UNIVERSITY OF TURKU

Weather Image Generation using a Generative Adversarial Network

Software engineering

Master's Degree Programme in Information and Communication Technology

Department of Computing, Faculty of Technology

Master of Science in Technology Thesis

Author:

Rami Ilo

Supervisors:

Docent Fahimeh Farahnakian (University of Turku)

March 2023

Master of Science in Technology Thesis
Department of Computing, Faculty of Technology
University of Turku

Subject: Software engineering
Programme: Master's Degree Programme in Information and Communication Technology
Author: Rami Ilo
Title: Weather Image Generation using a Generative Adversarial Network
Number of pages: 50 pages, 0 appendix pages
Date: March 2023

This thesis will inspect, if coupling a simple U-Net segmentation model with an image-to-image transformation Generative Adversarial Network, CycleGAN, will improve data augmentation result compared to sole CycleGAN. To evaluate the proposed method, a dataset consisting of weather images of different weather conditions and corresponding segmentation masks is used. Furthermore, we investigate the performance of different pre-trained CNNs in the encoder part of the U-Net model. The main goal is to provide a solution for generating data to be used in future data augmentation projects for real applications.

Images of the proposed segmentation and CycleGAN model pipeline will be evaluated with Fréchet Inception Distance metric, and compared to sole CycleGAN results.

The results indicate that there is an increase in generated image quality by coupling a segmentation model with a generator of CycleGAN, at least with the used dataset. Additional improvements might be achieved by adding an attention model to the pipeline or changing the segmentation or generative adversarial network model architectures.

Keywords: Generative Adversarial Network, GAN, Deep learning, Convolutional Neural Network, Image-to-image translation, Semantic Segmentation

Diplomityö
Tietotekniikan laitos, Teknillinen tiedekunta
Turun yliopisto

Oppiaine: Ohjelmistotekniikka
Tutkinto-ohjelma: Tieto- ja viestintätekniikka
Tekijä: Rami Ilo
Otsikko: Weather Image Generation using a Generative Adversarial Network
Sivumäärä: 50 sivua, 0 liitesivua
Päivämäärä: Maaliskuu 2023

Tämä tutkielma selvittää, tuottaako yksinkertaisen U-Net segmentaatiomallin yhdistäminen kuvasta-kuvaan generatiiviseen vastakkaisverkkoon, CycleGANiin, parempia tuloksia kuin pelkkä CycleGAN. Esitetyn ratkaisun arvioimiseksi käytetään sääkuvista ja niitä vastaavista segmentaatioleimoista koostuvaa datasettiä. Lisäksi tutkimme, paljonko eroavaisuuksia esiopetetuilla CNN:llä on U-Net arkkitehtuurin enkooderissa suorituskyvyn osalta. Tutkielman päätavoite on tuottaa ratkaisu uuden datan generoimiseksi reaalimailman sovelluskohteisiin.

Ehdotetun segmentaatio- ja CycleGAN-mallista koostuvan liukuhihnan suorituskyky arvioidaan Fréchetin aloitusetäisyys-menetelmällä, jota myös verrataan pelkällä CycleGANilla saatuihin tuloksiin.

Tutkielman tulokset implikoivat, että kuvanlaatu nousee esitettyä liukuhihnaa käyttämällä ainakin kyseessä olevalla datasetillä. Lisäparannuksia voi saada aikaan liukuhihnaan erillisen huomiomallin tai muuttamalla segmentaatio- tai generatiivisen vastakkaisverkon arkkitehtuuria.

Asiasanat: Generatiivinen vastakkaisverkko, GAN, syväoppiminen, Konvolutiivinen Neuroverkko, Kuvasta-kuvaan muunnos, Semanttinen segmentaatio

Table of contents

1 Introduction

1.1 Research background

Machine learning is the process of a program gaining new information from existing data to be used in novel situations. Generative Adversarial Network (GAN), a type of deep machine learning architecture, can generate new data by using a setup of two competing neural networks, and this new data can be further refined to address some data extensive problem. Augmenting new data examples with GANs can be used for example to generate new data for training a different model, such as computerized tomography scans, a stack of X-ray images of cancer patients to train a segmentation model [1], as collecting real data of medical patients can be expensive and slow, or to find some new knowledge about the patterns of the data. Generating new types of cyber-attacks [2] or drug types [3] can help to automate testing of different fields.

It might be difficult to collect images of every weather condition from every location, but with GANs one might need to only have a single image of a given location to use it to generate images of different weather conditions of the said location. These generated images could then be further used for environment design, traffic planning, etc.

Architecture of a GAN and deep learning model general is sometimes a black box, which makes choosing the right layers and hyperparameters empirical; this property leaves much room for optimization, and there are possible unfound solutions which might perform better. Therefore, it's important to both look into generally accepted practices and wherever possible, attempt to slightly alter an architecture.

1.2 Literature review

After the introduction on GANs in 2014 [4], the architecture has seen many different implementations. These include StyleGAN [5], CycleGAN [6] and DiscoGAN [7].

StyleGAN introduces a specific style-vector. StyleGAN generator model has upscaling structure, starting from very low resolutions. At each resolution, random noise input is fed to a fully-connected mapping network to produce the style-vector, which is combined with Adaptive instance normalization [8] and then fed to the generator model. This method is used to train and modify features at different resolutions, be it (in case of face generation) hair color or eyelash length etc. However, one common defect of StyleGAN was appearance

of artifacts or “blobs”, which during training appear to gradually transform into different features, but are almost always present somewhere in the image. Later, StyleGAN was improved with StyleGAN v2 [9], which solved the artifact problem by removing Adaptive instance normalization layers from the generator.

CycleGAN, the model used in this thesis, learns the mapping between two different image domains without requiring the images to be paired. Not requiring the images to be paired makes the data collection much easier, as one isn’t required to make image pairs with static backgrounds where only the domain subject varies. The idea during training is to transfer the input image from domain *X* to domain *Y* and then back to *X*, and measure the difference between the initial and final *X* domain images.

DiscoGAN works for the most part the same way as CycleGAN, the main difference being that CycleGAN uses one cycle-consistent loss between the models during training, whereas DiscoGAN uses two reconstruction losses, one for each generator. This difference might make CycleGAN more stable and require less supervision during training.

Challenges of training GANs include training instability (as there are two models, which improve each other in turn, so they have to be in sync), requirement for a large amount of data, overfitting to a certain output (mode collapse), and generally a large computational cost compared to classification models, for example [10]. Evaluating GANs can be expensive, but solutions including Fréchet Inception Distance [11] or Kernel Inception Distance [12] are commonly used by comparing images generated by a GAN model to the last feature layer of a pre-trained model, most commonly Inception v3 [13].

Out of the common GAN models, CycleGAN appears to be well-suited and popular choice for translating an input image to another domain with reasonable amount of data and training time [14]–[16], whereas StyleGAN excel in generating completely new images from random noise input, but is expensive computationally and requires a lot of data.

Segmentation models aim to detect certain areas from the input and distinguish different objects. Combining a generator of a GAN model with a segmentation model can improve the quality of the generated images, as the segmentation model can be used to both emphasize areas of interest, or to ensure that the generator detects certain objects. WeatherGAN [17] combines the generator of a GAN with a segmentation and attention

models, and trains all models simultaneously. In this thesis, the same idea is used but by having a distinctly trained segmentation model and GAN-generator model.

1.3 Research questions

The thesis will revolve around the following two research questions:

RQ1: Investigate, if there is significant difference in data augmentation results between CycleGAN output and CycleGAN output coupled with segmentation model.

RQ2: Inspect how well different segmentation model configurations and pre-trained models affect segmentation results and quality of images outputted by the proposed CycleGAN-segmentation model pipeline.

Given the research questions, this thesis will have a look and implement modern data augmentation techniques with GANs for the purpose of modifying images with image-to-image translation. In addition, I investigated wherever adding a segmentation network together with a GAN will produce better output images than just a GAN. Produced solution is intended to be applicable to different datasets. The scope of this thesis looks into image transformation with weather images by first transforming a source image of some weather condition X to a target weather condition Y (for example, sunny to cloudy). Additionally, I'm using a segmentation model to select the pixels that are most related to weather conditions to add the pixels with weather-cues to the original image. Finally, producing a distinct attention model which defines how strongly the weather-cues from the segmentation model should be applied over the original image is attempted.

Answering RQ1 can help us in future image-to-image transformation projects to consider, whether training a distinct segmentation model and generating the segmentation masks for a dataset will be useful.

Answering RQ2 will clarify, if there is a significant difference between using a pre-trained model networks over the other tested ones. Although the segmentation model is important as a part of the GAN-pipeline, the knowledge RQ2 can generate is also applicable to segmentation problems without the GAN module.

1.4 Thesis organization

Chapter 2 describes the background in semantic segmentation, GANs and Image-to-Image transformation, which will be needed to produce the presented solutions in this thesis.

Chapter 3 introduces an image transform pipeline based loosely on WeatherGAN [17]. The pipeline includes two modules; GAN-based generative model and segmentation model. Additionally, chapter 3 also presents the collected results in the thesis by evaluating the proposed methods based on the common metrics.

Finally, chapter 4 concludes the thesis and proposes some suggestion for possible future works.

2 Background

2.1 Deep learning

Deep learning (DL) is a subcategory of machine learning. The term “deep” (network) being ambiguous as opposed to “shallow” (network), and there is no clear definition of how many layers a network needs to have to be “deep”.

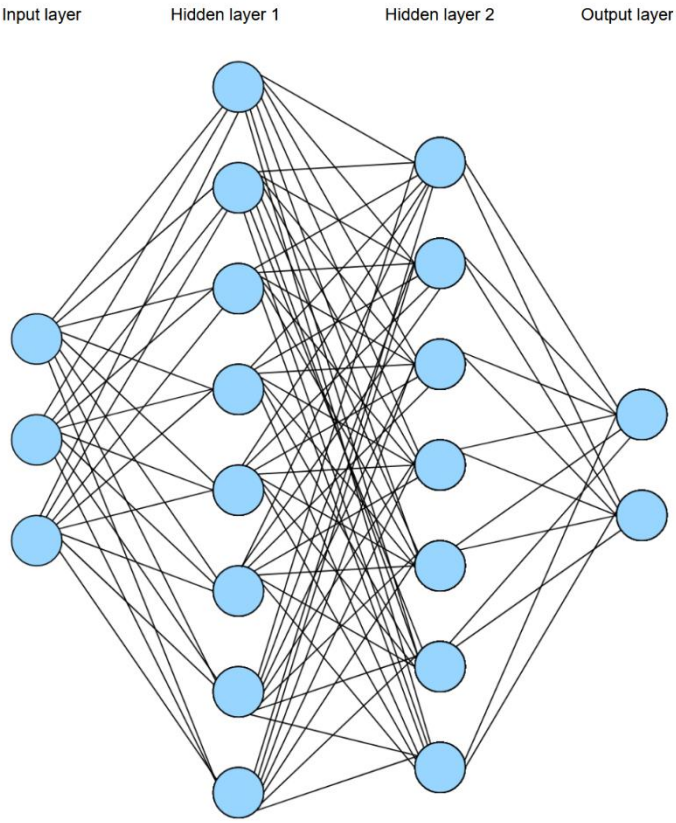

Figure 1: An example of fully-connected neural network with 3 input nodes, hidden layers with 8 and 7 nodes, and 2 output nodes.

A deep neural network layer consists of nodes that are connected to other layers’ nodes via weight connections. If every node of a layer connects to every node of the following layer, a network is fully-connected, as shown in Figure 1. The weights are learnable parameters, which adjust their values during network’s training phase. A cost function will be used to measure how far a neural network’s output, prediction, is from the real result. Based on the result of this cost function, a method known as backpropagation is the most common method to be applied to the network to optimize its weights.

How much the weights are updated depends on the difference between the prediction and target values, optimizer and learning rate used etc. Additionally, there are biases and activation functions with learnable parameters as well [18].

Hidden layers allow a deep neural networks' layer to specialize on specific tasks, such as image network having different layers extract features in certain resolutions. On the other hand, too many layers increase the complexity of a model and its training and interference time. Additionally, large number of layers might cause a model to overfit, where the model memorizes the data instead of being able to generalize to new data. Avoiding overfitting is one of the major reasons why models are trained and tested on different data, which allows one to see how a model performs on novel data.

2.2 Convolutional neural networks

Convolutional Neural Network (CNN) is highly efficient model for visual DL, as its architecture is capable of delegating working with different levels of details and patterns to specific layers. This reduces the number of connections required compared to a fully-connected network with multi-layer perceptrons, instead scanning the problem space with feature filters that map the space in how much certain features appear. CNN features scanned on first layers are often low-level features consisting of edges and other basic pattern shapes, and on later layers the features begin to be more specialized on certain details and distinguishing shapes.

Striding is a parameter of a CNN, that sets how much movement in an image kernel or filter window is present at a time, for example how many pixels to move along an axis. Related to striding is padding, that can add empty "pixels" to the edges of an image matrix to prevent striding from going over a corner, when the remaining pixels are not even with striding value. This prevents the kernel from visiting the center pixels more than the corner ones.

Pooling is a dimensionality reduction technique to downsample spatial (x and y) dimensions, which is done to a pooling window of specific size. The values inside each pooling window can be reduced by averaging the values (average pooling) or taking the largest value (max pooling). Pooling aids in reducing computational cost and overfitting of a model, as with less dimensions layers are forced to learn feature information of the data. As only either the average or largest value is carried over with pooling operation, it is non-invertible operation, but it's possible to estimate the original values from resulting pooling operation. To instead

increase the resolution of an image, upsampling with nearest neighbours can be used, where one value is copied to neighbouring pixels. Other methods to upsample include solutions such as linear or bi-linear interpolation and transposed convolution, which includes a learnable filter to learn the mappings from a lower resolution.

Convolution applies a learnable filter to the striding window, and transposed convolution is in practice an inverse of convolution, i.e. reversing the result image to the one it was before applying convolution filter [19]. Fully-strided convolution decreases the image size, while transposed increases it.

Figure 2 shows an example of CNN architecture, with two convolutional hidden layers each containing convolution and max pooling operations. The last hidden layer is a fully-connected layer, and output is two nodes producing predictions such as classification probabilities. Each convolution and max pooling operation gets its input as a filter of certain size.

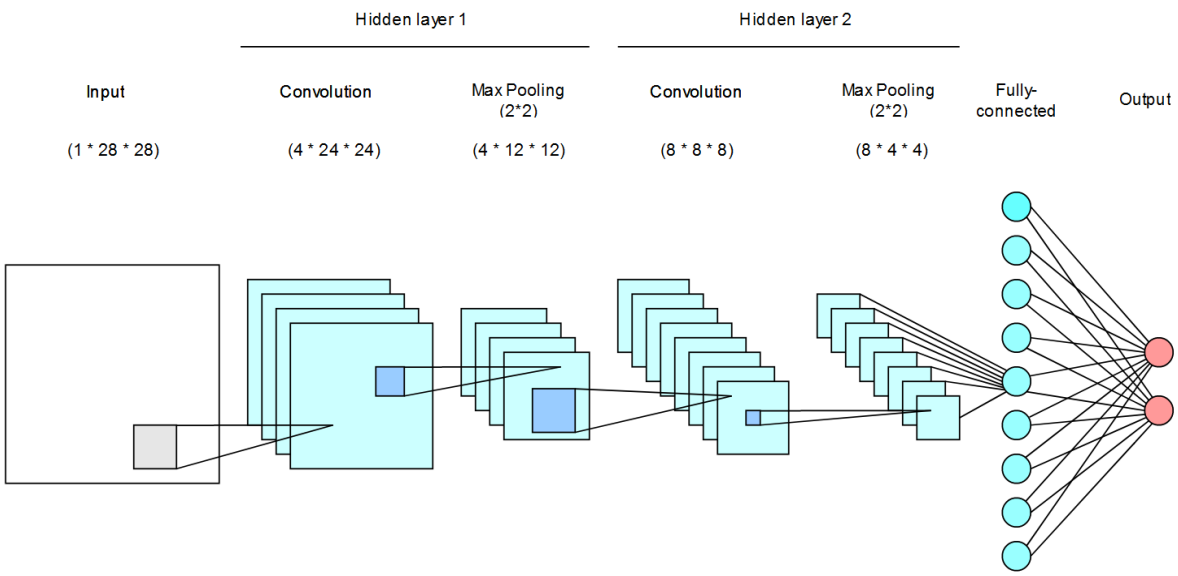

Figure 2: A basic architecture of a classification Convolutional Neural Network, with one-dimensional (black and white) input image, and two output nodes.

Full and transposed convolutions in CNNs make it possible for spatial down- and up-sampling operations, respectively, to be achieved. These operations handle the change in resolution and locations of patterns or features, that are mapped from image space to the

latent space. Striding is a useful alternative for usual pooling functions, as pooling is a static function but strided convolution has learnable parameters. Therefore striding can increase the task-specific adaptation of the neural network model at the expense of computation time in some cases [20].

Batch normalization is noted to be a useful addition to most CNN layers to increase the stability of training. Batch normalization is a normalization method to standardize the inputs inside each mini-batch (snapshot of training data used in the current round of gradient updates). However, it has been reported [21] that having a batch normalization layer for output layer of generator and input layer of discriminator can cause sample oscillation and model instability. This might have been caused by the two networks no longer sharing the same, independent data distribution.

Dropout layers are used to set a percentage of their randomly selected inputs to 0 during training. This helps by preventing some single weights from making the network too dependent on them, forcing the other weights to work with different combinations of the same layer weights. This reduces the chance that the network learns some exact feature values instead of generalizing to the whole dataset, so it helps to avoid overfitting.

2.3 Attention

Attention, as a method in neural networks and DL, is the process of noting what are the areas of interest in the neural network model's prediction, which features had the most influence on a prediction instead of the whole input (for example, a classification being decided to be of a certain class). Attention in computer vision works with the same analogy as the attention human eyesight has when focusing on a certain target, where the more distant or less significant targets are given lower weights. In CNNs, one can visualize the intermediate activation maps to see the high- or low-level features, depending on the layer. In addition to its applications to computer vision, attention has been used in natural language processing, for example in the bidirectional encoder representations from transformers models [22].

2.4 Semantic segmentation

Segmentation in computer vision is the process of partitioning an image to different subparts by every pixel of an image. This partitioning is done by applying some methodology, that can distinguish the parts from each other using various methods.

Semantic segmentation requires, that every processable unit of the segmentation object is labeled with the desired segmentation domain. Most commonly, this means an image that has an associated mask with every pixel labeled into segmentation categories.

Depending on the complexity of the object and the variation of the examples, a solution might be plausible with an algorithmic approach. One basic algorithmic segmentation method is thresholding [23], [24], in which an image is transformed to grayscale and pixels that exceed a threshold value are classified to a certain class, which might be coupled with flood-fill algorithm (expand selection in each direction, until encountered with a pixel that does not exceed the threshold) or other rules. Other algorithmic solutions include edge-based segmentation [25], [26], which detects changes of patterns when reaching the border of an area and the change's direction, and region-based detection [25] where an area is filled starting from certain seed location. In chapter 3.1 we review the proposed segmentation models in this thesis.

2.5 U-Net

U-Net [27] is an encoder-decoder type neural network architecture for segmentation, often used for medical imaging solutions. First part of U-Net has a contracting path structure that is often present with CNNs, in which the input feature resolution increases. U-Net combines contracting path structure with expansive path, where skip connections feed features from layers not directly before them. The benefit of these skip connections is that while using an encoder-decoder architecture that shrinks the number of features, more generalizable features might get overwhelmingly dominant weights, while the more dataset-specialized features might become irrelevant. However, with skip connections these more specialized features are also sent to the corresponding decoder layer from the encoder layer.

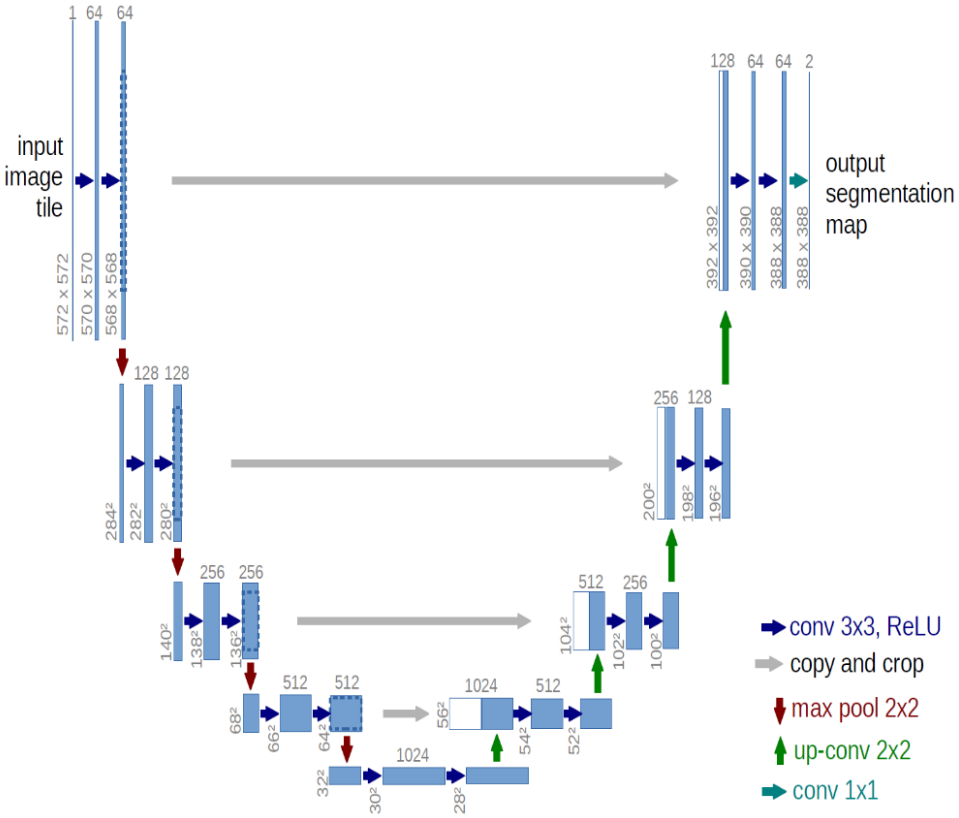

Figure 3: U-Net architecture [28].

The contracting and expansive paths are often similar to each other in structure, making the model resemble a U-shape. The segmentation is done with subparts of the input image, tiles, and these tiles require a border area around them larger than themselves to predict the tile area. For tiles whose surrounding border area is not completely in the input image, the missing area is mirrored from the existing border area.

U-Net was initially designed for biomedical image segmentation, where availability of data is often low [27]. In Figure 3, an example U-Net architecture is shown, with 572*572*1 input and 388*388*2 output size.

2.6 Data augmentation

Data augmentation [29], [30] is the process of generating more data examples by learning probability distribution of some data and adding variations to the data in some way. This includes cases such as removing sensitive information which allows the data to be used more freely and generating more data for training to learn more about the patterns in it or to balance an imbalanced dataset. Data augmentation operations such as rotating, cropping and changing colors are also commonly used in pre-processing steps of machine learning, which

help in allowing the model to be trained on a more representative and diverse dataset and be less prone to overfitting.

The aim of this thesis to changing the weather condition of an image but leaving the rest of the image unchanged is one example of conditional data augmentation. Uncontrolled generation, where a model will generate random data from the learned probability distribution is also a possibility [31].

2.7 Generative adversarial networks

Generative adversarial networks (GANs) [4] are most often unsupervised deep machine learning algorithms, which attempt to generate novel and diverse data, "fakes", which ideally is indistinguishable from real data. The basic structure of GANs consists of two opposing networks:

- **Generator network**, which learns to simulate the underlying patterns in the real data, and
- **Discriminator** (i.e., classifier) network, to rate both the generator's output and real data.

In Figure 4, the architecture of a basic GAN model is shown.

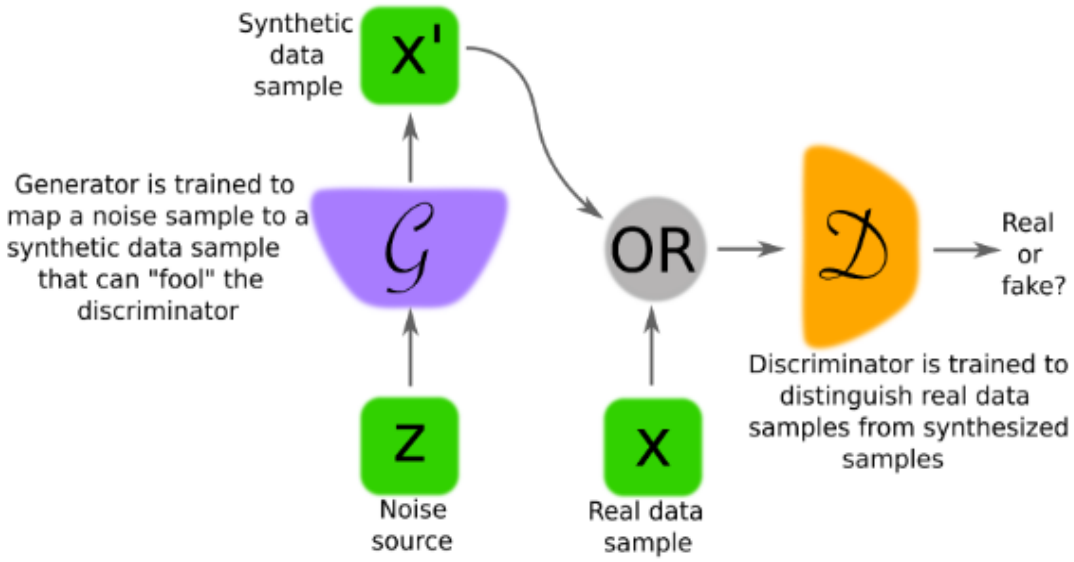

Figure 4: GAN logic [10].

These two networks are trained in succession, and the Min-Max game they are playing is the mainstay of this algorithm; the generator is trying to pass all of the fake images it has produced as reals to the discriminator. Disciminator receives as input both the generator's images and real images, and its goal is to rate real images with high probability (ideally 1) of being real, and fake images with low probability (ideally 0). Generator network's input space, which called latent space, will start at random noise, and based on discriminator's ratings on generator's output images the latent space will attempt to map combinations of features that would represent real data. An analogue about the generator being a person making fake artwork and discriminator being an art critic trying to distinguish real art from fake is very representative of GAN logic.

GANs fall into the category of implicit density models, which means that they do not find the data distribution, but instead learn a method to map some points of data into the latent space and build a model or simulation of the problem based on the features these examples reveal. Opposed to implicit density models are explicit density models, that try to find the exact data distribution, where some other generative models belong to.

Adversarity in machine learning applies as a min-max problem, where the minimizer (discriminator), who works as a classifier, tries to find parameters that minimizes the cost of classifying. The maximizer (generator) then tries to find inputs that maximize the classifying cost. Google's AlphaGo used an adversarial algorithm when it played different versions of itself against each other, successfully improving from it [32].

A generator represents mapping from noise space to data space with $G(z, \theta_g)$, where G is a differentiable function of neural network with parameters θ_g.

The original value function of a GAN is:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)} log D(x) + E_{z\sim p_g(z)} log\left(1 - D\big(G(z)\big)\right) \quad (1)$$

where $p_{data}(x)$ is the probability density function of random vectors x from the real data space, and $p_g(z)$ is the probability distribution of vectors from the generator's latent space.

Equation (1) is known as Binary Cross-Entropy Loss. The optimal case is $p_g(z) = p_{data}(x)$, when the discriminator outputs a probability of 0.5 for every data sample it receives, meaning it cannot tell fakes from reals [4]. This state is known as Nash equilibrium,

a game theory phenomenon where no player can choose more optimal strategy in response to their opponent's most optimal strategy [11]. In this thesis we are concentrating on deep convolutional neural network GANs (DCGAN), but GANs themselves can be implemented with any differentiable system, that can map datapoints between spaces. DCGANs use CNN architecture, which is ideal for image data, whereas original idea of neural network GANs [4] use Fully-connected neural networks consisting of multi-layer perceptrons.

2.8 Adaptive moment estimation (Adam)

DCGANs often use Adam optimizer [33], as opposed to first GANs that used Stochastic Gradient Descent with Momentum [34], a variant of Stochastic Gradient Descent (SGD) [35]. Adam uses adaptive, independent learning rate for each parameter. The moment estimation part of Adam comes from the fact it uses estimations of first and second moments (mean and variance) of the gradients.

Update in a network parameter θ_t with Adam can be written as

$$\mathbf{m}_t = \frac{\beta_1 * m_{t-1} + (1-\beta_1) * g_t}{(1-\beta_{1t}^t)} \tag{2}$$

$$\mathbf{v}_t = \frac{\beta_2 * v_{t-1} + (1-\beta_2) * g_t^2}{(1-\beta_2^t)} \tag{3}$$

$$\theta_t = \theta_{t-1} - \frac{\alpha * \boldsymbol{m}_t}{(\boldsymbol{v}_t + \epsilon)} \tag{4}$$

where α is learning rate, t is timestep, initially 0, m is first moment, mean, v is second moment, variance, β_1 and β_2 are the independent decay parameters for m and v (default value for $\beta_1 = 0.9, \beta_2 = 0.999$), ϵ is a small utility variable to prevent division by zero error (default value 10^{-8}), g_t is gradients at some time t, $\boldsymbol{m}_t$ and $\mathbf{v}_t$ are bias corrected with a decaying multiplier, so that at the start of the training their initialization to 0's wouldn't affect training too much.

All in all, Adam has multiple benefits compared to many other optimizers, including fast convergence, generalizes well for different datasets, is resistant to noise and needs little manual tuning [33].

As GANs have two networks to be trained simultaneously, it can be possible that the adaptability and stability of Adam comes in use. SGD with momentum can sometimes generalize better than Adam, while Adam tends to converge faster and better in extreme

regions such as saddle points of a function [36]. However, as it's hard to keep GAN training converging, it might be better to make use of Adam's ability to converge faster and more evenly, and use other methods (presented later) to ensure the quality of generations, that for example ensure that the generated images vary from each other.

In [37] they show Adam also has a property of performing better with cycle problem than SGD. Cycle problem rises from Min-Max game, where the non-continuous nature of two-player GAN-training prevents Nash equilibrium from being reached. Instead, the convergences of distributions p_g and p_{data} cycle a certain distance away from the equilibrium. Adam seems to perform well in cycle problem thanks to its historical averaging on both first and second moments and the decay parameters β_1, β_2. β_1 seems to help to deal with noise that spawns from mini-batch training, but a trade-off value for it is needed, as too large value will cause divergence. A well-chosen β_2 helps in approaching equilibrium with the second moment. As the training advances and the cycle radius shrinks, the window at which historical gradients are averaged also shrinks, which requires one to change the parameter β_2 at the end of training. Later introduced FID could help to solve cycle problem by setting a single equilibrium point, but it requires extra work if it's to be done during training at every gradient update. Good performance of Adam with cycle problem was seen to be coincidental. There have been some new optimization functions that are applicable to DCGANs, some of which are variations of Adam, including AdamW [38] and AdaBelief [39].

I think that because as Adam has been around for some time and been widely used, some methods and architectures still in use might have been planned to work well with it. Since some areas of DL are still rather experimental, perhaps the currently used "rules of thumb" (like architecture choices or hyperparameter values) are also found to be working well when using Adam. In addition, Adam's historical averaging and decay parameter can help to solve cycle problem. Adam can be seen as a safe choice in reaching convergence as it's tested to work with many cases, but newer optimizers like AdaBelief or AdamW might perform better in specific cases.

2.9 Training GANs

Training a GAN can be very unstable (examples in the following subchapter 2.10 "Problems in training GANs"), and careful management is needed as the two networks must always be

around the same level of convergence for the models to improve. Earlier mentioned Nash equilibrium is also one limitation that affects convergence.

During training the generator and the discriminator alternate turns, where one model interferes the other, frozen model, updating its weights. After the training is finished, only the generator is needed to generate new data. However, the discriminator might also still potentially have its uses, for example it can be used to rate the initial dataset of real examples. The real examples with the lowest score could then be removed from the dataset. These images might be useful to enrich the dataset, but in large datasets some images might be corrupted or not representative of the subject. The last layer of the discriminator can also be removed, and its last hidden layer used for feature extraction.

2.9.1 Problems in training GANs

There is a multitude of possible problems in the data examples that GANs generate, of which some are listed in this chapter. In the next chapter I introduce some of the most common solutions to these problems.

2.9.1.1 Mode collapse

Mode collapse occurs, when the generator gets stuck in generating the same output that the discriminator rates as very probable of being a real data example. If the discriminator is simple, it might never detect this kind of behaviour from the generator's part, and improvement of the GAN halts.

2.9.1.2 No convergence

Sometimes one or both networks fail to capture meaningful representation of the data. This can be caused by chosen model architecture or hyperparameters, data, loss function, noise or multiple other reasons.

2.9.1.3 Mismatch in convergence levels

If the training isn't stable one of the models (usually the discriminator) can get significantly better in detecting the fakes, so that the generator has no chance of getting a probability score much higher than zero for its output, therefore receiving no feedback on what images were on the right track and in what direction to update.

2.9.2 Solutions to problems of training GANs

2.9.2.1 Mini-batch discrimination

Mini-batch discrimination checks the distance between the current generated image and mini-batch of n images that it's a part of. Feature vectors **f** of every mini-batch image are collected to a matrix **M**. Then, calculate L1 norm between each *m* rows for current $\boldsymbol{M_i}$ and every other (*n*-1) feature matrix. L1 is the sum of absolute differences:

$$L1 = \sum_{k=1}^{n} |x_k - y_k| \tag{5}$$

Similarity $o(x_i)$ of input x_i is calculated with row-by-row L1 difference for each image, and taking a negative exponent of it:

$$\mathrm{o}(x_i) = \sum_{r=1}^{m} \sum_{j=1}^{n-1} \exp\left(-||M_{i,r} - M_{j,r}||_{L1}\right) \tag{6}$$

This output $o(x_i)$ is then concatenated with the feature vector $\mathbf{f(x_i)}$ which was the input of this minibatch layer. Fake images from the generator and real images from the training data are treated separately, and each mini-batch only contains images from one of them. Final layer of the discriminator will return probability of the input image being real like before, but now having this additional information about similarity. This helps to detect model collapse, as similarity $o(x_i)$ affects the final output the discriminator gives, which only affects the generator (as the real training images do not receive any information) [40].

2.9.2.2 Historical averaging (parameters)

Historical averaging of parameters can be used to check that change in model's parameter values is not too high to make the training unstable, which aids in reaching convergence. This is calculated with equation 7:

$$||\theta - \frac{1}{t}\sum_{i=1}^{t} \theta_i||^2 \tag{7}$$

which is a L2 difference between current parameter θ and t earlier parameters $\boldsymbol{\theta_i}$. This term is applied to discriminator's and generator's cost in the value function. Historical averaging together with mini-batch discrimination can help in keeping generator around a “golden mean”, preventing its output from getting too homo- or heterogeneous [40].

2.9.2.3 One-sided label smoothing

Set the target of discriminator's output probability to 0.9 instead of 1. Now the discriminator will never return 0 as a probability of image being real, therefore always giving the generator something to update on. Helps to solve discriminator getting too far ahead of the generator. It might also be useful to add noise to the real (and fake) images that the discriminator gets as an input during training, so that the boundary between real and fake images gets narrower. This way the imperfections (such as out-of-place colored blob artifacts) that the generator outputs in the images affect the predictions less, allowing it to concentrate on other features [40]–[42].

2.9.2.4 Rectified Linear Units vs Leaky Rectified Linear Units

Both mismatch in training and no convergence can be caused by vanishing gradients. Vanishing gradients is the phenomenon when action functions used start returning very small gradients, for example when original rectified linear unit (ReLU) [43], [44] activation function goes to negative values. This can lead it becoming "dead ReLU" and providing 0 gradient for the rest of the run [45].

ReLU can be presented with equation 8:

$$f(x) = \begin{cases} 0, when\ x < 0 \\ x, when\ x \geq 0 \end{cases} \tag{8}$$

, where x is the input value to ReLU.

LeakyReLU [46] can be presented with the equation 9:

$$f(x) = \begin{cases} \propto x, when\ x < 0 \\ x, when\ x \geq 0 \end{cases} \tag{9}$$

, where $\propto$ is a multiplier for the negative slope. Default value varies on implementations, but on Keras the default is 0.3.

Implementing LeakyReLUs instead of ReLUs into discriminator can reduce the vanishing gradient problem [46]–[48]. LeakyReLUs will return non-zero value even when negative, making sure that they will contribute at least minimally to training, as visualized in Figure 5. It was shown in [21] that LeakyReLUs in discriminator gave much better performance than regular ReLUs. generator however was not changed from using normal ReLUs. At least

computation-wise computing LeakyReLU activation is slightly more expensive than ReLU, and performance was not seen to be noticeably better in generator which had LeakyReLU's.

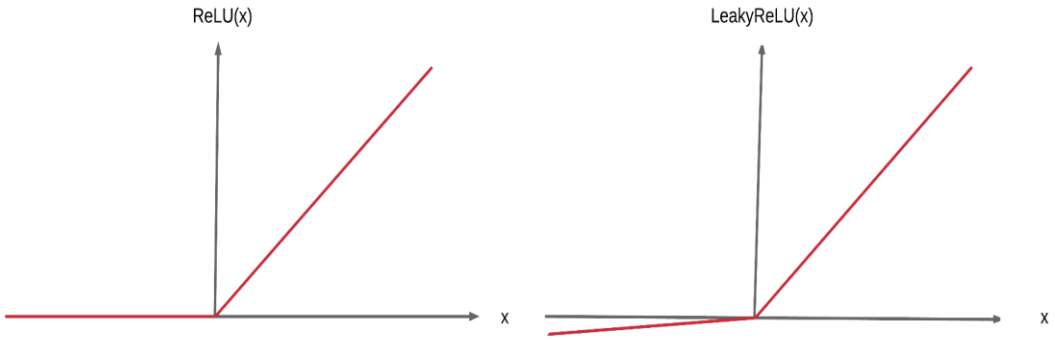

Figure 5: ReLU and LeakyReLU activation function values in relation to *x* [49].

2.9.2.5 Other solutions

One can also try to use different initial learning rate for discriminator optimizer than for the generator, to decrease the chance of training collapse. Updating the discriminator twice for every generator update was noticed with BigGAN to yield good results when one of the networks fell behind. However, this is also one tunable parameter, and using differing rates can also lead to instability [50].

Beginning the training from lower resolution than the final target resolution, to first train the networks on crude features, and slowly increase the resolution after certain number of batches. This can be achieved by downscaling the training data and decreasing the neurons at generator's output and discriminator's input. The benefit is to learn high-level features and patterns of the dataset first. Starting training from lower resolution can also help in convergence or increasing the quality of generated images.

2.10 Evaluating GANs

As GAN is usually a semi-supervised learning method (unsupervised except for the supervised loss from the discriminator), it is rather difficult to determine if the generator outputs realistic images. The discriminator might not be enough to determine, if a generated image would pass as real or not, and it is affected by bias from the training data which might not represent the whole problem space. Human evaluation on the other hand is expensive.

Some solutions for this have been presented, including:

2.10.1 Wasserstein GAN

Instead of using a discriminator to return a probability of an image being real, Wasserstein GAN [51] uses a critic (marked still as D) to rate the image's "quality". Wasserstein-1 distance, also known as earth mover's distance, measures the difference between two probability distributions (here fake and real), and tries to find the minimum cost to make these two distributions identical. The original GAN value function (equation 1) presented earlier used binary cross-entropy loss. Wasserstein-1, which is used as the loss function, looks very similar with its min-max game:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)} D(x) + E_{z\sim p_g(z)} \Big(1 - D\big(G(z)\big)\Big) \tag{10}$$

But it does not have logarithms unlike binary cross-entropy. This means that whereas BCE had to limit its output to be in relation to a ground truth probability, Wasserstein loss does not. It gives the benefit of easing vanishing gradient and mode collapse problems, as the generator now gets useful gradients also from very good or bad images. The loss function will also no longer contain flat areas like with BCE, which could lead to vanishing gradients problem.

Wasserstein loss has a requirement of having the critic to be 1-Lipschthiz continuous; this means that at any given point, gradient of its norm can be at most 1. This requirement ensures that the function grows linearly and that the output is continuous and differentiable, which are requirements for Wasserstein-1 distance to be applicable. This requirement is necessary, as we don't have logarithms in the loss function anymore.

We can make sure that 1-Lipschthiz continuity holds with either weight clipping or gradient penalty. With weight clipping, all gradient values that exceed maximum or minimum limit are set to the exceeded limit value.

With gradient penalty, real and fake images are sampled to form an interpolation image *y* by calculating $\epsilon x + (1-\epsilon)G(z)$, where ϵ is a random probability [0, 1]. This sampled image is used to calculate a regularization term, which is added to the loss function:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)} D(x) + E_{z\sim p_g(z)} \Big(1 - D\big(G(z)\big)\Big) + \lambda(\|g_y D(y)-1\|)^2 \tag{11}$$

where λ is an optimizable hyperparameter (10 was found to be working in the paper), y is an image formed from a sampled random pair of real and fake images, and g_y is a gradient associated with $\boldsymbol{y}$.

Gradient penalty was seen in the paper to be able to keep the norm of critic's gradient below or at 1 at "almost everywhere". With gradient clipping and enforcing a hard limit to gradient, there were problems in finding a working hyperparameters for correct values, and gradient penalty was seen to work generally better. Additionally, Wasserstein GAN uses RMSProp instead of Adam that the original GANs use [51], [52].

2.10.2 Fréchet inception distance

Fréchet inception distance (FID) [11] uses a squared Wasserstein-1 distance (making it Wasserstein-2) to input generated images to a trained CNN, usually Inception v3 network. This model gives useful features and patterns on how human recognition of certain objects might be done. Last output layer of Inception v3 network is removed in FID to gain access to usually the final hidden feature layer, which will output a data distribution of the generated images, which will be used to compare against the real images to get an FID score. The layer used for feature extraction does not necessarily need to be the last hidden layer, but that layer usually has the most finalized information about different patterns. To access cruder or less-category specific features, one can also use an intermediate hidden layer for the feature extraction.

With feature extraction, the aim is to minimize the amount of features the data contains, while retaining the representational value of the data. By using FID, the evaluation will not be pixel-by-pixel comparison, but a feature-to-feature level comparison, giving the benefit of pixels being shifted, image being rotated etc. data augmentations not having any major effect on the results. Additionally, high-level features about the most prominent attributes of the dataset will be compared between images, instead of single pixels. By using a pretrained network such as Inception v3, all the different features learned from the wide range of data the network has been trained on can be used for feature extraction.

FID is used in some state-of-the-art DCGANs, including StyleGANv1 and v2. Inception v3 network is trained using a subset of ImageNet, a dataset of 15 million images labeled into 1000 categories. 1.3 million of images from ImageNet dataset were used in training and 50000 in evaluation, with some of the images including bounding boxes for low-level

segmentation cues. This amount of data can help in learning general features of different objects and patterns and be generalized to different domains.

It appears that the wide range of data Inception v3 has, relatively low parameter count (and therefore low computational cost and less prone to overfitting) and its good accuracy makes it a good starting point for most general tasks. However, this might get problematic in some cases when these general features are not similar to the features in the training dataset. If this kind of problems arise, it might be beneficial to train a different comparison model or transfer learning Inception v3 network to the problem space [11].

2.11 Conditional GANs

In conditional GANs (cGAN) [53] generator and discriminator include additional labels c, such as categories, in addition to normal GAN inputs to bring semantic information. For example, the commonly used MNIST-digit challenge generates a random number with a normal DCGAN, but with a conditional DCGAN with classes 0-9, one could choose the digit they want to generate, or with styles what kind of font it uses etc. The value function of conditional GAN is the same as original GAN except it has per-category labels:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)} log D(x|c) + E_{z\sim p_g(z)} log\left(1 - D\big(G(z|c)\big)\right) \quad (12)$$

where *c* is the additional information label.

Figure 6 shows the logic for a cGAN.

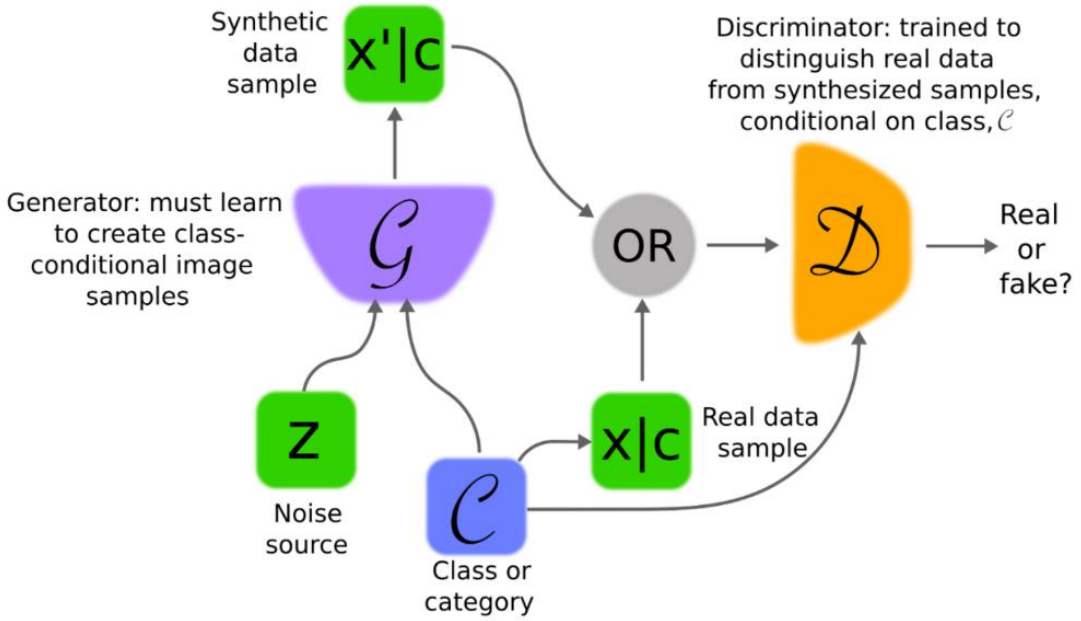

Figure 6: Conditional GAN logic [10].

Apart from the losses in the original GANs, where one just wants to minimize the distance between the generator and discriminator, Conditional GANs need to consider both the image's validity in the context space and its correct translation. For the second task people have sometimes added a L2 loss function, which is just the sum of squared differences between the real and fakes:

$$L2 = \sum_{i=1}^{n} (Real - Fake)^2 \tag{13}$$

where n is the training dataset size.

2.12 Image-to-image translation

Image-to-Image task is the translation of an image from source domain X to a target domain Y. It's harder to make generalize than uncontrolled DCGANs, and a major problem was that they often require specifically crafted loss functions or neural network architectures for every task [10].

2.12.1 Pix2Pix

Pix2pix [54] is a conditional GAN for transforming one image (source) based on a conditional second image (target), which allows the resulting image to capture the desired transformation of the conditional image, which can be Monet-style transformation of a photograph, horse getting a zebra's skin etc.

Authors of Pix2Pix further change the L2 loss to L1 loss, which is implemented in this context as

$$\mathcal{L}_{L1}(G) = E_{x,y,z}\left[\left|\left|c - G(x,z)\right|\right|_{L1}\right] \tag{14}$$

, as L2 loss resulted in more blurry generated images [6]. λ is used as a hyperparameter multiplier to optimize the ratio of L1 loss (generated image's plausibility as a transformation of the source image) compared to adversarial loss of the discriminator (generated image's plausibility in the target space). In the Pix2Pix paper 100 was found to be a good value for λ. Conditional adversarial loss is shown as:

$$\mathcal{L}_{cGAN}(G,D) = E_{x,y}[log\ D(x,y)] + E_{x,z}[\log(1 - D(x,G(x,z))] \tag{15}$$

G then trains on using a combined loss function of conditional adversarial loss between discriminator's predictions and its output and L1 multiplied with λ to measure the pixel difference between the generated image with transformation and expected real image:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda\mathcal{L}_{L1}(G) \tag{16}$$

Pix2Pix is intended to be a general algorithm for different image-to-image tasks. Unlike normal GANs, the generator does not take its input from latent space. Instead, the noise is drawn from the use of dropout layers in the architecture, that disregard random outputs of a layer at each forward pass. Pix2Pix generator uses the U-Net model architecture, which allows up- and downsampling by allowing layers to connect to layers not directly next to them.

The discriminator on the other hand uses PatchGAN architecture, which rates the realness of NxN pixel patches of the input image, instead of the whole image. 70x70 patch size was deemed to be very applicable to different tasks in the related paper. PatchGAN receives two inputs; the source image and the target image of said image, and the task is to predict if the

target image is a real transformation of the source image. The output is a feature map of predictions, which can be averaged to get a complete, single prediction of the image [54].

With the above features, Pix2Pix is able to generalize quite well to use the same architecture and loss function for different image-transfer tasks (satellite to map, day to night, labels to façade or street, black-and-white to color). With some earlier Conditional GANs, such as next frame prediction [50], image prediction from sparse annotations [55] and clothing transfer [56] it was necessary to manually craft an application-specific loss function and architecture.

2.12.2 Cycle-consistent adversarial networks

Cycle-consistent adversarial network (CycleGAN) [6] is a conditional, convolutional GAN for transferring an image from a source domain *X* to a target domain *Y*. The main attraction of CycleGAN is that it does not need paired training data, that is the training images don't need to be of the exact same subject in domains *X* and *Y*. Instead, it can learn the most prevalent features in the two dataset domains [6]. It works in a two-way fashion with two generators, with the generator *G* transferring an input image of source domain *X* to target domain *Y*, and the generator *F* transferring the generator *G*'s domain *Y* transformation back to the original domain *X*. Both generators *G* and *F* are penalized for difference between the generator *G*'s input image and generator *F*'s output image. With this, the generators are required to retain some aspects of the input image, therefore making sure that objects that are not points of interest do not change too much in the transformation. This penalization is called Cycle consistency loss;

$$\mathcal{L}_{Cyc}(G,F) = E_{x\sim p_{data}(x)}||F\big(G(x)\big) - x||_{L1} + E_{y\sim p_{data}(y)}||G\big(F(y)\big) - y||_{L1} \quad (17)$$

CycleGAN sometimes produces problematic results by transforming also un-related aspects of the image; this can be seen with the original horse ⟷ zebra translation in the original CycleGAN repo, where domain change also changed the hue of the whole background. Although when weather changes between sunny and cloudy the lightning and visibility of the ground might change also, I noticed this effect to be too prevalent by on every surface. This is why an added segmentation model might improve the results to detect the area, where weather cues are present and only apply the transformation to that area.

2.12.3 CycleGAN with attention

In [57], they proposed an modification of CycleGAN with added attention module, which is combined with the output of the generator with element-wise addition to form fake images.

2.12.4 WeatherGAN

WeatherGAN [17] is a conditional GAN introduced by Xuelong Li, Kai Kou and Bin Zhao in the paper “Weather GAN: Multi-Domain Weather Translation Using Generative Adversarial Networks”. Its goal is to design an image-to-image transformation method for weather images that can be generalized to different domains. This goal is achieved by having a generator, segmentation and attention modules. Attention module focuses on recognizing areas of interest and balancing different parts of the image.

In this thesis I implement loosely some ideas introduced in WeatherGAN paper, but as the code is not publicly available and the paper does not go very deep into details, I take the concept of coupling segmentation and attention models with an img2img-transformation GAN, as well as use a part of the dataset the authors presented.

3 Experimental setups and results

In this chapter, I introduce the libraries, models and data that are used for implementation in my thesis. The code used for this thesis can be found on GitHub [1].

I run my codes in a Jupyter notebook by using the following libraries: TensorFlow, an open-sourced machine learning library developed by Google and Keras API to construct, train and run the models.

Additionally, I use Numpy, which is a package for scientific computing, including inbuilt mathematical functions, array objects especially for multi-dimensional array processing and operations for linear algebra. In this thesis, Numpy is used for manipulating the datasets in both preprocessing and postprocessing steps.

To process the segmentation masks, which are in MatLab format, SciPy library's loadmat-function is used. Additionally, OpenCV, Pillow and matplotlib's pyplot libraries are used for visualization.

3.1 Segmentation

I used U-Net as the baseline segmentation model with ReLU and maxpooling layers. The main reason for choosing U-Net is that the model architecture has been shown to perform well for semantic segmentation [27], [58], [59]. There are multiple existing segmentation models, so different model options are tested to see which one performs best for the task of this thesis. The U-Net model is constructed from backbone encoder and Pix2pix decoder parts, where the decoder's weights are frozen to their pre-trained values. For the encoder part we included VGG19 model, as it was used in the WeatherGAN paper, as well as VGG16, which VGG19 is an updated version of. In addition, we included DenseNet 121, 169 and 201 models, as they were also able to adapt well into different datasets. These networks include a high number of network layers with pre-trained weights, of which one can choose a subpart of layers to use.

[1] https://github.com/rajuranpe/WeatherGAN-Master-s-Thesis

3.1.1 VGG

Visual Geometry Group (VGG) has developed a deep CNN, VGG19, that has performed well on ImageNet dataset. VGG16, an older version, is also tested. The full layer listing of VGG19 is shown in Figure 7, taken from plot_model()-function of Keras.

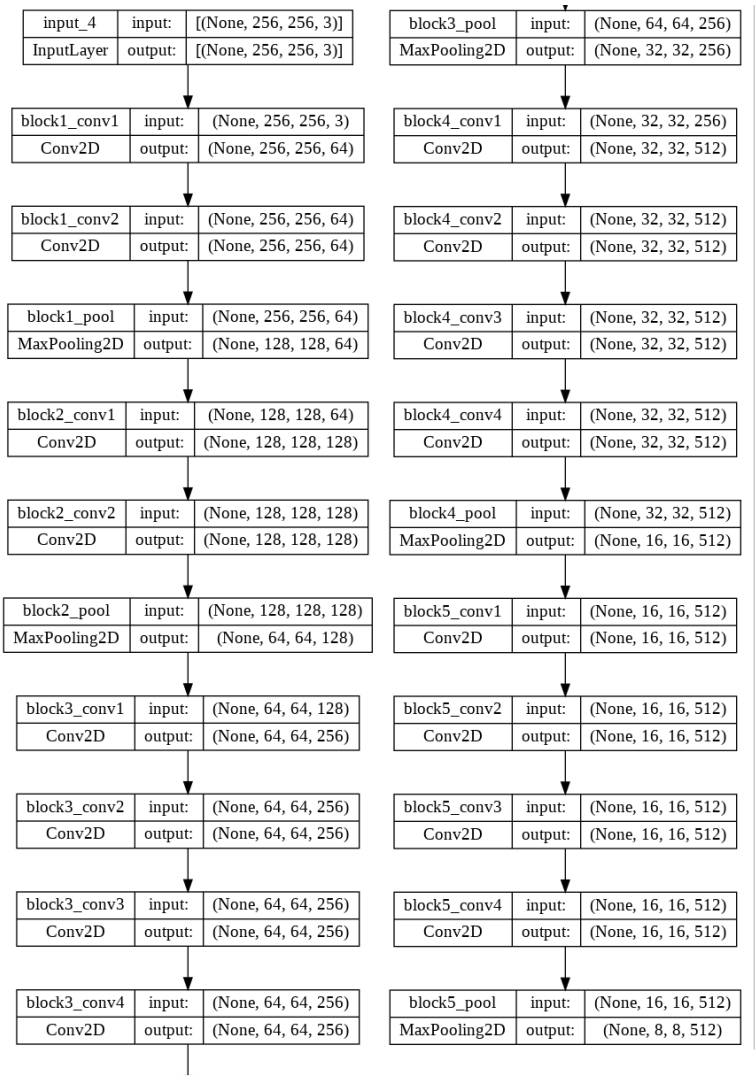

Figure 7: VGG19 full layer listing.

3.1.2 DenseNet

DenseNet is a type of neural network, that has feature maps connecting to all lower layers, not just the next one. The layers are also connected to layers not directly above or under them, making the number of connections higher than for example in ResNet [60], [61].

3.2 Generative adversarial network

To produce a generator model, the CycleGAN model is trained, because it has been shown to perform well for an array of different datasets, such as removing raindrops [14], damaged-undamaged structures [15] and aerial-satellite map translations [16]. This network is described in section 2.13.2 with more detail.

3.3 Dataset

The data used in this thesis is a part of "Multitask Weather" [62] dataset. The data consists of weather images in two categories "Sunny" or "Cloudy", and corresponding segmentation masks. Figure 8 shows example image and mask of these both categories.

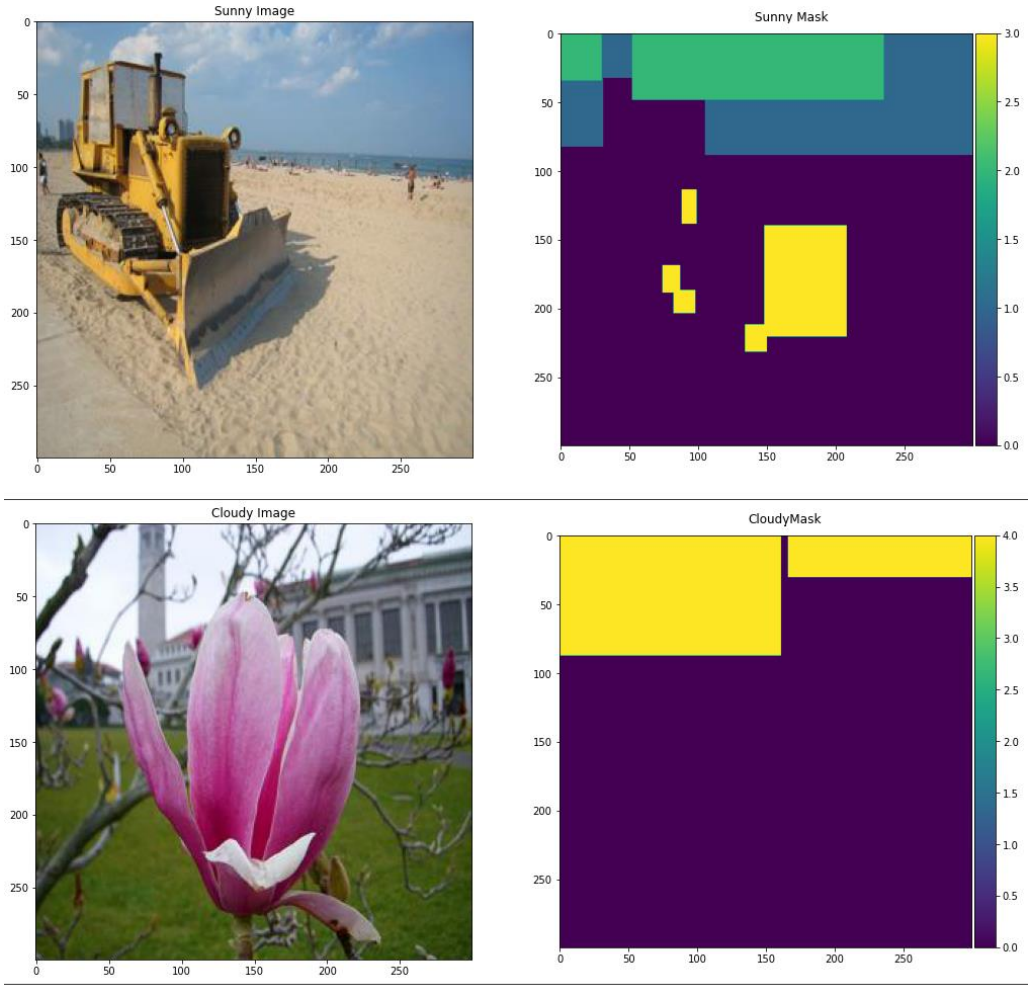

Figure 8: Data examples of Sunny and Cloudy category image and mask.

In the related paper [62] the dataset is mentioned to also include masks and images for Snowy, Rainy and Foggy categories, but those were not in available in the dataset provided

on the project's GitHub page [2]. The resolution of images and masks (as a pixel in the mask corresponds a pixel in the image) is 300*300*3 (Width * Height * (RGB color)). The segmentation masks have each pixel labeled with one category 0-5, whose meanings are explained in the Table 1.

Table 1: Segmentation label mask categories.

Label	Meaning
0	No category/ground
1	Sunny sky
2	Sunny sky clouds
3	Shadow
4	Cloudy sky
5	Clear sky

Segmentation masks are labeled manually by humans, which requires a large amount of work. However, the masks are not pixel-by-pixel accurate, instead marking the general areas of the image with corresponding or most prevalent label with bounding boxes. This means that some edges and details are not labeled correctly, which is the cause that the evaluation of the segmentation models is not completely accurate. The "Multitask Weather" dataset is used to train the segmentation network to learn the segmentation masks' correspondence to real weather images. Additionally, the real weather images without the segmentation masks are used for training the GAN in order to transform the domains cloudy to sunny and sunny to cloudy.

3.4 Implementation

To conclude, the pipeline consists of two parts: initial generator and segmentation modules, and additionally a discriminator to rate the generated images automatically during training.

In Figure 9 the pipeline implemented in this thesis is shown. The pipeline takes an input image, which is fed to a CycleGAN model, which transforms the image from one weather domain to another (Sunny ⟷ Cloudy). Additionally, the input image is fed to the U-Net-segmentation network to predict a segmentation mask. The transformed image and segmentation mask are then multiplied together with the input image to produce the final output image. In the final output image, the area in transformed image which intersects with

[2] https://github.com/wzgwzg/Multitask_Weather

the segmentation mask is applied over the input image, the rest remaining the same as the input.

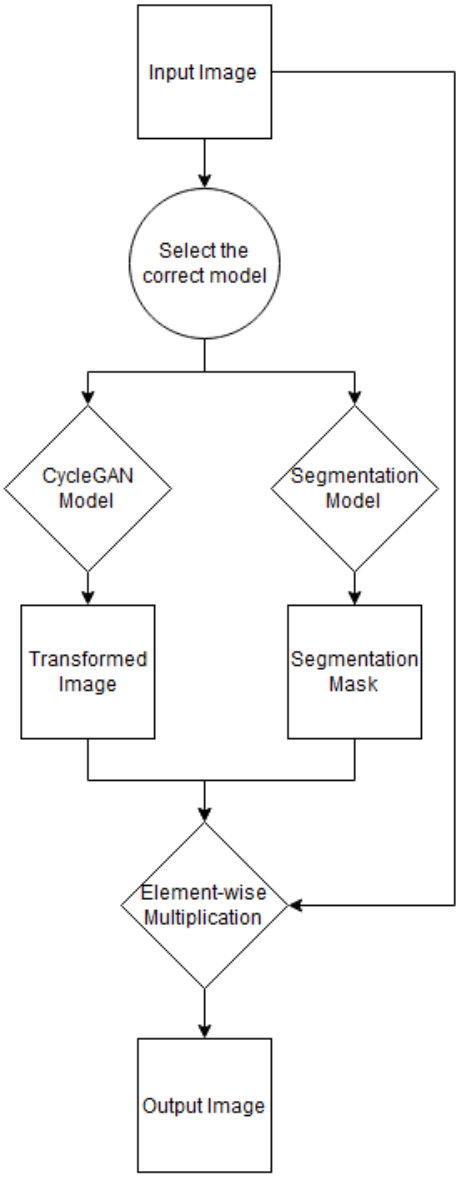

Figure 9: The proposed flowchart for implementation in this thesis.

3.4.1 Preprocessing

The dataset, “Multitask Weather”, consists of 10000 RGB images with size 300*300 and corresponding masks with size 300*300*1 (MATLAB files).

The dataset is divided into train and test datasets which consist 8000 and 2000 data examples, respectively. Then all images and masks in the dataset are resized into 256*256. The images and masks are normalized to -1, 1. Figure 10 shows a snapshot of code for reading the dataset images and segmentation masks.

```
images = []
masks = []

image_index = 0

reader = open(txtfile)
    item = i.split()

    img = cv2.imread(dir_img + item[0])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    images.append(tf.convert_to_tensor(img))

    maskmat = sio.loadmat(dir_mat + item[1])
    mask = tf.convert_to_tensor(maskmat["seg_mask"])
    masks.append(mask)

    image_index += 1

reader.close()
```

Figure 10: The code for reading and splitting the data.

The GAN does not need the segmentation masks, so I only added the images to their respective categories “sunny” or “cloudy”.

The input and output dimensions were chosen to be 256*256*3 to better use the U-Net model, as choosing a power-of-two resolution appeared to allow easier modification, but which could also have been solved with padding. It was decided that finding the right paddings would make testing different model configurations too time-consuming, since the results were good nonetheless and the image quality did not drop too much.

In Table 2, data augmentation operations applied to the segmentation data (images and corresponding masks) are presented. It’s imperative that operations such as rotation, which alter the disposition of an image are also applied to the segmentation mask, lest the correlation between the image and the mask would be lost. Operations that only affect style

or color balance of the image, such as hue or brightness, are not applied to the mask, as their benefit is in increasing the data augmentation of the dataset.

Table 2: Data augmentation methods for segmentation:

Operation	**Tensorflow (tf.) Function applied to data example x**	**Applied to**
Rotation	tf.image.rot90(x)	Image and mask
Crop	tf.image.central_crop(tf.image.resize(x, (256,256)), 0.6)	Image and mask
Horizontal Flip	tf.image.flip_left_right(x)	Image and mask
Vertical Flip	tf.image.flip_up_down(x)	Image and mask
Brightness	tf.image.adjust_brightness(x, 0.1)	Image
Gamma	tf.image.adjust_gamma(x, 0.1)	Image
Hue	tf.image.adjust_hue(x, -0.1)	Image

In Table 3, data augmentation operations applied to the GAN data (images) before training with CycleGAN are presented.

Table 3: Data augmentation methods for CycleGAN:

Operation	Tensorflow (tf.) Function applied to data example x	Applied to
Random flip	tf.image.random_flip_left_right(x)	Image
Resize before crop	tf.image.resize(x , (300, 300))	
Random crop	tf.image.random_crop(x)	Image
Resize after crop	tf.image.resize(x, (256, 256))	Image

3.4.2 GAN

Using the sample code at [63] to implement CycleGAN, I trained a sunny-to-cloudy and cloudy-to-sunny image-to-image transformation GAN models. After the training, only the generators are used in the pipeline, as the discriminators' use is in helping to train the generators, and right now there is no need to use the discriminator to rate and prune the dataset. I used Adam optimizer and trained the model for 20 epochs.

3.4.3 Segmentation

We compiled different segmentation models from VGG and DenseNet base models, and evaluated them to find the most suitable model for the task. In addition, the effect of two optimization algorithms (RMSProp and Adam) on the performance is investigated.

The sizes are chosen to produce a segmentation model with a symmetric encoder-decoder structure.

3.4.3.1 Encoder

For the encoder part, I evaluated segmentation models VGG16, VGG19, DenseNet121, DenseNet169 and DenseNet201, and the layers taken from those models are shown in Table 4. The results how these models performed in accuracy evaluation can be found in the chapter 3.5.1 on Table 7.

Table 4: Layers chosen from corresponding pre-trained models and their output dimensions.

Encoder	**Layers**
VGG16	conv1 (256, 256, 64), pool1 (128, 128, 64), pool2 (64, 64, 128), pool3 (32, 32, 256), pool4 (16, 16, 512), pool5 (8, 8, 512)
VGG19	conv1 (256, 256, 64), pool1 (128, 128, 64), pool2 (64, 64, 128), pool3 (32, 32, 256), pool4 (16, 16, 512), pool5 (8, 8, 512)
DenseNet121	conv1/relu (128, 128, 64), pool2_relu (64, 64, 256), pool3_relu (32, 32, 512), pool4_relu (16, 16, 1024), relu (8, 8, 1024)
DenseNet169	conv1/relu (128, 128, 64), pool2_relu (64, 64, 256), pool3_relu (32, 32, 512), pool4_relu (16, 16, 1024), relu (8, 8, 1024)
DenseNet201	conv1/relu (128, 128, 64), pool2_relu (64, 64, 256), pool3_relu (32, 32, 512), pool4_relu (16, 16, 1024), relu (8, 8, 1024)

Figures 11 and Figure 12 have a plot for U-Net model with backbone VGG19 and DenseNet201 as an example, respectively. Architectures for VGG16, DenseNet121 and DenseNet169 are similar to their newer versions VGG19 and DenseNet201.

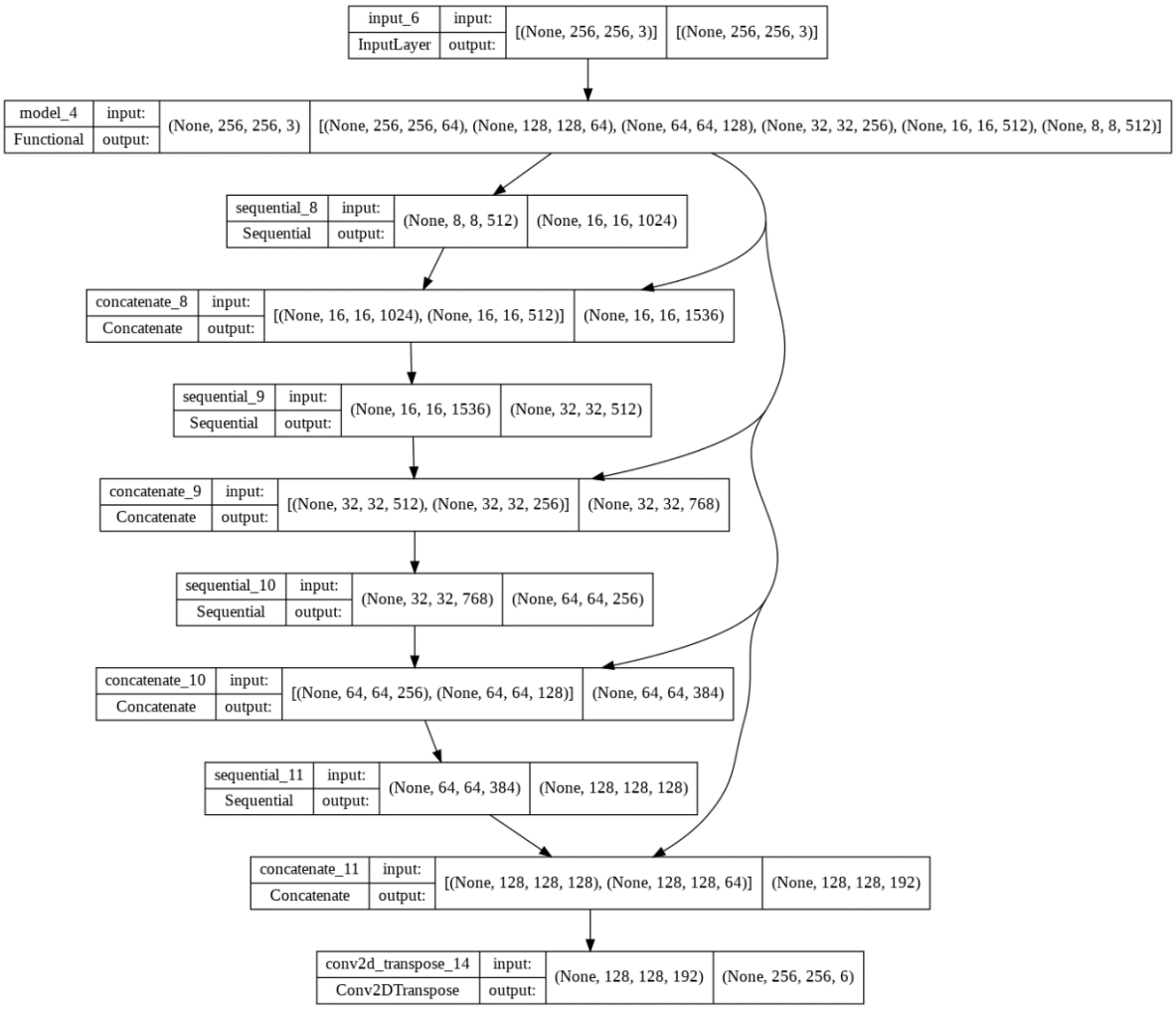

Figure 11: U-Net architecture with backbone VGG19.

Output shape of the segmentation models is 256x256x6; 6-sized vector for each class at every pixel.

3.4.3.2 Decoder

For the decoder, the Pix2Pix network is used to construct the upsampling stack with pix2pix.upsampling() function with the values shown in Table 5, each feature layer decreasing the number of the features [64]. The layer shapes are chosen to match the shape of the corresponding encoder layers. The pix2pix.upsampling() function in TensorFlow includes Transposed convolution, BatchNormalization and ReLU layers by default, as shown in Table 6.

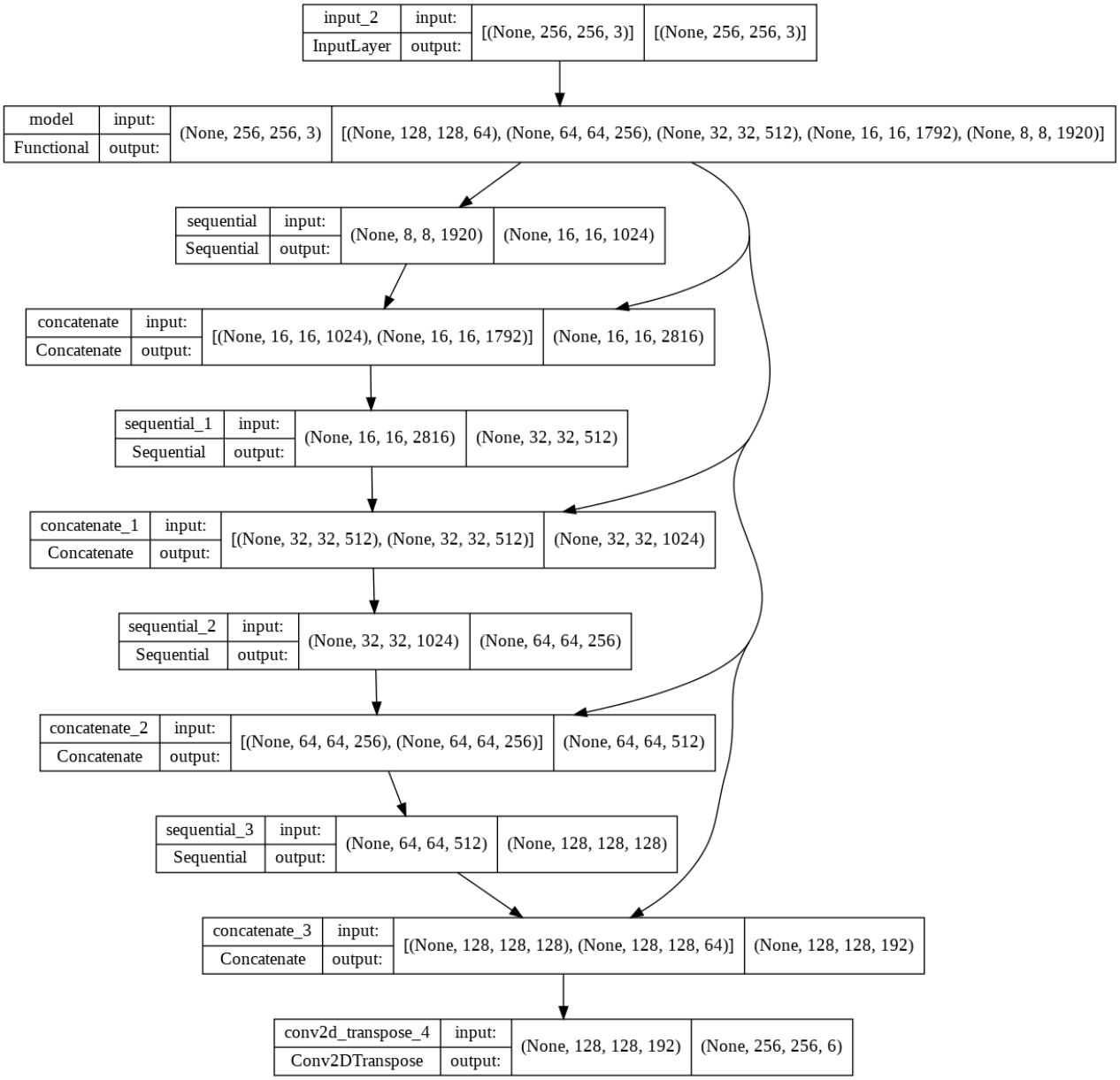

Figure 12: U-Net architecture with backbone DenseNet201.

Table 5: Layer shapes used for Pix2Pix decoder part of the segmentation models.

Pix2Pix layer shape
16, 16, 1024
32, 32, 512
64, 64, 256
128, 128, 128

Table 6: Operations added to a layer in pix2pix.upsampling() function in TensorFlow.

Operation	Function
Transposed 2D Convolution	tf.keras.layers.Conv2DTranspose(filters, 3, strides=2, padding='same', kernel_initializer=initializer use_bias=False)
BatchNormalization	tf.keras.layers.BatchNormalization()
InstanceNormalization	tfa.layers.InstanceNormalization()
DropOut	tf.keras.layers.Dropout(0.5)
ReLU	tf.keras.layers.ReLU()

In Figure 13, the code for loading a pre-trained VGG19 model and creating a U-Net model is shown. A new model for the segmentation is built starting with an input layer with input shape (256, 256, 3). Desired layer names from the pre-trained VGG19 model are defined, their outputs fetched and stacked, which form the encoder (downstack) part of the model. The encoder weights are set to non-trainable, as the pre-trained features appear to perform well for the initial segmentation task, and this reduces the training complexity.

After that, each pix2pix-upsampling decoder layer is connected to an encoder layer in reverse order with concatenation. For the output layer, a Transposed 2D convolution layer with output shape 6 is used, as there are 6 segmentation labels. The code is similar for the VGG16 and each DenseNet model.

```
[ ]  vgg19 = keras.applications.VGG19(input_shape=[256,256,3],
                                      include_top=False,
                                      weights="imagenet")
```

```
"""Input layer"""
input = keras.layers.Input(shape=[256,256,3])

"""Downstack layer names from VGG19"""
downstack_layers = ["block1_conv1",
            "block1_pool",
            "block2_pool",
            "block3_pool",
            "block4_pool",
            "block5_pool"
            ]

"""Upstack layers created with pix2pix"""
upstack_layers = [pix2pix.upsample(1024,3),
          pix2pix.upsample(512,3),
          pix2pix.upsample(256,3),
          pix2pix.upsample(128,3)]#,
          #pix2pix.upsample(64,3)]

"""Get the list of skip connections outputs"""
skip_outputs = [vgg19.get_layer(x).output for x in downstack_layers]
for i in range(len(skip_outputs)):
    print(skip_outputs[i])

"""Initialize the downstack model"""
downstack = keras.Model(inputs=vgg19.input,
                        outputs=skip_outputs)
downstack.trainable = False

"""Create the downstack hierarchy"""
encoder = downstack(input)
layer_out = encoder[-1]
skip_connections = reversed(encoder[:-1])

"""Map the skip connections"""
for decoder, skip in zip(upstack_layers, skip_connections):
    layer_out = decoder(layer_out)
    layer_out = keras.layers.Concatenate()([layer_out, skip])

"""6 classed output (1-5 weather categories + 0 as none)"""
output = keras.layers.Conv2DTranspose(6, 3,
                                      strides=2,
                                      padding="same",
                                      )(layer_out)

"""Combine the connections"""
seg_model = keras.Model(inputs=input, outputs=output)
```

Figure 13: Code snippet of creating the U-Net model from VGG19 layers.

3.4.4 Attention

Attention can be useful when combining different images with segmentation masks, as with it one can give higher emphasis to areas that contribute more to a pixel being labeled into a

certain class. The segmentation network only outputs the argmax of the different class probabilities of a certain pixel. The predicted classes will get values between [0, 1], called class activation maps, which affects how strongly a transformation takes effect together with segmentation. The class activation maps are multiplied with the segmentation layer to determine how strongly CycleGAN transformation output should be applied.

3.4.5 Combining proposed model outputs

Finally, our proposed pipeline consists of the generator model's transformation of the input image *I*, segmentation model's predicted mask of *I* and the segmentation networks class activation maps for the predicted class's value for *I*. For the final result, these outputs are then combined with element wise multiplication and then applied over the original input image in the following fashion, inspired by the Weather GAN paper:

$$T(x)_{Gen+Seg+Att} = \big(Seg(x) \odot\ Att(x)\big) \odot Gen(x) + \Big(1 -\ \big(Seg(x) \odot\ Att(x)\big)\Big) \odot x \quad (18)$$

Output without the class activation maps is:

$$T(x)_{Gen+Seg} = \ \big(Seg(x)\big)\ \odot Gen(x) + \big(1 -\ Seg(x)\big)\ \odot x \quad (19)$$

where x is the initial image, $T(x)_{Gen+Seg+Att}$ is the final transformation between CycleGAN model translation, segmentation model output mask and class activation maps form the segmentation model, $T(x)_{Gen+Seg}$ is the final transformation between CycleGAN model translation and segmentation model output mask, $Seg(x)$ is the segmentation model output mask, and $Att(x)$ is the class activation map of the segmentation model. Figure 14 shows an example transformations from cloudy to sunny and sunny to cloudy.

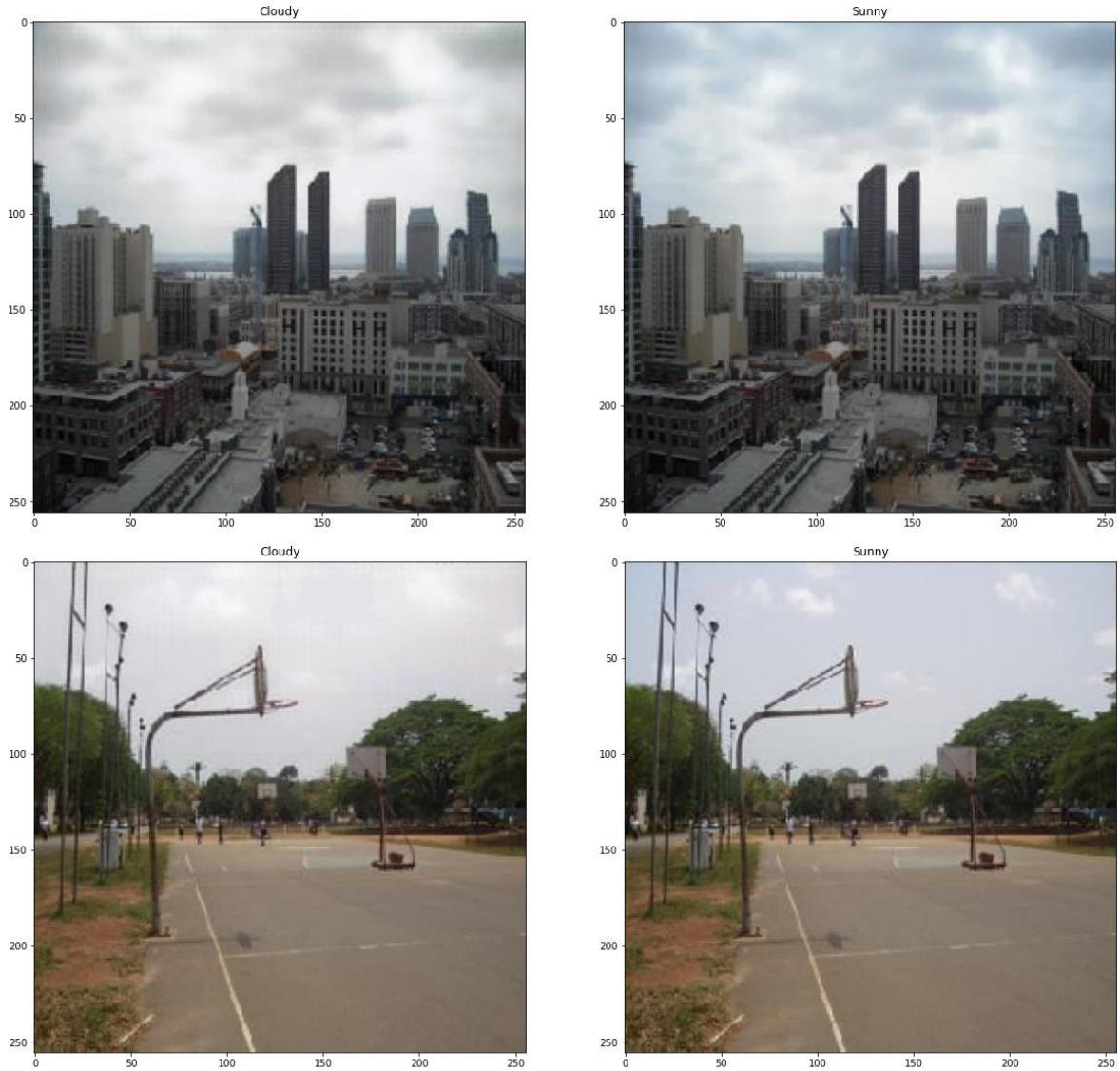

Figure 14: Examples of of the final output using the proposed model.

3.5 Evaluation

3.5.1 Semantic segmentation

Accuracy metric is simply testing what percentage of pixels of an image are correctly predicted, i.e., how similar the predicted mask and ground truth mask are. It's important to note that the true masks in the Multitask Weather dataset are crude, as seen in the Figure 15 and chapter 3.3.

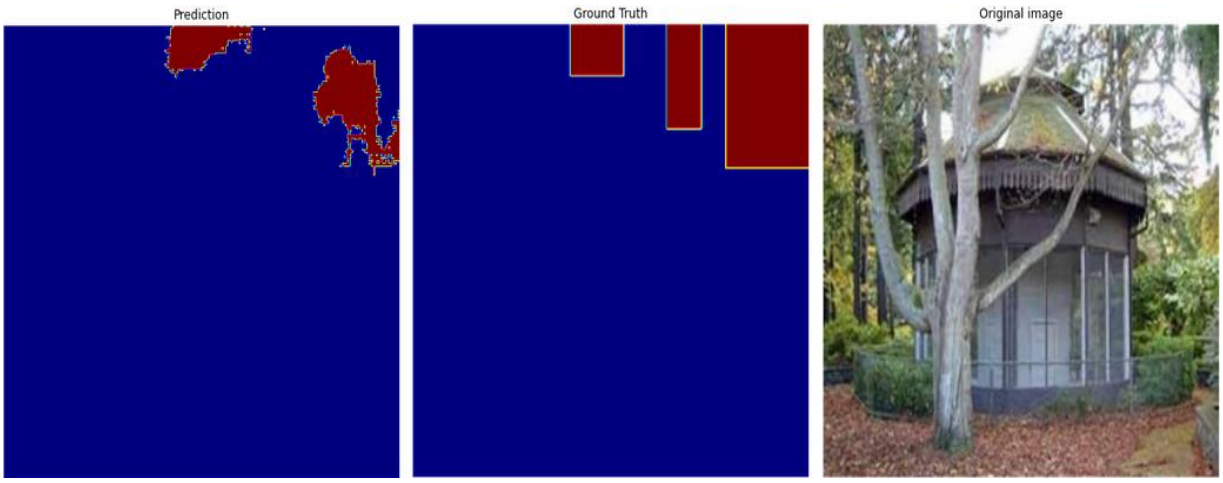

Figure 15: Problem with the way small details are labeled in the original data (Ground Truth) vs Prediction of the segmentation model.

Nevertheless, the model is able to make predictions that do not follow the same large rectangular bounding box-pattern as the dataset. However, the evaluation metric results that use the dataset segmentation masks as ground truth cannot be completely trusted because of this reason. In Table 7, the evaluation results for the segmentation model are shown for validation and training data.

Table 7: Average validation and training accuracy and losses:

Default Model (Optimizer used)	Validation Accuracy	Val. Loss	Training Accuracy	Training Loss
VGG16 (RMSProp)	83.39%	0.585	82.42%	0.600
VGG16 (Adam)	83.6%	0.520	82.99%	0.565
VGG19 (RMSProp)	**85.06%**	0.500	84.9%	0.489
VGG19 (Adam)	85.01%	**0.451**	**84.98%**	**0.406**
DenseNet121 (RMSProp)	83.24%	0.569	82.82%	0.533
DenseNet121 (Adam)	83.9%	0.556	83.08%	0.520
DenseNet169 (RMSProp)	84.24%	0.542	83.98%	0.491
DenseNet169 (Adam)	84.07%	0.551	83.66%	0.509
DenseNet201 (RMSProp)	81.67%	0.614	81.95%	0.550
DenseNet201 (Adam)	84.02%	0.524	84.23%	0.490

A general problem with accuracy metric is, that if there is a large imbalance in the categories, one might get a high accuracy result while the model would not perform well. For example, if only 1% pixels of an image is Class *A* and 99% of the pixels are Class *B*, if the model predicts 100% of the image to be of class *B* the accuracy will be 99%. If this kind of ratio is

also present in the training data, the model might well converge in a direction where always predicting class *B* will yield higher reward, i.e. overfit.

However, in this training process the categories are divided to equal sizes. A possible problem with this dataset is that, since the ground data does not consider small details like shown in Picture 6, the benefit from alternative metrics such as Intersection-Over-Union [65] would be negligible.

Overfitting does not appear to be present, as the model performed well on non-training data that it hadn't seen before, and training and especially validation losses decreased throughout the training. Addition to this, the prediction results shown in Picture 6 also affirms on its part that the model hasn't overfit to the dataset's way of labeling large bounding box chunks. As shown in Figure 16, some images of the dataset do not provide much information for the generator or the segmentation network, and it might have been better to prune the dataset for the segmentation and/or GAN model of images containing only one segmentation mask.

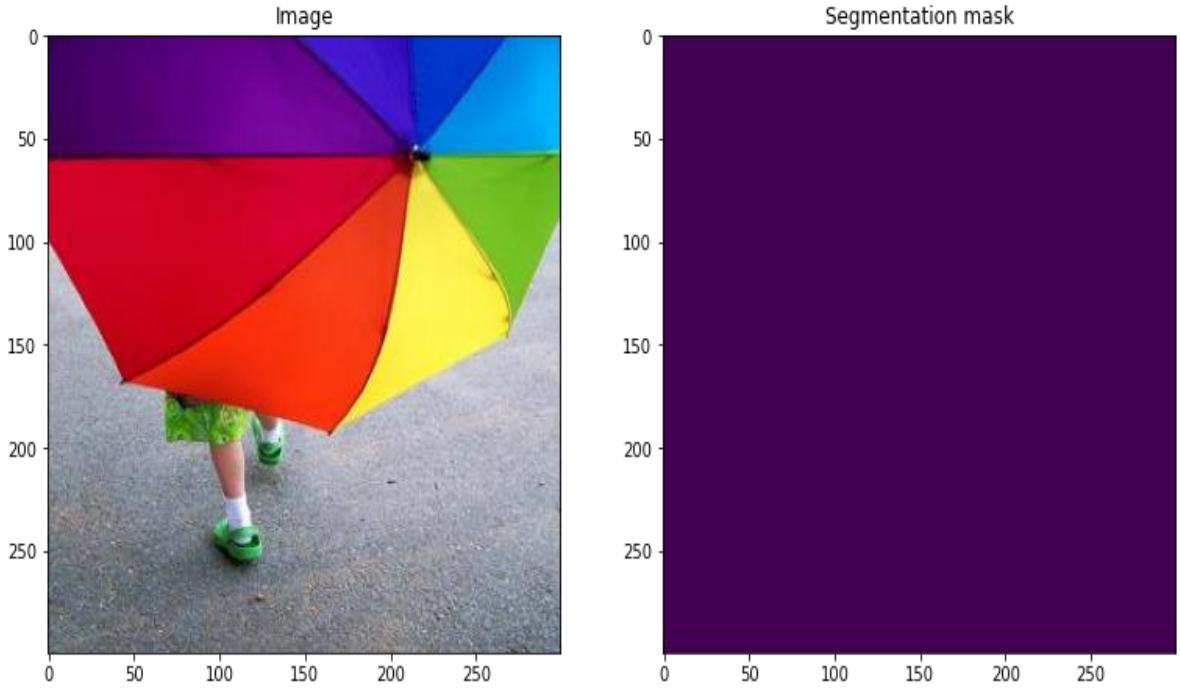

Figure 16: Some images in the dataset "Multitask weather" are not very distinguishable between categories, and only contain of one segmentation class.

On manual inspection, and since evaluation based on the ground truth masks is not accurate, VGG19-based model appears to score the actual weather cues best out of the 5 models, and perform better on edges. Therefore, since there was no large gap in accuracies, VGG19 is

chosen as the segmentation model. However, all the proposed models appear to perform reasonably well for the task, answering RQ2 for this thesis that there isn't a significant difference between tested segmentation models or optimizers.

3.5.2 Combined model

The combined model is evaluated with Fréchet inception distance (FID). FID is used to evaluate the performance of CycleGAN with and without the attention model. Figure 17 shows the code for FID score implementation.

```
def get_fid_score(model, real_data, fake_data):
  mult = 2.0
  act1 = model.predict(real_data)
  act2 = model.predict(fake_data)

  d1, s1 = act1.mean(axis=0), np.cov(act1, rowvar=False)
  d2, s2 = act2.mean(axis=0), np.cov(act2, rowvar=False)

  cmean = scipy.linalg.sqrtm(s1.dot(s2))

  if np.iscomplexobj(cmean):
    cmean = cmean.real

  fid = np.sum((d1 - d2)**mult) + np.trace(s1 + s2 - mult * cmean)

  return fid

fid_cyclegan_s = get_fid_score(inception_model, //
                               batch_real, combg_batch.reshape(-1, 256, 256, 3))

cyclegan_fakes = generator_fid(np_sunny)

fid_cyclegan = calculate_fid_score(inception_model, batch_real, cyclegan_fakes)
```

Figure 17: Code for FID score implementation.

The problem with dataset segmentation masks being crude, mentioned in Chapter 3.5.1 "Semantic segmentation", does not similarly affect the evaluation of the combined model, as the comparison is done between the generated weather transformations and real images. The evaluation process is done by choosing a batch of images from weather category *A*, which are transformed to category *B*. After that, a batch of images from weather category *B*

are chosen for the evaluation, and FID scores are calculated for both batches using Inception v3 network. Figure shows the code for implementing FID score.

Average FID score of the models (lower is better) is calculated over 1000 images, as shown in Table 8. The comparison was done by selecting different samples from the test dataset, which was not used in training, in both categories, and calculating the scores for same real data on CycleGAN, CycleGAN with segmentation and CycleGAN with segmentation and Class-activation attention.

To give a metric to compare to, Table 9 shows FID scores calculated between real cloudy and real sunny batches was around 380, and average between batches of the same domain was 238.5.

Table 8: Fréchet inception score comparison between pipelines.

Model	Cloudy-to-Sunny FID	Sunny-to-Cloudy FID
CycleGAN	310.8	301.4
CycleGAN with segmentation	**290.5**	285.7
CycleGAN with segmentation and Class-activation	291.2	**283.0**

Table 9: Fréchet inception scores between real data domains.

Real data scores (for relevance)	FID
Score between batches of different domains	380.1
Score between batches of Cloudy domain	232.0
Score between batches of Sunny domain	244.9

CycleGAN without segmentation model sometimes transfers also non-weather-related areas like the ground. Although such features often also change in color when the weather changes (for example, the brightness change between sunny and cloudy weather), the FID scores do support the claim that CycleGAN by itself transfers unrelated areas of the image too much.

According to the FID scores, the answer to (RQ1) is that there are slight improvements when combining a segmentation model with a CycleGAN model, at least with the given data. When taking into account that the FID score between batches of the same domain was 238.5, the results appear to be acceptable, since they are closer to that score than the difference between the two domains, (380.1).

When also taking into note the blurring that CycleGAN produces to the images after domain transform, the benefit to overall image quality might be even higher. However, there was no clear benefit when multiplying the class-activation maps with the segmentation maps (equation 18) compared to only using the segmentation model (equation 19).

On the other hand, there is added complexity when training two (CycleGAN and segmentation) models, and also a higher probability of one of the models not performing well. Therefore, adding additional models should only be done when a generator's output alone is not enough to produce realistic images.

4 Conclusions

To summarize, the following answers are provided to address the research questions introduced at the beginning;

RQ1: **Is there a significant difference in data augmentation results between CycleGAN output and CycleGAN output coupled with segmentation model?**

> There is a slight difference, and when considering the blurring that CycleGAN does to the entire image when transferring image domains, there is a clear benefit on using a segmentation model. Future work might benefit from more sophisticated attention method (a distinct model).

RQ2: **How well different segmentation model configurations and pre-trained models affect segmentation results and CycleGAN-segmentation model pipeline output?**

> VGG19 appeared best both according to the metrics and visual inspection of segmentation results, as the accuracy metric was not completely reliable with the segmentation masks having bounding box-level labeling. Difference between different segmentation model configurations was within 3.5% range using accuracy metric, but the configurations did however use mostly similar architectures. Large-scale testing of different layer combinations was not carried out, but upon changing dataset and/or resolution, if model performance drops changing the architecture would become necessary.

4.1.1 Benefits

As can be seen from Figure 18, one of the benefits of the proposed thesis method is, that lone CycleGAN transformation often blurs all the details of an image, compared to the combined segmentation and CycleGAN model. Additionally, it can be seen that the roofs of the houses and some dirt is also colored in blue when they most likely shouldn’t, as CycleGAN alone is overfit to translating colors to sky blue. Using a segmentation model to only carry over the weather-cue pixels has the benefit of keeping the rest of the image intact.

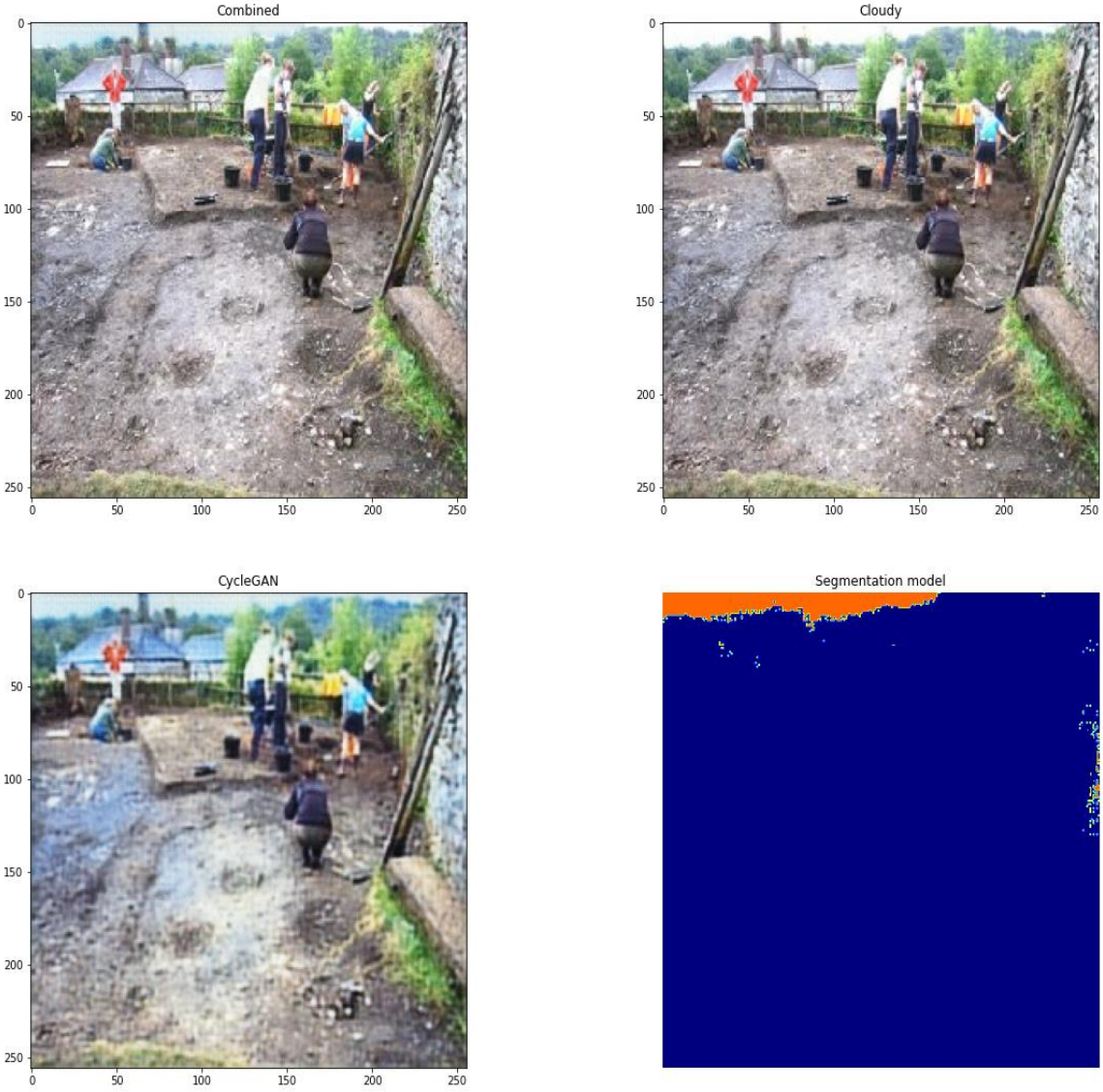

Figure 18: Comparison between the blurring that CycleGAN sometimes does to the entire image and the image, that applies transformation only to area the segmentation model predicts.

4.1.2 Problems

Sometimes the segmentation mask does not correctly detect all weather-related areas, which results in sharp transitions between domains or CycleGAN weather transformation not being applied, as can be seen in Figure 19. Possible solution in future work to combat problems with rough edges or missing transformations, apart from improvements to the model, might be Gaussian blurring or denoising to the relevant border pixels.

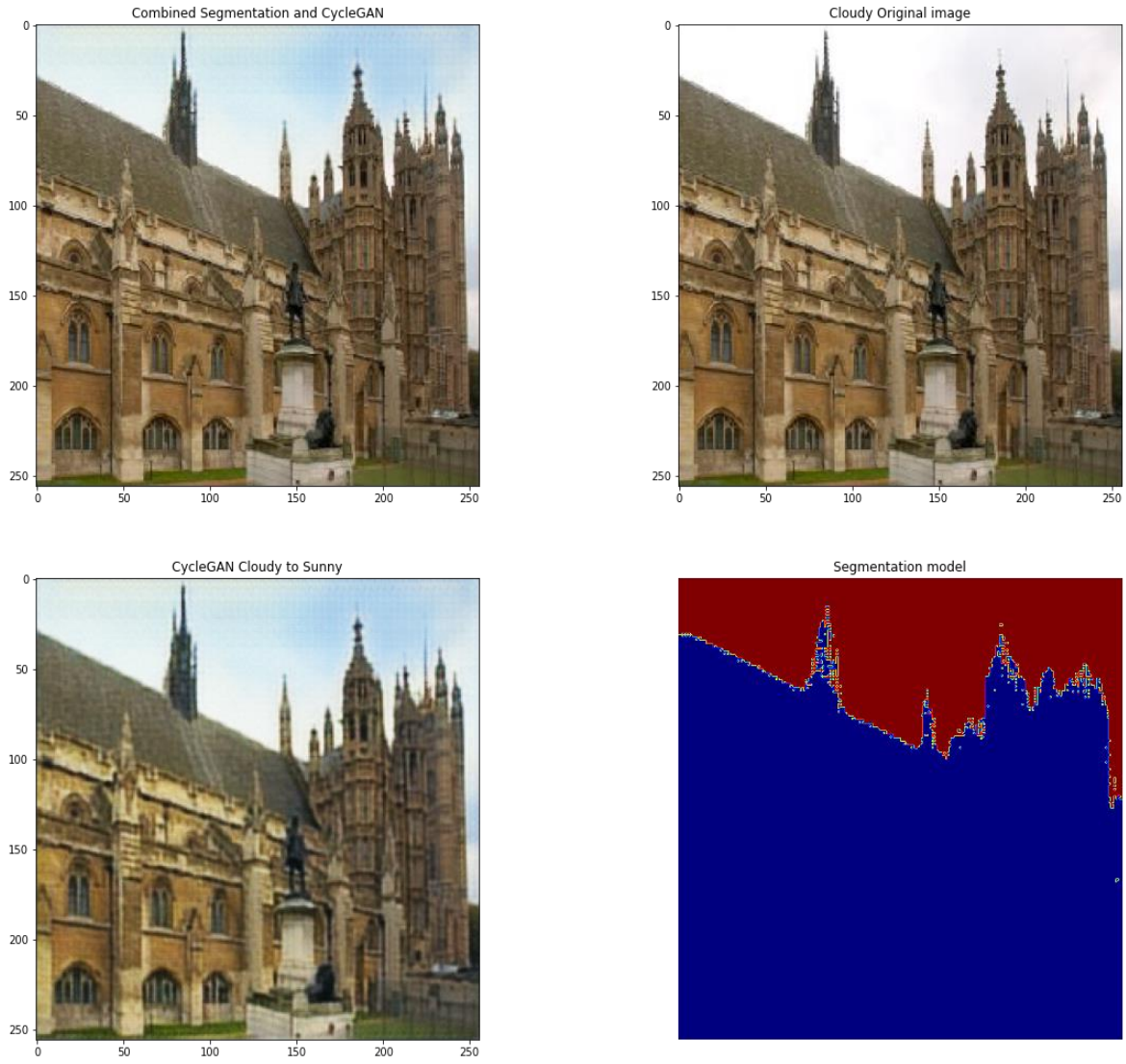

Figure 19: A problem with the segmentation mask sometimes occurs, where all weather-related pixels are not correctly classified, as can be seen in the lower right corner. Blurring that occurs with CycleGAN only can also be seen in this image by inspecting, how features of the building and statue are affected.

4.2 Future work

4.2.1 Generalization to different datasets

Given that there is a dataset that includes segmentation labels, this implementation can be used with other datasets with little alteration to number of segmentation model output classes, and if the resolution changes or performance is low, to the layer structure. For projects without segmentation labels, CycleGAN can be used by itself.

4.2.2 Improvement suggestions

There would have been more room to experiment on by changing the U-Net model architecture, for example to use more convolutional layers instead of maxpooling layers from the pretrained DenseNet/VGG networks, increasing the amount of either layers or features and using a U-Net model with attention to see if the final output results improve. Attention U-Net adds an attention Gate module to U-Net's decoder skip connections, which combine two layer outputs; the decoder layer before it, and corresponding encoder layer [66]. Another improvement on the segmentation model might have been using UNet++ or UNet3+ architectures, which extend ground truth supervision to additional layers, not just the output layer. These two architectures have been shown to improve on segmentation results at least in the medical field [67], [68].

Also, training a distinct attention model by using perceptual loss [69] against the segmentation model's or base VGG19's last hidden layer as target might have also provided more accurate transformations. However, adding a yet another model does increase the training complexity and possibility of one model not converging, but mutually robustness of the pipeline might increase if at least one spatial (segmentation or attention) model is accurate, given that the equations (18, 19) combining the outputs of the segmentation and GAN models is chosen well. Training by instead of training the models separately, combining the segmentation and generator outputs and inputting the result to the discriminator might also have improved the produced model pipeline.

The dataset could also have been pruned by having the CycleGAN discriminator or distinctly trained classifier classify the images and remove lowest scored 10% of the images, for example. In this case the lowest score means the images, that according to the discriminator's rating have the lowest probability of belonging to the class they are labeled to. As of now, the data did contain some images with minimal weather-cues, such as close-up images of environment without visible sky or other weather-elements, and no clear lightning difference, which should be removed for a more useful dataset. This could simply be done by removing images containing only one unique segmentation label, or where the distribution of labels is too unbalanced. Alternatively, one could attempt to generate segmentation labels for novel images by inputting an image to the segmentation model and using the generated segmentation mask. This practise is however risky, and would only produce useful masks for weather conditions that the segmentation model is already trained

on, unless some categories would reasonably translate to each other's in labels (such as cloudy to foggy or rainy).

Sometimes the segmentation mask did not correspond to whole weather area, leaving gaps consisting of a few pixels. Aside from improvements to the models, solution to this might be thresholding or denoising to the border of such segmentation result to ensure that the combining phase (with equations 18 or 19) doesn't leave rough patches. However, this method would be troublesome for guessing how the image is supposed to usually be.

There has been a tremendous advancement on the generative model field with diffusion models [70], which include text-to-image models StableDiffusion [71] and Dall-e [72], to name a few. The methods that those models use include both textual input and different generation methods. On the other hand, methods used in state-of-the-art projects might be expensive to train or use, or require a large amount of data.

StableDiffusion's diffusion model can be coupled with variational autoencoder (VAE). VAEs capture the average representation of data, thus capturing the low-dimensional patterns in data, but making generated examples less original than GANs, and VAEs have been less popular for image generation than GANs after StyleGAN and other state-of-the-art GAN architectures. StableDiffusion's method is to combine generative model to generate an initial image, and VAE to fix small detail errors, for which it appears to work well [71]. ControlNet [73] is one such StableDiffusion variation candidate, well suited for image-to-image transformation from an input image without affecting all features of the input image. Diffusion model and VAE might be useful to replace the CycleGAN model altogether, use VAE with CycleGAN model, or alternatively look for different image-to-image GAN architectures.

Possible continuations for the research questions in this thesis could be how well the produced data can be used in real-world applications, where collecting images of the same location in different weather conditions would be expensive. This could mean, for example, using an additional computer vision model to evaluate, how visibility changes in different weather conditions by inspecting the generated images.

4.3 Final words

The thesis generated knowledge about producing a working segmentation+GAN pipeline for a custom solution with weather data, which appeared to be a good baseline for future work.

The used evaluation methods seem to support this claim. The results give insights about the potential of the proposed method and what kind of alterations would most likely be useful to try to improve or generalize the solution.

References

[1] Y. Skandarani, P.-M. Jodoin, and A. Lalande, “GANs for Medical Image Synthesis: An Empirical Study.” arXiv, Jul. 19, 2021. Accessed: Dec. 15, 2022. [Online]. Available: http://arxiv.org/abs/2105.05318

[2] H. Chen and L. Jiang, “Efficient GAN-based method for cyber-intrusion detection.” arXiv, Jul. 24, 2019. Accessed: Dec. 15, 2022. [Online]. Available: http://arxiv.org/abs/1904.02426

[3] G. Ravindra Padalkar, S. Dinkar Patil, M. Mallikarjun Hegadi, and N. Kailash Jaybhaye, “Drug Discovery using Generative Adversarial Network with Reinforcement Learning,” in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore, India, Jan. 2021, pp. 1–3. doi: 10.1109/ICCCI50826.2021.9402449.

[4] J. P.-A. Ian J. Goodfellow Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, “Generative Adversarial Networks,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1406.2661

[5] T. Karras, S. Laine, and T. Aila, “A Style-Based Generator Architecture for Generative Adversarial Networks.” arXiv, Mar. 29, 2019. Accessed: Mar. 20, 2023. [Online]. Available: http://arxiv.org/abs/1812.04948

[6] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks,” *arXiv:1703.10593 [cs]*, Aug. 2020, Accessed: Apr. 18, 2022. [Online]. Available: http://arxiv.org/abs/1703.10593

[7] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, “Learning to Discover Cross-Domain Relations with Generative Adversarial Networks.” arXiv, May 15, 2017. Accessed: Mar. 20, 2023. [Online]. Available: http://arxiv.org/abs/1703.05192

[8] X. Huang and S. Belongie, “Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization.” arXiv, Jul. 30, 2017. Accessed: Mar. 20, 2023. [Online]. Available: http://arxiv.org/abs/1703.06868

[9] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, “Analyzing and Improving the Image Quality of StyleGAN.” arXiv, Mar. 23, 2020. Accessed: Mar. 20, 2023. [Online]. Available: http://arxiv.org/abs/1912.04958

[10] T. W. Antonia Creswell Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, Anil A. Bharath, “Generative Adversarial Networks: An Overview,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1710.07035

[11] Martin Heusel Hubert Ramsauer Thomas Unterthiner Bernhard Nessler, “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium,” no. Journal Article, [Online]. Available: https://papers.nips.cc/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf

[12] E. Betzalel, C. Penso, A. Navon, and E. Fetaya, “A Study on the Evaluation of Generative Models.” arXiv, Jun. 22, 2022. Accessed: Mar. 21, 2023. [Online]. Available: http://arxiv.org/abs/2206.10935

[13] V. V. Christian Szegedy Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, “Rethinking the Inception Architecture for Computer Vision,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1512.00567

[14] Y. Wei *et al.*, “DerainCycleGAN: Rain Attentive CycleGAN for Single Image Deraining and Rainmaking.” arXiv, Apr. 07, 2021. Accessed: Dec. 19, 2022. [Online]. Available: http://arxiv.org/abs/1912.07015

[15] F. Luleci, F. N. Catbas, and O. Avci, “CycleGAN for Undamaged-to-Damaged Domain Translation for Structural Health Monitoring and Damage Detection.” arXiv, Mar. 06, 2022. Accessed: Dec. 19, 2022. [Online]. Available: http://arxiv.org/abs/2202.07831

[16] S. P. Tadem, “CycleGAN with three different unpaired datasets.” arXiv, Aug. 12, 2022. Accessed: Dec. 19, 2022. [Online]. Available: http://arxiv.org/abs/2208.06526

[17] X. Li, K. Kou, and B. Zhao, “Weather GAN: Multi-Domain Weather Translation Using Generative Adversarial Networks.” arXiv, Mar. 09, 2021. Accessed: Nov. 30, 2022. [Online]. Available: http://arxiv.org/abs/2103.05422

[18] I. Goodfellow, Y. Bengio, and A. Courville, “Deep Learning,” 2016th ed., MIT Press, 2016.

[19] E. S. Jonathan Long Trevor Darrell, “Fully Convolutional Networks for Semantic Segmentation,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1411.4038

[20] A. D. Jost Tobias Springenberg Thomas Brox, Martin Riedmiller, “STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET,” no. Journal Article, [Online]. Available: https://arxiv.org/pdf/1412.6806.pdf

[21] L. M. Alec Radford Soumith Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1511.06434

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” arXiv, May 24, 2019. Accessed: Feb. 05, 2023. [Online]. Available: http://arxiv.org/abs/1810.04805

[23] V. Sivakumar and V. Murugesh, “A brief study of image segmentation using Thresholding Technique on a Noisy Image,” in *International Conference on Information Communication and Embedded Systems (ICICES2014)*, Chennai, India, Feb. 2014, pp. 1–6. doi: 10.1109/ICICES.2014.7034056.

[24] B. Enyedi, L. Konyha, and K. Fazekas, “Threshold procedures and image segmentation,” in *47th International Symposium ELMAR, 2005.*, Zadar, Croatia, 2005, pp. 29–32. doi: 10.1109/ELMAR.2005.193633.

[25] C.-C. Chu and J. K. Aggarwal, “The integration of region and edge-based segmentation,” in *[1990] Proceedings Third International Conference on Computer Vision*, Osaka, Japan, 1990, pp. 117–120. doi: 10.1109/ICCV.1990.139507.

[26] M. Brejl and M. Sonka, “Edge-based image segmentation: machine learning from examples,” in *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, Anchorage, AK, USA, 1998, vol. 2, pp. 814–819. doi: 10.1109/IJCNN.1998.685872.

[27] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation.” arXiv, May 18, 2015. Accessed: Oct. 25, 2022. [Online]. Available: http://arxiv.org/abs/1505.04597

[28] “Albert-Ludwigs-Universität Freiburg //lmb.informatik.uni-freiburg.de/people/ronneber/u-net/ (Accessed 25.01.2023)”.

[29] L. Perez and J. Wang, “The Effectiveness of Data Augmentation in Image Classification using Deep Learning.” arXiv, Dec. 13, 2017. Accessed: Mar. 05, 2023. [Online]. Available: http://arxiv.org/abs/1712.04621
[30] S. Yang, W. Xiao, M. Zhang, S. Guo, J. Zhao, and F. Shen, “Image Data Augmentation for Deep Learning: A Survey.” arXiv, Apr. 18, 2022. Accessed: Mar. 05, 2023. [Online]. Available: http://arxiv.org/abs/2204.08610
[31] F. H. K. dos S. Tanaka and C. Aranha, “Data Augmentation Using GANs.” arXiv, Apr. 19, 2019. Accessed: Nov. 29, 2022. [Online]. Available: http://arxiv.org/abs/1904.09135
[32] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.
[33] J. B. Diederik P. Kingma, “Adam: A Method for Stochastic Optimization,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1412.6980
[34] N. Qian, “On the momentum term in gradient descent learning algorithms,” *Neural Networks*, vol. 12, no. 1, pp. 145–151, Jan. 1999, doi: 10.1016/S0893-6080(98)00116-6.
[35] H. Robbins and S. Monro, “A Stochastic Approximation Method,” *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, Sep. 1951, doi: 10.1214/aoms/1177729586.
[36] R. R. Ashia C. Wilson Mitchell Stern, Nathan Srebro, Benjamin Recht, “The Marginal Value of Adaptive Gradient Methods in Machine Learning,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1705.08292
[37] B. M. Ian Gemp, “The Unreasonable Effectiveness of Adam on Cycles,” no. Journal Article, [Online]. Available: https://sgo-workshop.github.io/CameraReady2019/11.pdf
[38] F. H. Ilya Loshchilov, “Decoupled Weight Decay Regularization,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1711.05101
[39] T. T. Juntang Zhuang Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, James S. Duncan, “AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients,” no. Journal Article, [Online]. Available: https://arxiv.org/pdf/2010.07468.pdf
[40] I. G. Tim Salimans Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, “Improved Techniques for Training GANs,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1606.03498v1
[41] J. C. Casper Kaae Sønderby Lucas Theis, Wenzhe Shi, Ferenc Huszár, “Amortised MAP Inference for Image Super-resolution,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1610.04490
[42] L. B. Martin Arjovsky, “Towards Principled Methods for Training Generative Adversarial Networks,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1701.04862
[43] K. Fukushima, “Cognitron: A self-organizing multilayered neural network,” *Biol. Cybernetics*, vol. 20, no. 3–4, pp. 121–136, 1975, doi: 10.1007/BF00342633.
[44] A. F. Agarap, “Deep Learning using Rectified Linear Units (ReLU).” arXiv, Feb. 07, 2019. Accessed: Mar. 05, 2023. [Online]. Available: http://arxiv.org/abs/1803.08375
[45] Vinod Nair Geoffrey E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” no. Journal Article, [Online]. Available: https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf

[46] A. Y. H. Andrew L. Maas Andrew Y. Ng, “Rectifier Nonlinearities Improve Neural Network Acoustic Models,” no. Journal Article.
[47] A. S. Fisher Yu Yinda Zhang, Shuran Song, Thomas Funkhouser, Jianxiong Xiao, “LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1506.03365
[48] N. W. Bing Xu Tianqi Chen, Mu Li, “Empirical Evaluation of Rectified Activations in Convolution Network,” no. Journal Article, [Online]. Available: https://arxiv.org/pdf/1505.00853.pdf
[49] J. F. Ritchie Ng, “DeepLearningWizard.com; Weight Initializations & Activation Functions,” no. Journal Article, [Online]. Available: https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/weight_initialization_activation_functions/)
[50] J. D. Andrew Brock Karen Simonyan, “Large Scale GAN Training for High Fidelity Natural Image Synthesis,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1809.11096
[51] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein GAN.” arXiv, Dec. 06, 2017. Accessed: Oct. 27, 2022. [Online]. Available: http://arxiv.org/abs/1701.07875
[52] F. A. Ishaan Gulrajani Martin Arjovsky, Vincent Dumoulin, Aaron Courville, “Improved Training of Wasserstein GANs,” no. Journal Article.
[53] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets.” arXiv, Nov. 06, 2014. Accessed: Mar. 05, 2023. [Online]. Available: http://arxiv.org/abs/1411.1784
[54] J.-Y. Z. Phillip Isola Tinghui Zhou, Alexei A. Efros, “Image-to-Image Translation with Conditional Adversarial Networks,” no. Journal Article, [Online]. Available: https://arxiv.org/abs/1611.07004
[55] A. G. Xiaolong Wang, “Generative Image Modeling using Style and Structure Adversarial Networks,” no. Journal Article, [Online]. Available: https://arxiv.org/pdf/1603.05631.pdf
[56] N. K. Donggeun Yoo Sunggyun Park, Anthony S. Paek, In So Kweon, “Pixel-Level Domain Transfer,” no. Journal Article, [Online]. Available: https://arxiv.org/pdf/1603.07442.pdf
[57] Y. A. Mejjati, C. Richardt, J. Tompkin, D. Cosker, and K. I. Kim, “Unsupervised Attention-guided Image to Image Translation.” arXiv, Nov. 08, 2018. Accessed: Nov. 30, 2022. [Online]. Available: http://arxiv.org/abs/1806.02311
[58] L. Qian, X. Zhou, Y. Li, and Z. Hu, “UNet#: A UNet-like Redesigning Skip Connections for Medical Image Segmentation.” arXiv, May 23, 2022. Accessed: Dec. 19, 2022. [Online]. Available: http://arxiv.org/abs/2205.11759
[59] I. Ahmed, M. Ahmad, F. A. Khan, and M. Asif, “Comparison of Deep-Learning-Based Segmentation Models: Using Top View Person Images,” *IEEE Access*, vol. 8, pp. 136361–136373, 2020, doi: 10.1109/ACCESS.2020.3011406.
[60] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition.” arXiv, Dec. 10, 2015. Accessed: Mar. 21, 2023. [Online]. Available: http://arxiv.org/abs/1512.03385
[61] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks.” arXiv, Jan. 28, 2018. Accessed: Nov. 01, 2022. [Online]. Available: http://arxiv.org/abs/1608.06993

[62] B. Zhao, X. Li, X. Lu, and Z. Wang, “A CNN-RNN Architecture for Multi-Label Weather Recognition,” *arXiv:1904.10709 [cs]*, Apr. 2019, Accessed: Apr. 21, 2022. [Online]. Available: http://arxiv.org/abs/1904.10709
[63] “CycleGAN Tensorflow tutorial a thttps://www.tensorflow.org/tutorials/generative/cyclegan (Accessed 31.10.2022).”
[64] “Code at tensorflow_examples ‘pix2pix.py’ (accessed 4.1.2023).” [Online]. Available: https://github.com/tensorflow/examples/blob/master/tensorflow_examples/models/pix2pix/pix2pix.py
[65] S. Nowozin, “Optimal Decisions from Probabilistic Models: The Intersection-over-Union Case,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, Jun. 2014, pp. 548–555. doi: 10.1109/CVPR.2014.77.
[66] O. Oktay *et al.*, “Attention U-Net: Learning Where to Look for the Pancreas.” arXiv, May 20, 2018. Accessed: Nov. 28, 2022. [Online]. Available: http://arxiv.org/abs/1804.03999
[67] H. Huang *et al.*, “UNet 3+: A Full-Scale Connected UNet for Medical Image Segmentation.” arXiv, Apr. 19, 2020. Accessed: Dec. 24, 2022. [Online]. Available: http://arxiv.org/abs/2004.08790
[68] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, “UNet++: A Nested U-Net Architecture for Medical Image Segmentation.” arXiv, Jul. 18, 2018. Accessed: Jan. 27, 2023. [Online]. Available: http://arxiv.org/abs/1807.10165
[69] J. Johnson, A. Alahi, and L. Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution.” arXiv, Mar. 26, 2016. Accessed: Dec. 04, 2022. [Online]. Available: http://arxiv.org/abs/1603.08155
[70] L. Yang *et al.*, “Diffusion Models: A Comprehensive Survey of Methods and Applications.” arXiv, Oct. 23, 2022. Accessed: Mar. 21, 2023. [Online]. Available: http://arxiv.org/abs/2209.00796
[71] CompVis, “Stability-AI GitHub repository https://github.com/Stability-AI/stablediffusion.” [Online]. Available: https://github.com/Stability-AI/stablediffusion
[72] A. Ramesh *et al.*, “Zero-Shot Text-to-Image Generation.” arXiv, Feb. 26, 2021. Accessed: Dec. 11, 2022. [Online]. Available: http://arxiv.org/abs/2102.12092
[73] Lvmin Zhang, “ControlNet on GitHub https://github.com/lllyasviel/ControlNet.”