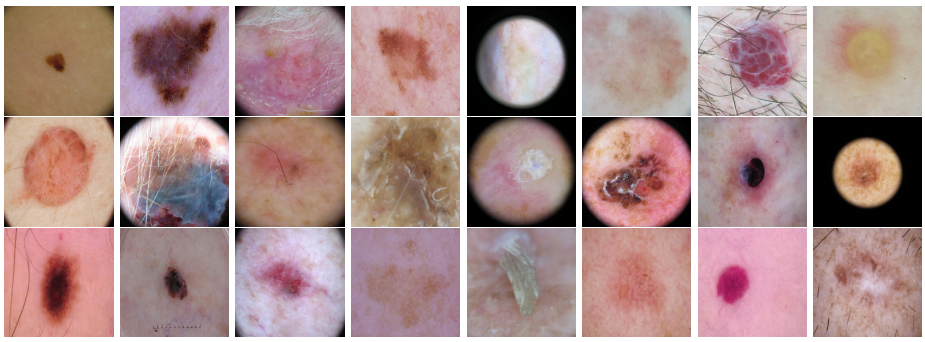

Figure 6.4: Images from the ISIC 2019 data-set. Label order: *NV, MEL, BCC, BKL, AK, SCC, VASC, DF*

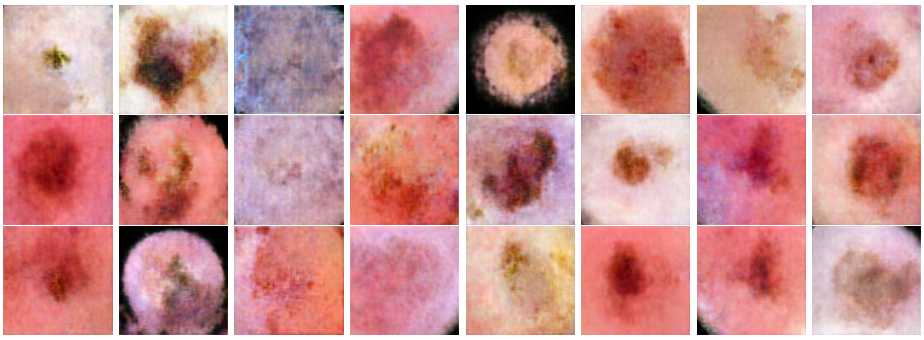

Figure 6.5: Images generated using the ACGAN model. Label order: *NV, MEL, BCC, BKL, AK, SCC, VASC, DF*

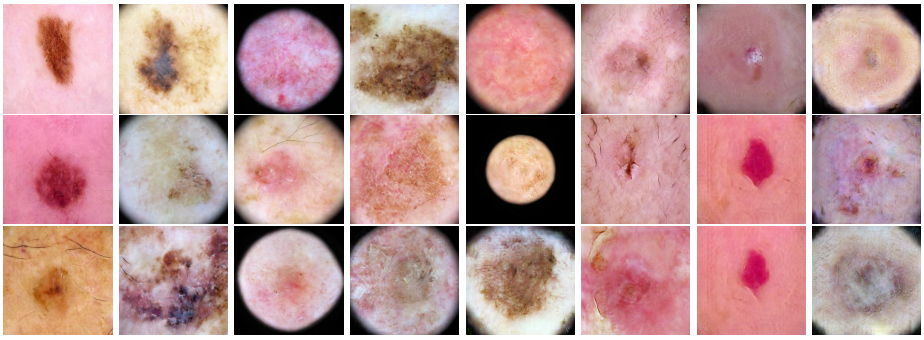

Figure 6.6: Images generated using the CycleGAN model. Label order: *NV, MEL, BCC, BKL, AK, SCC, VASC, DF*

Model	MEL	NV	BCC	AK	BKL	DF	VASC	SCC	Accuracy
ISIC-0	0.823/0.721	0.894/0.939	0.854/0.877	0.761/0.628	0.786/0.783	0.708/0.708	0.913/0.84	0.683/0.683	0.856
acgan-250	0.763/0.757	0.909/0.911	0.824/0.925	0.736/0.616	0.752/0.692	0.727/0.667	0.957/0.88	0.738/0.714	0.845
cyclegan-250	0.782/0.739	0.903/0.925	0.853/0.889	0.679/0.64	0.78/0.753	0.692/0.75	1.0/0.8	0.712/0.667	0.85
cyclegan-500	0.803/0.748	0.894/0.932	0.864/0.898	0.651/0.651	0.826/0.741	0.857/0.75	0.846/0.88	0.714/0.635	0.856
acgan-500	0.811/0.701	0.883/0.94	0.845/0.883	0.675/0.605	0.795/0.738	0.783/0.75	0.95/0.76	0.695/0.651	0.847
cyclegan-1000	0.797/0.748	0.892/0.93	0.861/0.895	0.671/0.64	0.785/0.722	0.762/0.667	0.84/0.84	0.804/0.651	0.851
acgan-1000	0.812/0.743	0.9/0.938	0.846/0.889	0.671/0.616	0.802/0.753	0.842/0.667	0.88/0.88	0.707/0.651	0.856
acgan-2000	0.839/0.701	0.885/0.946	0.839/0.889	0.685/0.709	0.793/0.73	0.783/0.75	0.95/0.76	0.698/0.587	0.852
cyclegan-2000	0.813/0.759	0.907/0.942	0.842/0.913	0.738/0.686	0.807/0.73	0.857/0.75	0.952/0.8	0.741/0.635	0.864
acgan-3000	0.806/0.706	0.896/0.932	0.844/0.91	0.638/0.698	0.785/0.734	0.8/0.667	1.0/0.8	0.644/0.603	0.848
cyclegan-3000	0.785/0.752	0.895/0.936	0.845/0.886	0.641/0.581	0.824/0.711	0.762/0.667	0.958/0.92	0.696/0.619	0.85

Table 6.2: *Model* number of generated images that has been added to the training-set, ISIC-0 is only the original training data. *Classification* results per class showing precision/recall. *Accuracy* is accuracy for all classes

Chapter 7

Generating Images From Other Classes

7.1 Introduction

In an image dataset, it can occur similarities in different classes. Take for example The Oxford-IIIT Pet Dataset [84] with images of dogs and cats where each class is the breed of the animal. Here it should be possible to find similar features in images of two different dog breeds. An image with some similarities, makes a much better base for a GAN model than to get random noise as input.

This concept is transferable to the ISIC2019 dataset [85]. In a classification task with the ISIC dataset, the classifier often has problems separating the Nevus and Melanoma class. Since melanoma in many cases develops from nevus it is logical to conclude that nevus and melanoma have feature similarities.

The approach for this chapter is inspired by the work of Jerry Wei et. al. [37], where we want to generate melanoma images from nevus images, and balance the dataset by using generated melanoma images.

Another interesting factor with this approach from Jerry Wei et. al. [37], is to see if good quality data is more important than the amount of data to generate training data that are supposed to improve a classification model.

7.2 Methods and materials

7.2.1 Data

For this experiment, we are only using the melanoma and nevus images for training the CycleGAN model (Fig. **??**). All classes from the ISIC 2019 data set is used for training the classifier for testing.

7.2.2 Models

ResNet

A ResNet classifier that is trained on the training data. The purpose of this model is then to predict all the melanoma training data, to find the images the

Label	Train	Validation	Test
NV	10300	1287	1288
MEL	3618	452	452
Sum	13918	1739	1793

Table 7.1: The number of images for each class in the data set.

model is most confident in classifying correctly. With this information we are able to do the path-rank-filtering, which you can read more about in 7.2.3

CycleGAN

A CycleGAN model that is trained on the nevus images and the α melanoma images which the ResNet model was most confident on, further explained in 7.3. When finished training the model is able to generate new images of melanoma from the nevus images.

7.2.3 Experiment pipeline

In an image to image problem with no obvious image pairs, a CycleGAN model is preferred. But to decide the training data for the model an idea from the Jerry Wei et. al. paper [37] called Path-Rank-Filter is used. When the training data is ready, the next step is to train the CycleGAN model, which is able to generate melanoma from nevus images, and vice versa, when finished training. See figure 7.1

Path-Rank-Filter

A path-rank-filter is a way to filter out data that a classifier has difficulties to predict correctly. In this case, you want to generate melanoma images from a generator trained on melanoma data that a classifier is confident is melanoma.

By using a ResNet model trained with all the original data, and make the classifier predict the training data of the class a generator is going to generate images of at a later time. Then pick an α of the images that the classifier predicted correctly with the highest confidence, and use these images to train the generator.

7.3 Experimental setup

The main objective of this experiment is to improve a classification model by adding generated images. Another interesting aspect of this experiment is to observe if good quality data, meaning data that is classified correctly with high confidence, or the amount of data is decisive for a good result, and if there are more important features than visually good looking synthetic images when training a classifier.

In this experiment we train four CycleGAN models, to generate four types of melanoma images. The difference in these CycleGAN models is the training data, which have been sent through the path-rank-filter with α=[1/8, 1/4, 1/2,

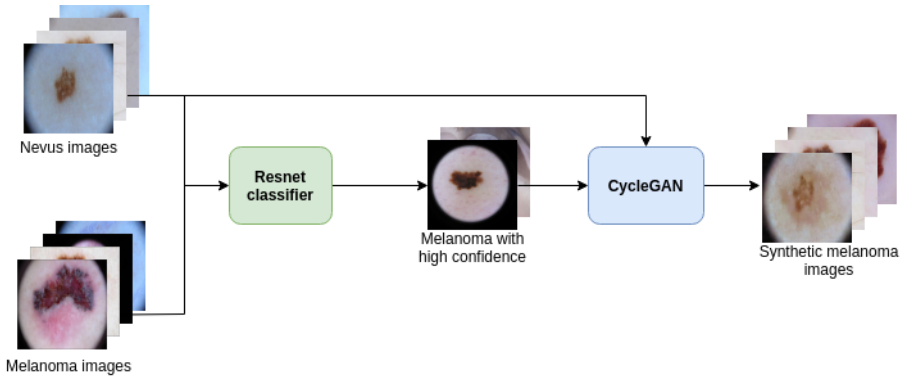

Figure 7.1: The experimental pipeline for experiment 2

1/1], and ended up as four data sets (containing the output of the path-rank filter and all the nevus images in the training set), one for each CycleGAN model. After training the models, all the nevus images was generated to their corresponding synthetic melanoma image.

7.4 Evaluation setup

For testing if the additional synthetic data could improve a classifier, we created 32 new data sets where we added different amounts of synthetic melanoma images, which was generated by different α values. All these different combinations of generated melanoma images was added to the original training set, and was used to train 32 different resnet classifier models. More specific, the data sets where made out of all the combinations of:

- α = [1/8, 1/4, 1/2, 1/1]
- Number of generated data = [500, 1000, 2000, 3000, 4000, 5000, 6000, 7000]

The images that where used was random picked, but for every α value the generated melanoma image from the same nevus image was used, and the images in 500 $\in$ 1000 $\in$ 2000 and so on. The generated image was then added to the original training set to train the test classifier.

7.5 Experimental results

Figure 7.2 shows a sample of generated melanoma images. To the naked eye it looks like the higher α is, the images are more similar to their original nevus images. This provides basis to belief that with a lower α the CycleGAN model uses more of the dominating features in the melanoma images to generate new ones. To test if these generated images is of any use as an augmentation approach, we have completed a set of classification tasks with different training data. Different amounts of synthetic MEL images, which is generated with different α value, is added to the original dataset. See results in table 7.2.

	$\alpha = \frac{1}{8}$			$\alpha = \frac{1}{4}$			$\alpha = \frac{1}{2}$			$\alpha = \frac{1}{1}$		
		MEL			**MEL**			**MEL**			**MEL**	
Num. gen.	**acc.**	**prec**	**recall**	**acc.**	**prec**	**recall**	**acc.**	**prec**	**recall**	**acc.**	**prec**	**recall**
500	0.852	0.797	0.712	0.853	0.807	0.712	0.857	0.820	0.724	0.856	0.819	0.710
1000	0.851	0.824	0.717	0.848	0.797	0.719	0.861	0.843	0.701	0.851	0.825	0.708
2000	0.857	0.815	0.730	0.847	0.817	0.719	0.856	0.836	0.732	0.856	0.828	0.712
3000	0.858	0.837	0.715	0.857	0.837	0.728	0.857	0.833	0.726	0.859	0.825	0.746
4000	0.854	0.801	0.746	0.856	0.802	0.728	0.864	0.855	0.715	0.854	0.807	0.730
5000	0.858	0.848	0.717	0.856	0.804	0.754	0.855	0.818	0.737	0.858	0.811	0.741
6000	0.865	0.851	0.759	0.852	0.816	0.724	0.856	0.825	0.730	0.858	0.819	0.741
7000	0.854	0.781	0.748	0.864	0.810	0.763	0.849	0.783	0.743	0.856	0.819	0.752
ISIC-0	0.856	0.823	0.721	0.856	0.823	0.721	0.856	0.823	0.721	0.856	0.823	0.721

Table 7.2: The results from the classification test in experiment 2. The rows is number of generated melanoma images added to the original training data, ISIC-0 is when no generated data is added. α is the value in the path-rank-filter. *acc.* is the all over accuracy when predicting on the test set. *MEL prec/recall* is the class specific precision and recall for the melanoma class.

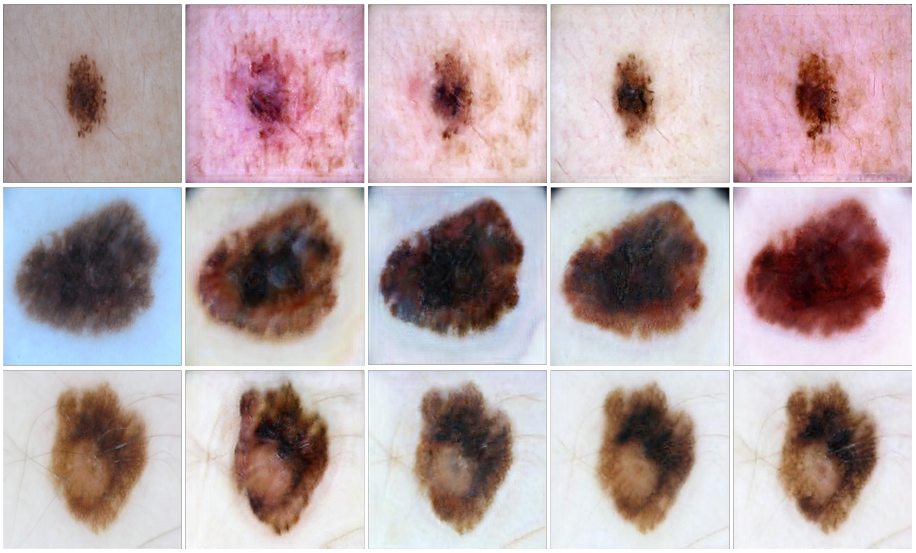

Figure 7.2: Column 1 - Nevus images, column 2 - generated melanoma images α=1/8, column 3 - generated melanoma images α=1/4, column 4 - generated melanoma images α=1/2, column 5 - generated melanoma images α=1/1

Chapter 8

Discussion

In this chapter we discuss the results obtained in the above experiments, and look at reasons that can explain what we did and didn't achieve with our various approaches. We point to general challenges with GANs, linking them to what we have observed throughout our work. Finally, we discuss some future work that could potentially lead to improved results.

8.1 Results

In this work we have been able to reach our goal in generating realistic looking images of skin lesions (seen in figure 6.6 and 7.2), and for an untrained eye it would be difficult to separate the real images from the generated images. In our two approaches, generating synthetic images from random noise (Experiment 1), and an image translation with an image-to-image approach (Experiment 2), we have show that GANs have the ability to generate realistic looking synthetic images.

When it comes to using this generated data to improve a classification model, we have not been able to find an approach that we can confidently recommend. We have found cases that see some improvements, but also some for which adding synthetic images lessens classification accuracy. When looking at the tables 6.2 and 7.2, the numbers seems to be a bit random, and there is no clear structure in the results.

8.2 Challenges with GAN

8.2.1 Failure to converge

A possible pitfall when training GANs is that it can have problems with converging, meaning that the model has a hard time reaching a Nash equilibrium, but for example rather jump back and forth between lowering the loss of the generator and lowering the loss of the discriminator. There is a variety of different theoretical interpretations for why this problem occurs, and each variety of GAN has their own set of convergence issues.

In the original GAN paper they speculate that the cause of stagnant training can be traced back to how the objective function is built. In our earlier chapter

on GANs, we discussed that the objective function is a minimax game between the discriminator and the generator, where improvements to one network is supposed to cause improvements to the other.

But if the discriminator becomes to good at differentiating the generated images from the real ones too early on in the training process, it can be difficult for the generator to know what changes it needs to do in order to improve its output. If this happens the generator can reach a point where it never figure out how to adjust itself in the right direction, failing to converge.

In the early stages of our research we noticed this failure to converge while implementing a simple GAN model to produce skin lesions images from random noise. All outputs from the generator were these green and red images that remained the same no matter how long it was trained for. Fig. 8.1 shows an example.

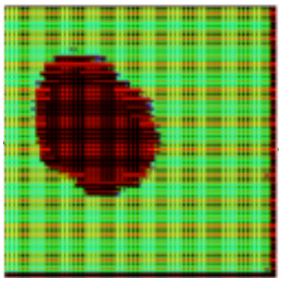

Figure 8.1: A generated image from one of our early attempts at constructing a working GAN model

8.2.2 Mode collapse

A good generator should generate outputs of a wide variety over the distribution of the data. Mode collapse is when the generator excludes parts of the data distribution when generating images, and end up generating the same, or close to the same, output whatever the input. Take for example the MNIST data set, if the generator only outputs images of the digit 3, and all the other numbers in the data distribution never appear, you are dealing with a mode collapse. In practice, GAN generators can quickly end up in this situation, until the discriminator starts classifying these digit correctly as being fake. At which point the generator end up in another collapse, only generating images of another digit. And so on, ad nauseam. This is one of the biggest challenges of training GAN models. For technical details of what this is and why it happens see [86].

To know if this has happened in our work is difficult to say. In our generated data we can see that there is a lot of variety in the images that is produced. But there is a high probability that we have a mode collapse at some level, even though it is difficult to discover. We can also see examples in our generated data that some images are very similar to each other, signifying mode collapse.

8.2.3 Need for data

To generate images of high quality the need of a large and high quality dataset is crucial [87]. State-of-the-art models generating high quality realistic models such as [11, 13, 79] all use large data set to achieve their results. To generate images of such high quality requires an enormous amount of computing power and time. To use images like these for data augmentation would be cumbersome, and likely not even be very impactfule for a classifier, as the already complex original data set could give good results on its own.

8.2.4 Difficult to evaluate

The lack of evaluation metricis for GAN makes it difficult to improve and adjust the model. Even though many measures have been introduced, visual examination is still the go-to method. This is time consuming, and you often need an expert to do a thorough evaluation [86]. In the paper [88] they review many of the evaluation measures that have been introduced to GAN, and point out the importance of settling on a few good measures to steer the progress.

8.2.5 Is GAN the way to go for data augmentation?

To improve a classifier with synthetic data, does the optimal generated data look as realistic as possible? In the work of Han et. al. they conclude that realistic images does not always guarantee better data augmentation [89]. While a discriminator in a GAN model predicts if an image looks real or not, should it not instead see if the generated image improve a classifier? An example of this approach is Uber AI Labs recent work reported in [90]

8.3 Our approach

While working with the thesis, we have been through many types of GAN models with different approaches. The product we have presented here is the approaches we found to give the best results in generating as realistic skin lesion images as possible. Other models we have tried to generate synthetic images of is Wasserstein GAN [77], Conditional GAN [69] and BigGAN [11], but none of them gave the visual quality we where looking for.

We have also looked at the segmentation problem from the ISIC 2018 data set [66]. Here we tried to generate both skin lesion images and its segmentation mask. We explored the models, Wasserstein GAN [77], Deep Convolutional GAN [75], perceptual loss [91] and Bicycle GAN [92], but because of the lack of improvement in our model tasked with segmenting lesions by using these synthetic masks, we put the problem aside.

In this work we have not focused that much on fine tuning hyperparameters. This is because of the extremely long training time in many cases, which makes trying multiple parameter settings time consuming, and the difficulties of evaluating small improvements in the output of the generator just by trying to spot the visual differences in the generated images. That is why we have prioritized exploring different GAN models, where it is a lot more variation in the output, and easier to spot which model has the potential or not.

While exploring these GAN models we have stumbled upon some of the challenges described in section 8.2. In figure 8.2 we can see some examples of generated images from our CGAN model and BigGAN model. Because of the similarities in the images generated from both models, much indicates that these models have a mode collapse problem. The bad quality in the images is probably because we don't have a big enough data set. BigGAN is known for needing a lot of training data, such as the 1.4 million images in ImageNet, so it's reasonable to conclude that the bad quality is at least partly a result of not having enough data.

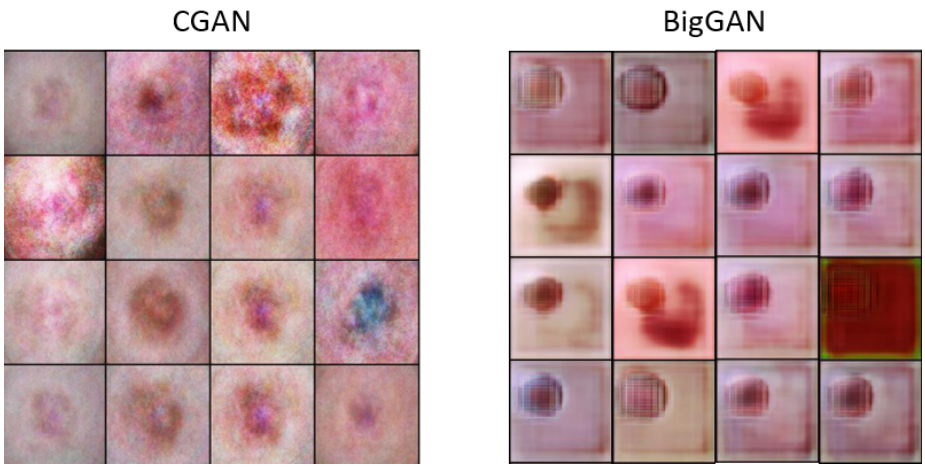

Figure 8.2: Examples of generated images from our CGAN and BigGAN models. A lot indicates that these images suffers from mode collapse and the need of more training data.

8.4 Further work

The main focus in the work reported in our thesis was to test a variety of different GAN models, in order to find the ones that worked best. The process of finding new models and setting up a functional environment, including implementation of the models, was very time consuming. In addition to that, the training time for GAN models can be extremely slow as well. Some of the models took weeks to train, sometimes without converging.

This left some "intended work" that we never found the time to execute. In this section we will discuss some those project.

One of the things we would have liked to spend more time on is to try to improve the quality of the models used in our experiments. Generative adversarial nets are notoriously hard to train. Since they often requires a lot of manual supervision (checking if the generator output makes sense) and fine tuning of parameters. In [38] they were successful in using a GAN to mitigate problems related to data imbalance for classifier training, but also noted that the successful model demanded a lot of supervision in order to perform: "we also want to emphasize that training the LAPGAN is very difficult, requiring constant supervision and adjustment of hyperparameters". Ideally, we would have used a large-scale grid search or a similar strategy to find sets of hyperparameters resulting in high performance. But this would have been far too time-consuming given the time and computational resources we had available.

Another aspect that we would like to investigate is different ways of testing the quality of the generated images.

In [37], the paper that our second experiment was motivated by, they had an additional test we did not have the time to perform. They devised what they called a "Turing-test" where the generated images where mixed in with real images, and then displayed to experts that would try to separate between the two.

This style of image evaluation is also mention in [30] that reviews the use

of GANs for medical imaging. Where they recognize that this approach to evaluation can be time consuming and not easily scalable. In response to that they list several other ways of evaluation used in the papers they reviewed. It could certainly be interesting to try and use these ways of measure on our models and compare the results to theirs.

It could also be useful to use our pipelines on similar projects, medical or not, and then compare our generated images with those obtained by others.

In addition to further improve the models used, it would be interesting to explore even more GAN models as well.

There are many variations of GAN that has shown promising results in medical image synthesis, like the LAPGAN [76]. That was successfully used for medical image synthesis where the goal was to improve classification results caused by data imbalance [38].

In addition to popular models for medical imaging it would be interesting to try other GAN models that has shown promising results in non-medical image synthesis. I.e. [90] where they use a different approach to model training. In the standard usages the generators output is evaluated by how good the images looked to the human eye. Here the images are, during training, evaluated by there usefulness in improving classification models. This can lead to some interesting results, where the images that are generated could look like a total mess and seemingly completely useless, but they improved the classifier. Which poses some very interesting questions for our approach of generating realistic looking images with the intended use as training data for a image classifier.

8.5 Conclusion

When we started this work we formulated two research questions (see section 1.4). The first one was to see if we could generate realistic-looking images of skin lesions, and the second was to see if the generated images could be used as training data to improve a classifier.

For the first question we have overcome the challenges of failure to converge and mode collapse, at least to a point where we are able to generate images that are visually pleasing, look realistic and have quite a lot of variety. In our first experiment we came up with a pipeline that we haven't seen anyone else use before us. The pipeline was able to generate realistic-looking images, and have outperformed all our other attempts in generating images of skin lesions from random noise.

When it comes to the second research question, it has turned out to be more challenging. When using the generated data to train a classifier, we have not been able to see any improvement. Because of this it has been difficult to recommend an approach where GANs should be used for data augmentation. Others have also asked the question if GAN is a good tool for data augmentation [93], and in the work of Perez et. al. they show that traditional augmentation gives better results then applying GANs as a data augmentation tool [94]. Figuring out whether a positive answer to our second research question exists requires further work aimed at discovering and surmounting the current limitations for GAN models and their ability to move beyond the training data distribution.

References

[1] World Health Organization. Cancer. https://www.who.int/news-room/fact-sheets/detail/cancer, 2012. Online; accessed 17 September 2019.

[2] The Skin Cancer Foundation. Skin Cancer Facts Statistics. https://www.skincancer.org/skin-cancer-information/skin-cancer-facts/, 2019. Online; accessed 17 September 2019.

[3] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[4] Holger A Haenssle, Christine Fink, R Schneiderbauer, Ferdinand Toberer, Timo Buhl, A Blum, A Kalloo, A Ben Hadj Hassen, Luc Thomas, A Enk, et al. Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. *Annals of Oncology*, 29(8):1836–1842, 2018.

[5] Alexander Selvikvåg Lundervold and Arvid Lundervold. An overview of deep learning in medical imaging focusing on MRI. *Zeitschrift für Medizinische Physik*, 29(2):102–127, 2019.

[6] Titus J Brinker, Achim Hekler, Alexander H Enk, Carola Berking, Sebastian Haferkamp, Axel Hauschild, Michael Weichenthal, Joachim Klode, Dirk Schadendorf, Tim Holland-Letz, et al. Deep neural networks are superior to dermatologists in melanoma image classification. *European Journal of Cancer*, 119:11–17, 2019.

[7] Stephanie Chan, Vidhatha Reddy, Bridget Myers, Quinn Thibodeaux, Nicholas Brownstone, and Wilson Liao. Machine Learning in Dermatology: Current Applications, Opportunities, and Limitations. *Dermatology and therapy*, pages 1–22, 2020.

[8] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *Jama*, 316(22):2402–2410, 2016.

[9] Rishab Gargeya and Theodore Leng. Automated Identification of Diabetic Retinopathy Using Deep Learning. *Ophthalmology*, 124(7):962–969, 2017.

[10] Jaakko Sahlsten, Joel Jaskari, Jyri Kivinen, Lauri Turunen, Esa Jaanio, Kustaa Hietala, and Kimmo Kaski. Deep Learning Fundus Image Analysis for Diabetic Retinopathy and Macular Edema Grading. *Scientific reports*, 9(1):1–11, 2019.

[11] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[12] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and Improving the Image Quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[13] Yan Wu, Jeff Donahue, David Balduzzi, Karen Simonyan, and Timothy Lillicrap. LOGAN: Latent Optimisation for Generative Adversarial Networks. *arXiv preprint arXiv:1912.00953*, 2019.

[14] The Skin Cancer Foundation. Skin Cancer 101. `https://www.skincancer.org/skin-cancer-information/`, 2019. Online; accessed 17 September 2019.

[15] Norsk Helseinformatikk AS. Hudkreft - animasjon. `https://nhi.no/animasjoner/hud/hudkreft/`, 2014. Online; accessed 17 September 2019.

[16] Shivangi Jain, Nitin Pise, et al. Computer aided Melanoma skin cancer detection using Image Processing. *Procedia Computer Science*, 48:735–740, 2015.

[17] The International Skin Imaging Collaboration. Background. `https://challenge2019.isic-archive.com/background.html`, 2019. Online; accessed 17 September 2019.

[18] Inger Kristin Larsen, B Møller, TB Johannesen, S Larønningen, T Robsahm, T Grimsrud, and G Ursin. Cancer in Norway 2016-Cancer incidence, mortality, survival and prevalence in Norway. *Book Cancer in Norway 2010–Cancer Incidence, Mortality, Survival and Prevalence in Norway (Editor ed.ˆ eds)*, 2017.

[19] Norsk Elektronisk Legehandbok. Malignt melanom. `https://legehandboka.no/handboken/kliniske-kapitler/hud/pasientinformasjon/foflekker-pigmenterte-utslett/foflekkreft-malignt-melanom`, 2019. Online; accessed 17 September 2019.

[20] Hensin Tsao, Jeannette M Olazagasti, Kelly M Cordoro, Jerry D Brewer, Susan C Taylor, Jeremy S Bordeaux, Mary-Margaret Chren, Arthur J Sober, Connie Tegeler, Reva Bhushan, et al. Early Detection of Melanoma: Reviewing the ABCDEs. *Journal of the American Academy of Dermatology*, 72(4):717–723, 2015.

[21] Darrell S Rigel, Julie Russak, and Robert Friedman. The Evolution of Melanoma Diagnosis: 25 Years Beyond the ABCDs. *CA: a cancer journal for clinicians*, 60(5):301–316, 2010.

[22] John F Thompson, Richard A Scolyer, and Richard F Kefford. Cutaneous Melanoma. *The Lancet*, 365(9460):687–701, 2005.

[23] Harold Kittler, H Pehamberger, K Wolff, and MJTIO Binder. Diagnostic Accuracy of Dermoscopy. *The lancet oncology*, 3(3):159–165, 2002.

[24] ME Vestergaard, PHPM Macaskill, PE Holt, and SW Menzies. Dermoscopy Compared With Naked Eye Examination for the Diagnosis of Primary Melanoma: A Meta-Analysis of Studies Performed in a Clinical Setting. *British Journal of Dermatology*, 159(3):669–676, 2008.

[25] Ashley Privalle, Thomas Havighurst, KyungMann Kim, Daniel D Bennett, and Yaohui G Xu. Number of Skin Biopsies Needed Per Malignancy: Comparing the Use of Skin Biopsies Among Dermatologists and Nondermatologist Clinicians. *Journal of the American Academy of Dermatology*, 82(1):110–116, 2020.

[26] Andre GC Pacheco and Renato A Krohling. The impact of patient clinical information on automated skin cancer detection. *Computers in biology and medicine*, 116:103545, 2020.

[27] Abhishek Bhattacharya, Albert Young, Andrew Wong, Simone Stalling, Maria Wei, and Dexter Hadley. Precision Diagnosis Of Melanoma And Other Skin Lesions From Digital Images. *AMIA Summits on Translational Science Proceedings*, 2017:220, 2017.

[28] Shivangi Jain, Nitin Pise, et al. Computer aided Melanoma skin cancer detection using Image Processing. *Procedia Computer Science*, 48:735–740, 2015.

[29] Ana Filipa Duarte, Altamiro da Costa-Pereira, Veronique Del-Marmol, and Osvaldo Correia. Are General Physicians Prepared for Struggling Skin Cancer?—Cross-Sectional Study. *Journal of cancer education*, 33(2):321–324, 2018.

[30] Xin Yi, Ekta Walia, and Paul Babyn. Generative adversarial network in medical imaging: A review. *Medical image analysis*, page 101552, 2019.

[31] Jelmer M Wolterink, Tim Leiner, Max A Viergever, and Ivana Išgum. Generative Adversarial Networks for Noise Reduction in Low-Dose CT. *IEEE transactions on medical imaging*, 36(12):2536–2545, 2017.

[32] Yuhua Chen, Feng Shi, Anthony G Christodoulou, Yibin Xie, Zhengwei Zhou, and Debiao Li. Efficient and Accurate MRI Super-Resolution using a Generative Adversarial Network and 3D Multi-Level Densely Connected Network. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 91–99. Springer, 2018.

[33] Ohad Shitrit and Tammy Riklin Raviv. Accelerated Magnetic Resonance Imaging by Adversarial Neural Network. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 30–38. Springer, 2017.

[34] Dong Yang, Daguang Xu, S Kevin Zhou, Bogdan Georgescu, Mingqing Chen, Sasa Grbic, Dimitris Metaxas, and Dorin Comaniciu. Automatic Liver Segmentation Using an Adversarial Image-to-Image Network. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 507–515. Springer, 2017.

[35] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *Icdar*, volume 3, 2003.

[36] Pedro Costa, Adrian Galdran, Maria Inês Meyer, Michael David Abràmoff, Meindert Niemeijer, Ana Maria Mendonça, and Aurélio Campilho. Towards Adversarial Retinal Image Synthesis. *arXiv preprint arXiv:1701.08974*, 2017.

[37] Jerry Wei, Arief Suriawinata, Louis Vaickus, Bing Ren, Xiaoying Liu, Jason Wei, and Saeed Hassanpour. Generative Image Translation for Data Augmentation in Colorectal Histopathology Images. *arXiv preprint arXiv:1910.05827*, 2019.

[38] Christoph Baur, Shadi Albarqouni, and Nassir Navab. MelanoGANs: High Resolution Skin Lesion Synthesis with GANs. *arXiv preprint arXiv:1804.04338*, 2018.

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

[40] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1033–1038. IEEE, 1999.

[41] David J Heeger and James R Bergen. Pyramid-Based Texture Analysis/Synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238, 1995.

[42] Jeremy S De Bonet. Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 361–368, 1997.

[43] CJ Rose and CJ Taylor. A Statistical Model of Texture for Medical Image Synthesis and Analysis. *Med. Image Understand. Anal*, pages 1–4, 2003.

[44] Amod Jog, Snehashis Roy, Aaron Carass, and Jerry L Prince. Magnetic resonance image synthesis through patch regression. In *2013 IEEE 10th International Symposium on Biomedical Imaging*, pages 350–353. IEEE, 2013.

[45] Michael A Nielsen. *Neural Networks and Deep Learning*, volume 2018. Determination press San Francisco, CA, USA:, 2015.

[46] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5 (4):115–133, 1943.

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

[48] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[49] Fast.ai. vision.transform. `https://docs.fast.ai/vision.transform.html`, 2020. Online; accessed 15 May 2020.

[50] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

[51] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[52] Sean Murray. An Exploratory Analysis of Multi-Class Uncertainty Approximation in Bayesian Convolutional Neural Networks. Master's thesis, University of Bergen, 2018.

[53] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[54] Jeremy Howard and Sylvain Gugger. fastai: A Layered API for Deep Learning. *Information*, 11(2):108, 2020.

[55] Emma Beede, Elizabeth Baylor, Fred Hersch, Anna Iurchenko, Lauren Wilcox, Paisan Raumviboonsuk, and Laura Vardoulakis. A Human-Centered Evaluation of a Deep Learning System Deployed in Clinics for the Detection of Diabetic Retinopathy. 2020.

[56] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[57] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[58] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural computation*, 1(4):541–551, 1989.

[59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[60] Alok Sharma, Edwin Vans, Daichi Shigemizu, Keith A Boroevich, and Tatsuhiko Tsunoda. DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture. *Scientific reports*, 9(1):1–7, 2019.

[61] Cicero Dos Santos and Maira Gatti. Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, 2014.

[62] Ye Jia, Yu Zhang, Ron Weiss, Quan Wang, Jonathan Shen, Fei Ren, Patrick Nguyen, Ruoming Pang, Ignacio Lopez Moreno, Yonghui Wu, et al. Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis. In *Advances in neural information processing systems*, pages 4480–4490, 2018.

[63] Alexander LeNail. NN-SVG: Publication-Ready Neural Network Architecture Schematics. *Journal of Open Source Software*, 4(33):747, 2019.

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

[65] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[66] Noel CF Codella, David Gutman, M Emre Celebi, Brian Helba, Michael A Marchetti, Stephen W Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, et al. Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging Collaboration (ISIC). In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 168–172. IEEE, 2018.

[67] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[68] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[69] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-Image Translation with Conditional Adversarial Networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[70] Karol Kurach, Mario Lučić, Xiaohua Zhai, Marcin Michalski, and Sylvain Gelly. A Large-Scale Study on Regularization and Normalization in GANs. In *International Conference on Machine Learning*, pages 3581–3590, 2019.

[71] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *arXiv preprint arXiv:1411.1784*, 2014.

[72] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional Image Synthesis With Auxiliary Classifier GANs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org, 2017.

[73] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[74] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[75] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv preprint arXiv:1511.06434*, 2015.

[76] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[77] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[78]

[79] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[80] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data*, 5:180161, 2018.

[81] Marc Combalia, Noel CF Codella, Veronica Rotemberg, Brian Helba, Veronica Vilaplana, Ofer Reiter, Allan C Halpern, Susana Puig, and Josep Malvehy. BCN20000: Dermoscopic Lesions in the Wild. *arXiv preprint arXiv:1908.02288*, 2019.

[82] Jeremy Howard et al. fastai. `https://github.com/fastai/fastai`, 2018.

[83] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[84] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[85] Noel Codella, Veronica Rotemberg, Philipp Tschandl, M Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti, et al. Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging Collaboration (ISIC). *arXiv preprint arXiv:1902.03368*, 2019.

[86] Divya Saxena and Jiannong Cao. Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions. *arXiv preprint arXiv:2005.00065*, 2020.

[87] Keyang Cheng, Rabia Tahir, Lubamba Kasangu Eric, and Maozhen Li. An analysis of generative adversarial networks and variants for image synthesis on MNIST dataset. *Multimedia Tools and Applications*, pages 1–28, 2020.

[88] Ali Borji. Pros and Cons of GAN Evaluation Measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[89] Changhee Han, Hideaki Hayashi, Leonardo Rundo, Ryosuke Araki, Wataru Shimoda, Shinichi Muramatsu, Yujiro Furukawa, Giancarlo Mauri, and Hideki Nakayama. GAN-based synthetic brain MR image generation. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 734–738. IEEE, 2018.

[90] Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data. *arXiv preprint arXiv:1912.07768*, 2019.

[91] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[92] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward Multimodal Image-to-Image Translation. In *Advances in neural information processing systems*, pages 465–476, 2017.

[93] Christoph Baur, Shadi Albarqouni, and Nassir Navab. Generating Highly Realistic Images of Skin Lesions with GANs. In *OR 2.0 Context-Aware Operating Theaters, Computer Assisted Robotic Endoscopy, Clinical Image-Based Procedures, and Skin Image Analysis*, pages 260–267. Springer, 2018.

[94] Luis Perez and Jason Wang. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *arXiv preprint arXiv:1712.04621*, 2017.