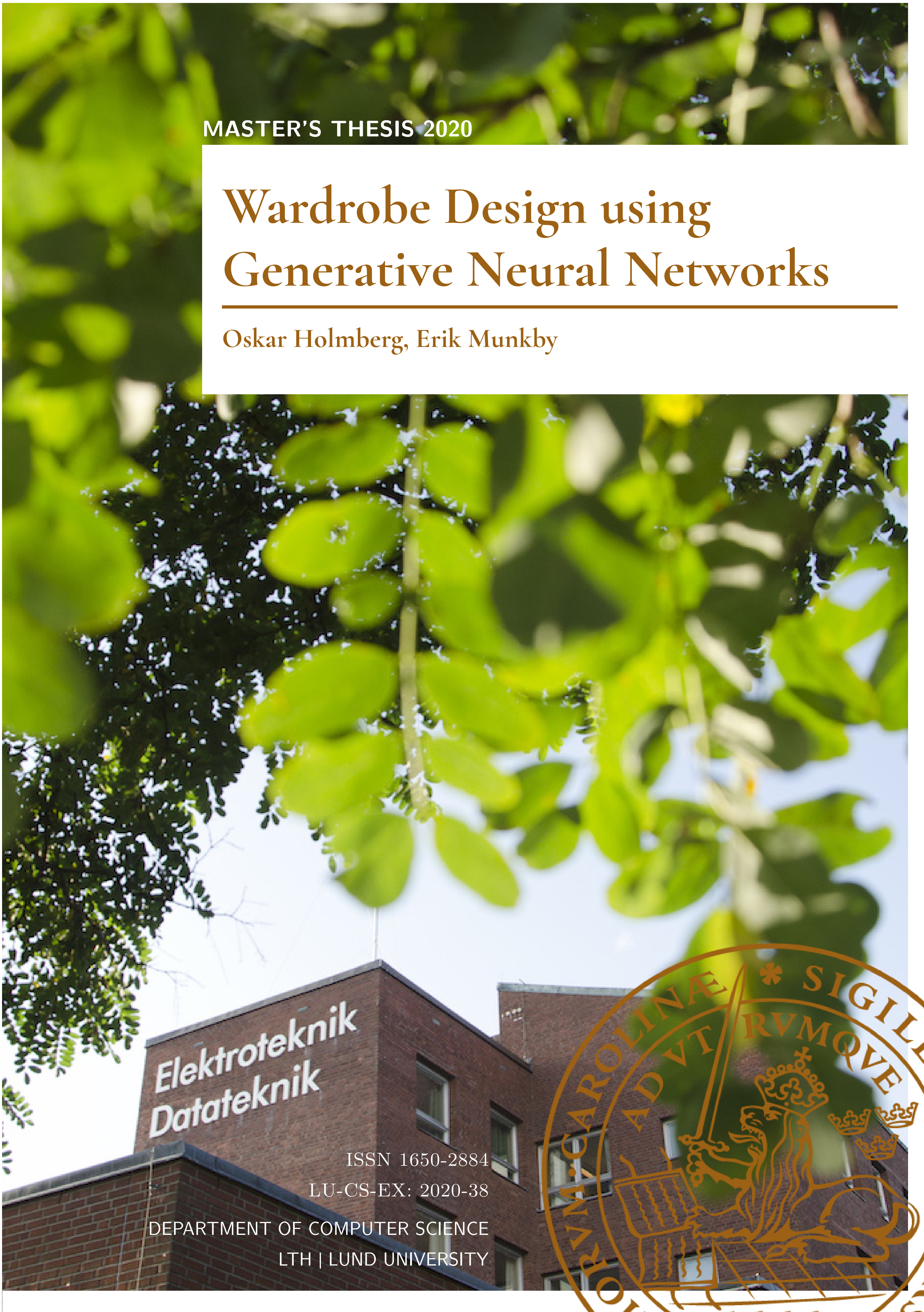
MASTER'S THESIS 2020

Wardrobe Design using Generative Neural Networks

Oskar Holmberg, Erik Munkby



ISSN 1650-2884
LU-CS-EX: 2020-38

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2020-38

Wardrobe Design using Generative Neural Networks

Oskar Holmberg, Erik Munkby

Wardrobe Design using Generative Neural Networks

(A Machine Learning study)

Oskar Holmberg
dat12oho@student.lu.se

Erik Munkby
dat12emu@student.lu.se

May 21, 2020

Master's thesis work carried out at Jayway.

Supervisors: Johnny Dang, johnny.dang@jayway.com
Elin Anna Topp, elin_anna.topp@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

Generative Neural Networks have for many years proven successful at producing continuous data, e.g. images or music. When it comes to discrete data such as text or speech, comparable success has to our knowledge not yet been found. In this study we tackle some of the difficulties of generating discrete data in an attempt to assemble wardrobe configurations from a set of products. This study combines the two Generative Adversarial Network archetypes Auxiliary Classifier and Deep Convolutional to render wardrobe configurations presented in a 3D digital environment. The study shows that the approach can be considered a viable option for automatic generation of modular wardrobes.

Keywords: MSc, machine learning, neural networks, generative neural networks, product design, generative adversarial networks, auxiliary classifier, deep convolutional

Acknowledgements

We want to thank our supervisor Elin Anna Topp for advice and guidance throughout the project, and assistance with the structuring of the paper.

This project would not have been possible without the help our hosting company Jayway, providing us with equipment and an inspirational environment. Special thanks to our supervisor Johnny Dang for guidance and valuable discussions. Petter Skerfving for giving us this opportunity and assisting us in any way he could. We also want to thank the representatives from Company X for showing such interest and supplying us with data and insights.

Contents

Chapter 1
Introduction

This chapter introduces the reader to the study, describing the basis of the study, its purpose and goals. It will also define the scope of the study, what we hope to achieve, and present our contributions.

1.1 Project Description

The project was based on a use case given by a furniture company that wishes to remain anonymous and will henceforth be referred to as Company X. Company X is hosting a digital builder on their website (and in-store machines) that allows customers to virtually construct wardrobe solutions using 3D-models. The builder is meant to inspire customers and allow them to explore possible solutions for wardrobe storage before a purchase is made. Company X is interested in improving the builder and are therefore looking in to ways of dynamically suggesting complete wardrobe configurations based on the individual customer's needs.

Currently the existing suggestions displayed in the digital builder are configured manually by experts, and the same suggestions are given to all customers, furthermore the current solution does not scale in a large international market where demands may differ between regions and seasonal trends. In the new solution Company X wishes to display a variety of suggestions, preferably tailored to the needs of the individual customer, in addition, it is preferable if the solution provides a degree of simulated innovation. By innovation we mean that the solution should be able to present wardrobe configurations that might not have been seen before but still display the same level of functionality and aesthetic appeal as if they were manually configured by an expert in the field.

The IT consultant company Jayway is on behalf of Company X exploring possible solutions to the problem, and the purpose of this study was to investigate whether it could be solved with the help of machine learning. Should this succeed, future work might be directed towards development of a similar solution for Company X's other product lines.

The resources available consisted of large data sets of wardrobe configurations specifying the country where it was sold, which products were used in the configuration, and the relative spacial positioning between the products. The data was provided in JavaScript Object Notation (JSON) [12] and Extensible Markup Language format (XML) [30]. Each data point represents a complete configuration within the digital builder, not including building compatibility and rules. The data entries consist of wardrobe configurations built using the digital builder, at a warehouse, by an expert together with a customer, and subsequently sold.

1.2 Scope

The specific problem of generating wardrobe configurations meeting Company X's criteria of variety and input-dependency mentioned in section 1.1 could possibly be solved using an algorithm, which can output wardrobes based on a set of rules. However, the area Company X wanted to explore whether Machine Learning (ML) was a viable possibility.

The data set available contains wardrobe configurations where several wardrobes have been combined to form a complete unit. We considered the possible solutions for such a space to be too large and time consuming for this study. We also believe that if the problem could be solved for single wardrobes, it would likely be possible to expand the model's capabilities to multi-wardrobe configurations. Therefore we will limit our focus to single wardrobe configurations, and leave multi-wardrobe configurations to future work. In close relation to this, the data set contains wardrobes that hold different amounts of products, and it is less complicated to build ML models for symmetric input and output shapes, since one model will typically take input and produce output of a specific shape. There are methods to solve dynamic shapes but we considered this to be out of our scope. Thus we will focus on wardrobe configurations with a specific amount of products.

One of the other criteria for the wardrobe configurations was the functionality, in this paper we define functionality as *how usable the wardrobe is in regards to product distribution and placement*. This is considered by us to be vital when it comes to the design of the wardrobe, the reason behind this is that we deem an aesthetically appealing wardrobe useless if it does not provide the wanted functionality. Due to this, and the highly subjective nature of how aesthetically appealing an object is, we chose to exclude an aesthetic evaluation from our scope.

Finally, Company X wanted to offer individualized suggestions. The suggestions could be based on factors such as available space, what the wardrobe should store, and what type of room it will be placed in. Company X did not provide us with data regarding the customers intended use for each wardrobe, thus we will limit our individualization to factors retrievable from the data set. We believe that this is enough to prove the concept.

To summarise, our scope is limited to using machine learning and the data provided by Company X to generate single wardrobe configurations, with a specific amount of products. In addition to this, one should be able to define characteristics beforehand, which the generated wardrobes should follow. Moreover, the wardrobes should display variance i.e. follow a non-deterministic behavior while still upholding functionality.

1.3 Method

In this study three different approaches to generative neural networks were evaluated in the context of solving Company X's problem statement. The architecture that showed most promise was the Generative Adversarial Network (GAN) model. The GAN model used in this project was derived from an existing source intended for image generation, created by Radford et. al. [21], and modified to handle our discrete data.

Initially the data provided by Company X was analysed to identify key factors of the provided wardrobe configurations. These factors were considered when building the model and during evaluation of the results. When shaping the model result evaluation was primarily done through quantitative measures such as studying the networks training progression and analysing product distributions in the output. This provided a theoretical measurement of how similar the generated wardrobes were to the real ones. After achieving promising quantitative results, a qualitative study was made in the form of a survey. The purpose of the survey was to receive subjective feedback of our wardrobes, and how well they compared to the real ones from a human perspective.

1.4 Contributions

Generative networks are often used for generating continuous data, such as images or music, in particular Generative Adversarial Networks (GANs) have to our knowledge not proven effective when generating discrete sequential data such as text or speech. Our study presents an example of how this could be approached, in addition to this, we translate the low-level generated output back into 3D models, paving way for a new use of GANs on the consumers market.

The majority of the source code was written together in a pair programming fashion though minor distributions of work was done. Erik Munkby spent more time refining the model, and Oskar Holmberg spent more time gathering and analysing data, but both parties have been involved at all times. The paper was written jointly with some individual focus on the same areas as mentioned above.

Chapter 2
Background

This chapter presents the basis of our project and what previous research we have considered. The chapter also explains how we processed our data and provides some basics in neural networks required to understand the project's technical architecture.

2.1 Neural Networks

This section will provide the reader with a description of some fundamental aspects of neural networks needed to understand our process. We recommend reading this section to readers who are new to Machine Learning (ML) or Artificial Neural Networks (ANNs), or just in need of a brush up on the areas used in this project.

2.1.1 Artificial Neural Network

The terminology "neural network" stems from the main source of inspiration to the field, the brain. Similarly to the brain, neural networks contain neurons and the strength of their collective connections to each other represent memory [4]. Much like the brain an Artificial Neural Network (ANN) makes a decision based on the collective input from all of its neurons, thus making it a useful tool in order to solve non-linear problems.

ANNs are commonly built in the shape of layers, where each neuron is connected to neurons in the previous layer. The connections between neurons are weighted with a value between -1 and 1, meaning that the connection will output the input multiplied by the weight. In an untrained network the weights are assigned randomly, and changed during training to optimize the final result. In addition to the weights, each neuron passes its input through a mathematical function and outputs the transformed value, this function is referred to as an activation function, the specific functions used in this project are further described in 3.4.1.

Figure 2.1 illustrates a simple fully connected neural network with two hidden layers. Each neuron in the network, represented as circles, will receive an input consisting of the

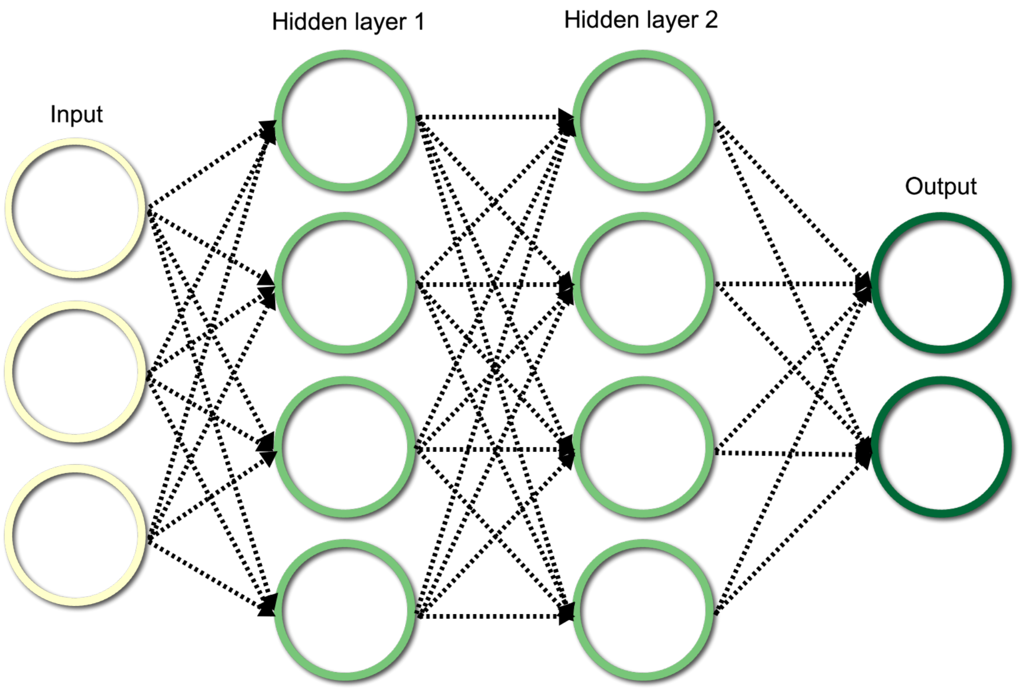


Figure 2.1: A simple neural network with input layer, two hidden layer and an output layer. Each circular node represents a neuron.

sum of the weighted outputs from neurons in the previous layer. The neuron passes the input through its activation function and passes the result forward to all neurons in the next layer, the output of the whole network is given by the values in the output layer.

After the initial setup of an ANN, the network is trained to learn how to produce the wanted output. Training from a neural network's perspective is an iterative process of updating connection weights between neurons until they reach a state where the network outputs desired results. Each time data is run through the network, one can calculate the error in the network by comparing the given output to the expected output, and the goal is to minimize the error. The measurement of the error is also referred to as loss and how the loss is calculated depends on the selected loss function. The most commonly used method, is through back-propagation [10]. The back-propagation algorithm will compare the loss of the output layer (denoted in figure 2.1) and calculate which weight modification in the previous layer (hidden layer 2 in figure 2.1) will cause the most rapid decrease to the overall loss. Further the algorithm will recursively, going backwards, perform this calculation through every layer until it reaches the input layer. Back-propagation is a very computationally heavy operation and is often optimized by optimization functions, such as stochastic gradient descent (SGD) or as predominately used in our model, the Adam optimizer, the details of this will not be explained in this paper but Diederik P. et. al. explain this well in Method for Stochastic Optimization [13].

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) have proven to be a successful machine learning approach within image recognition ([15], [1], [2], [17]). The mathematical theories behind CNNs are quite complicated and will not be described here, instead we will try to explain the general concepts of how they work. For further reading we recommend taking a look at Krizhevsky et al. [15], the authors received impressive results in ImageNet LCVRC 2010 and 2012 competition [23].

Convolution

In order to better understand what a CNN does we must first understand the operation that has given name to the network, namely convolution. In mathematics a convolution is an operation on two functions that produce a third function which in our case is the sum of the point-wise products of two matrices. Figure 2.2 illustrates a 3x3 data matrix being convoluted using a 2x2 filter. To produce the result the filter is slid across the input left to right, top to bottom. In the same order, the result matrix is populated with the sum of the point-wise multiplication in each step. For simplicity's sake the values in the example are either one or zero, but in reality they could be any values. Another aspect of convolutions yet not mentioned is the filter's step size, known as stride. The example demonstrates a convolution with a stride of $(1, 1)$. This means that horizontal steps and vertical steps will both result in a one cell shift of the filter.

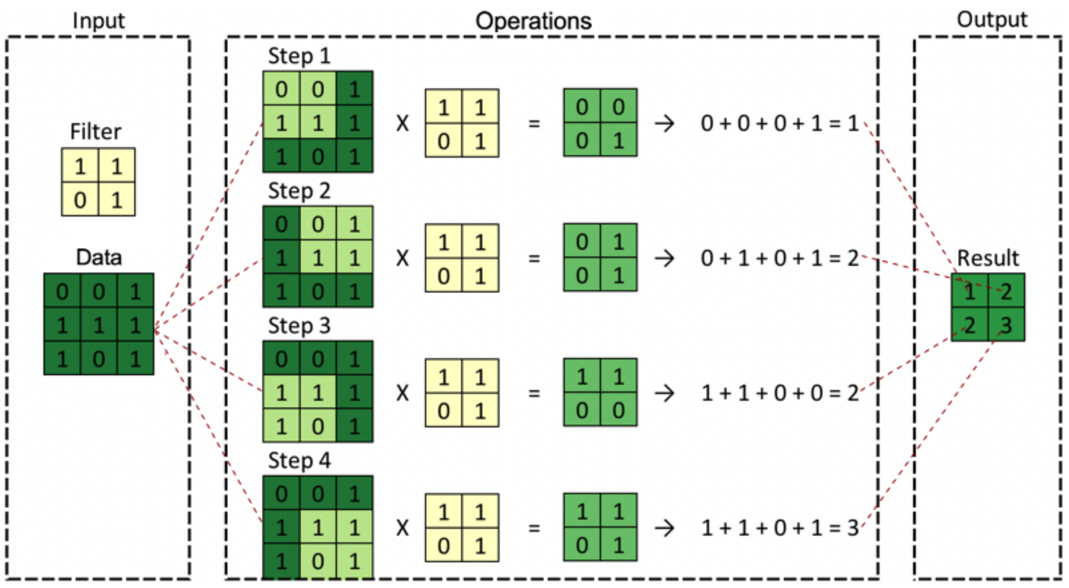


Figure 2.2: Illustration of a convolution of a 3x3 data matrix using a 2x2 filter.

Training

A CNN trains through supervised learning, meaning each data input has been labeled with the correct class beforehand. Figure 2.3 illustrates the feed-forward process of a simple CNN used for image classification. This CNN uses two convolutional layers, two sub-sampling layers, and one fully connected layer, the squares depicted in the figure represent the output of each layer. The input data is run through the network and the error between the output and the correct class is calculated. The network then tries to minimize the error by adjusting the filter values through back propagation.

The filters in each convolutional layer start out with random weight values taken from a specified distribution. The filters will during training specialize in recognizing different patterns, and will together get better at recognizing the whole image. After this another convolution is done, producing a second set of feature maps, at this point the simple patterns learned previously may be combined to form more advanced patterns. Finally there is another sub-sampling before the fully connected layer which reduces the amount of parameters to a probability distribution array over all possible categories. For instance, if the network in the figure has been trained to classify images of dogs, cats and penguins, and an image of a penguin is processed, the position in the output corresponding to a penguin should be as high as possible, and the other two as low as possible.

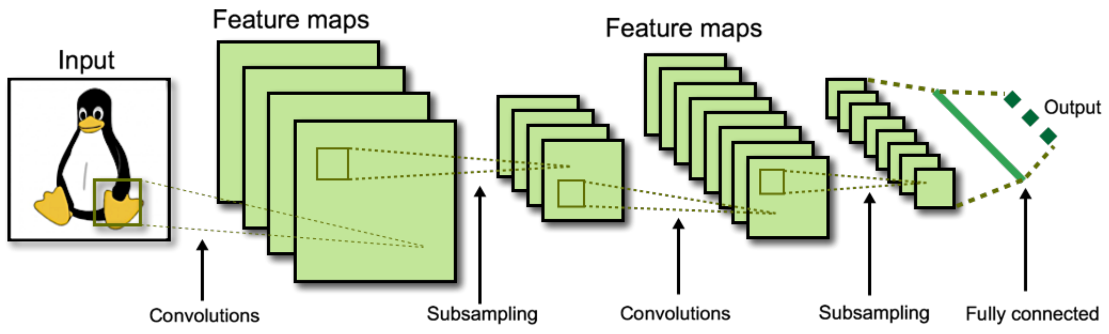


Figure 2.3: Simple CNN for image classification.

2.2 Generative Neural Networks

GNNs have seen a significant upswing in popularity during the last five years, as a demonstration: a keyword search for “Generative Neural Network” on Cornell University journal archive [29] results in 6,154 abstract matches for articles published in the field of Computer science from 2014 to 2019, compared to only 218 matches in the period 2000 to 2013.

Though machine learning has proven to be successful at solving non-linear problems, generating something new, or creating computational creativity is an important mile stone within ML research [3]. In our specific case this is not required, a simpler simulation of creativity based on observations and random input will suffice. Presented below are three different generative approaches we have explored along with some of their potential advantages and drawbacks.

2.2.1 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a form of neural network that during new predictions considers previous predictions made [9]. In other words the output of the network y_t depends on the previous output y_{t-1}. RNNs have shown to have incredible potential when generating text [28]. If we consider the wardrobe as a sequence of products it could be possible to design a model generating complete wardrobes in a similar way as text. This does however pose one immediate problem. RNNs build sentences by looking at previous characters and selecting the most likely character to follow. The RNN is confined to select characters from a determined vocabulary, in the case of text, this would be the alphabet and some special signs. If we were to build a vocabulary based on products it would become too large to work with. Consider each of our one thousand product ids being be placed at hundreds of different positions, this results in a vocabulary of size close to $\binom{1000}{500} \approx 2.7 \cdot 10^{299}$ which is out of the question for obvious reasons.

2.2.2 Auto-encoder

Auto-encoders were first mentioned in an article analyzing back-propagation within neural networks by Rumelhart et. al. [22]. Auto-encoders are ANNs aimed at learning how data is represented, traditionally with the goal of data compression. Auto-encoding networks operate similarly to GANs as described in section 2.2.3 but with the two networks individually reversed. The first network receives an image as input and is trained to construct a smaller representation similar to the latent z-vector of a GAN. The second network takes the latent variables as input and is trained to reconstruct the image as close as possible to the original. Recently in the machine learning field, auto-encoders have been used for generative and manipulative purposes such as Auto-Encoding Variational Bayes (VAE) proposed by Kingma et al. [14]. The introduction of VAE made it possible to sample the generation from a Gaussian distribution as well as a latent vector and thus allowing generation of new data.

Auto-encoders train using a loss function directly comparing the output against the original image, this makes the training more straight forward compared to a GAN. This effectively means that the output of the trained network is more comparable to a reconstruction rather than a generation. In our situation this could be suitable for suggesting additional products to an existing wardrobe, but it does not adequately fit our project description of generating complete wardrobe configurations.

2.2.3 Generative Adversarial Networks

Generative Adversarial Networks or GANs were first proposed by Ian Goodfellow et al. in 2014 [8] and have since become popular in the field of image generation. A GAN is built by letting two neural networks learn by competing against each other. One network will act as a generator, trying to create solutions to the problem at hand. The generator will be pitted against a discriminative network, which purpose is to debunk the generator's output. The only feedback the generative network receives is whether or not the discriminator thought it was real or fake, and thus, is never directly connected to the training data. The discriminator on the other hand trains by looking at both training data and the generator's output, and tries to master telling the two apart. In other words, the generator's job is to fool the discriminator and the discriminator's job is to not let itself get fooled. The discriminator is in practice a convolutional neural network, trained to classify input in two categories; real and fake. The opposing generator is a transpose convolutional network (sometimes referred to as deconvolutional). A transposed convolutional network receives a small input, and through the use of filters, transforms it to an output of larger dimension. In the case of the generator within GANs this means that the generator would receive a vector of random numbers generated from an arbitrary distribution, which is referred to as a latent z-vector. The generator's process, from latent z-vector to output dimension matching the data is illustrated in figure 2.4. When the network has finished training, the generator is separated from the network and used independently. The generator will be able to create output mimicking the patterns of the training data, without being fed the training data as a baseline. This setup would arguably allow for the generator to find new patterns acceptable to the discriminator but not found in the training data. This property of generation based on randomness aligns with our scope of simulated innovation. Something noteworthy about GANs is that in order to get good results, neither part should overpower the other, a generator that gets too good will always trick the discriminator and thus will not learn the actual patterns of the data. In contrary, an overpowering discriminator will never approve the generator's suggestions, and the generator might struggle to identify which features are sought after.

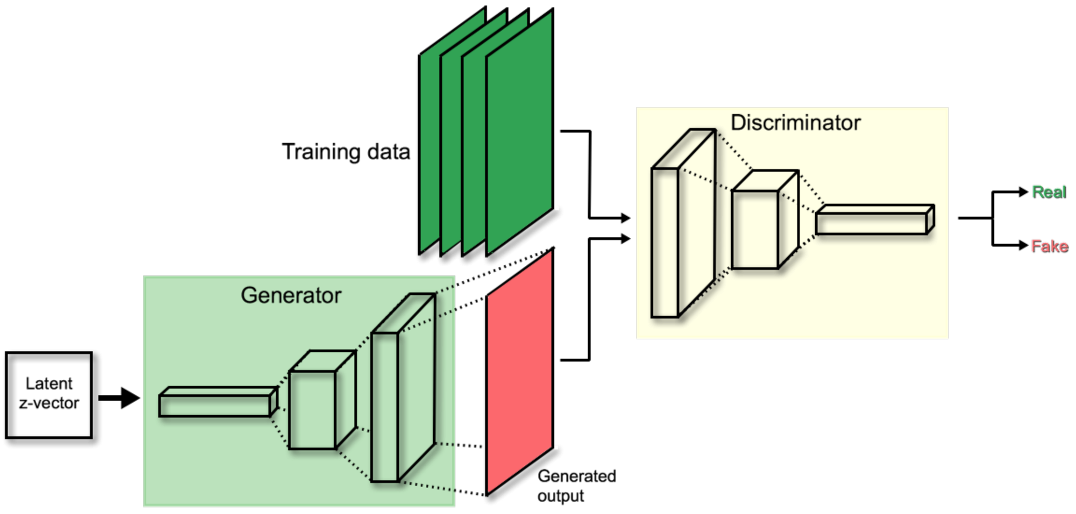


Figure 2.4: Illustration of a Generative Adversarial Network (GAN)

2.3 Related Work

Given the three approaches to Generative Neural Networks (GNN) the GAN architecture with its generative properties and simulated innovation most adequately fits our project description.

One of the most cited articles within the field of GNNs is the introduction of Generative Adversarial Networks (GANs) in 2014 by Goodfellow et al. [8]. Since then many other contributions to the field have been made, two of which have been especially influential to this study. The first of these is Deep Convolutional Generative Adversarial Networks (DCGAN), proposed by Radford et. al. which proved to be a strong candidate for unsupervised learning [21]. Radford et. al. used Ian Goodfellow's GAN design, and successfully applied a deep convolutional approach to both generator and discriminator. Their study showed that the inclusion of deep convolutional neural networks provided more stable learning. The second is the Auxiliary Classifier Generative Adversarial Networks (ACGAN) where Odena et. al. introduced a way to train deep neural networks to recognize classes in the input, and produce output within the same class [19]. This is closely related to what we want to achieve in regards to the product characteristics mentioned in the scope.

Both Radford et. al. and Odena et. al. train their networks using classified images, and on the surface, our matrix represented wardrobes share many structural similarities with an image. One of the largest differences however, is that images are represented by linear values as opposed to the discrete nature of our data.

Chapter 3
Method

This chapter starts with a description of how we processed and prepared our data before training. We then explain the design process of the network with motivations of the design choices made. Finally we present our network model and training setup, with an in-depth explanation.

3.1 Approach

First off we studied the data set and data structure to get a better grasp of the available resources. When we had settled on a representation of the data (this is further described in Section 3.3), we researched neural network solutions constructed for generative purposes. When a set of generative networks fitting the scope of our thesis were found, we evaluated an out-of-the box version of the networks in order to get a grasp of their possibilities. Comparing the networks and our scope we decided on the Generative Adversarial Network (GAN) model. Once the GAN architecture had been selected we derived an initial model fitting our data representation, based on an existing model of an DCGAN made by Radford et. al. [21].

When the initial model had been established we applied an iterative process of improving the model through the steps illustrated by the flowchart in figure 3.1. The first step after a modification had been made was to re-train the network. When the training was completed the results were evaluated through multiple criteria such as loss, training balance, visual appearance, and whether or not the results conformed to our scope (this is explained further in 3.4.4). If the modification was rejected, i.e. no improvement, the modification was not included in the core model and added to a backlog describing why and how it affected the results of the model. If the modification was deemed an improvement the changes were applied to the core model together with a description on how they improved the model. Whether the modifications were accepted or not meant either revisiting the previous problems or identifying new problems. These problems were compared with relevant research solving similar problems and investigated whether they could potentially be applied to our model. If such a solution was found the modification was included in the next iteration following the same procedure until we had reached a satisfactory state.

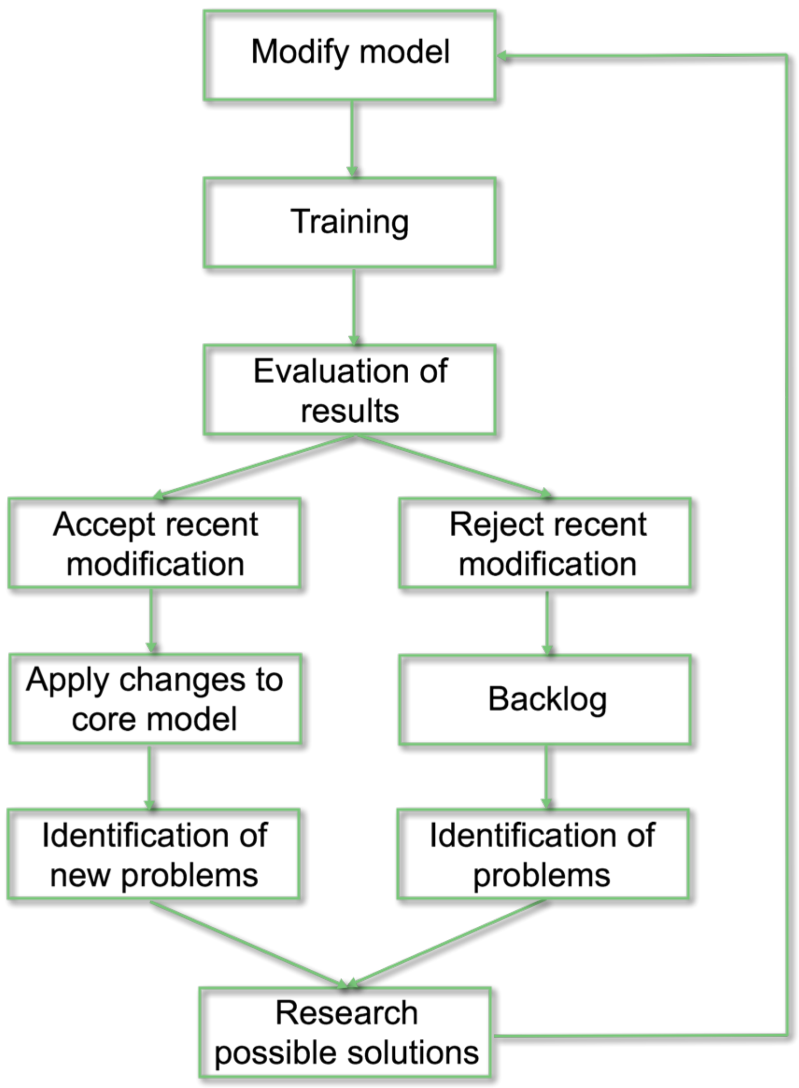

Modify model
Training
Evaluation of results
Accept recent modification
Reject recent modification
Apply changes to core model
Backlog
Identification of new problems
Identification of problems
Research possible solutions

Figure 3.1: A flowchart describing the iterative process of improving the network model. Each loop from the "Modify model" cell, to the "Research possible solutions" cell represents one iteration.

3.2 Tools and Technologies

For a detailed list of versions and specifications see appendix A. The entire project was coded in python using two environments, IntelliJ's PyCharm IDE and Jupyter Notebook [6]. The model design was developed in PyCharm while the data visualizations and analyses were performed in Jupyter Notebook. Additionally for data analysis we used the Tableau data analysis tool. The network model was built using the open source neural network high-level

API Keras backed by Google's TensorFlow backend [5], combined with specific functions written in TensorFlow. We used the following three python libraries during the project: Scikit-learn [20] which supplies pre-process functions such as value scaling. NumPy which is a massive high-level mathematical library allowing and improving operations such as matrix manipulation, interpolation, and much more. Pandas which supports manipulation of large quantities of data quickly and efficiently.

The training of the network was performed on a Macbook Pro in macOS, using an Intel Core i5 2.3GHz processor.

3.2.1 3D Model Construction

In order to be able to visually validate the generated wardrobes a 3D model program was provided by Company X.

The generator's raw output consists of numerical float numbers as seen in Appendix B.1. In order to visually analyze these results we rescaled them and remapped them from their encoded values back into their corresponding numerical and string values (not presented to avoid revealing internal data structures). This representation gave us insight into the validity of a generated product, but in order to get the most significant results we reconstructed them into 3D-objects. The 3D tool did not have any constraints on the input except valid product ids, which meant that all generated wardrobes could be analyzed provided a suitable product id was found. Using a product sheet provided by Company X, a decision tree for finding a product id based on a given set of attributes was created. The product id was then combined with the positions and offsets given by the generator and fed to the web server tool which constructed a 3D image based on the input. Since there was no guarantee that the generator would only output existing attribute combinations, a priority order was introduced, and product ids were found through a search using the following priority order: product type, width, depth, and color. However 3D-objects produced in this manner would not necessarily represent the generator's output, and whenever a mismatch occurred the wardrobe would be flagged as inaccurate. This proved useful during early stages of testing, but no such wardrobes were encountered when using the final model.

3.3 Data Processing

This section will explain what type of data we had access to, how the data was collected and how it was prepared for usage by our neural network model.

3.3.1 Data structure

The data received from Company X consisted of XML files representing purchase orders. The part of these files interesting to us is the product composition of the orders. Product relationships are represented following the JSON structure, each with an id and a position relative to its parent.

Based on the format of the data, three different ways to interpret a wardrobe came to mind. Option one is a list of products, where each product entry contains all the information about the product. The second option is a matrix where each row represents a product and each column a product feature. The result of this is that one matrix will represent a wardrobe in its entirety, much like the way a digital image is represented. The last option is a sequence of entries from a determined vocabulary. Each unique product, regarding all features will be given a specific identifier, and a wardrobe is represented by a sequence of these identifiers. This would be similar to how a word or sentence is built up by using the alphabet as vocabulary.

3.3.2 Data Collection

Company X was able to provide us with a data set of just above half a million sold wardrobe configurations. The data was scattered, unstructured and not very easily accessible. This resulted in spending a significant amount of time collecting and gathering data in a reasonable format. Since we were not completely sure of what type of data might prove useful we aimed to collect as many features as possible. The data received directly from our contact at Company X was a collection of purchase orders. The specific data about each order had to be individually collected by a direct request to one of Company X's servers. The response given contained a base64 encoded JSON file of the specific product ids and how the products were placed in relation to each other. At this point some important data about the products was still missing. In order to fill the gaps, each product was entered in a large lookup table based on a XML-file provided by Company X mapping product ids to product details. Finally all data was gathered in a single .csv file which served as basis for our training.

3.3.3 Data Analysis

Initially our plan was to create a solution that would generate wardrobe configurations of arbitrary size and number of products. Rather early on we realized the amount of extra work this would entail and we decided to limit our scope to generating configurations using exactly one frame and seven products. This specific setup was chosen since it was the most common occurrence in the source data 3.2. The limitation also facilitates the use of neural network models, since the input and output dimensions of these networks often are static. We also made a decision to view the data representing a wardrobe as a matrix with products as rows and features as columns. As previously stated in section 2.2.3, GANs have proven to be very successful at generating images. Thus our main reason for the matrix approach was the property similarities it has to such an image.

After settling on a matrix representation of the wardrobes we structured all the data into a table, each wardrobe bound to its products with a unique identifier. With the data structure in place we started with an exploratory data analysis using Tableau. During this initial analysis we found that, even using a limited amount of allowed products, each wardrobe is essentially unique. This is due to the continuous placement possibility of the products within a wardrobe, along with a large number of different products each with its own set of possible colors. This meant that simply representing a wardrobe as a collection of product identification numbers along with their placement, was both hard to represent structurally, but more importantly very hard to cognitively interpret. Instead we opted towards a more descriptive structure using the labels describing a specific product: type name, color, and width. This helped us understand the data, and products could still be uniquely reconstructed based on these parameters.

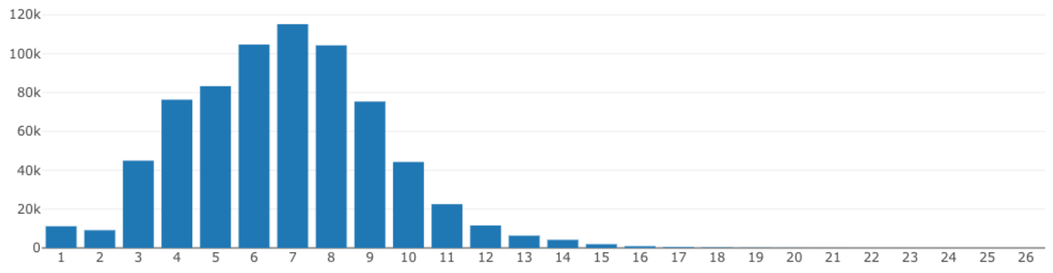


Figure 3.2: Histogram over number of wardrobes (y-axis) per number of products within them (x-axis).

As defined in the project scope we need a classifiable individualization metric, extractable from the data. After studying the data this left us with few options without requiring some manual classification and/or clustering of the wardrobes. Therefore the individualization metric chosen was the width of the wardrobe. The width is clearly stated and easily classifiable within the data set. Additionally it is distinguishable both from a visual standpoint, and speculatively from a "user need" point of view. There are three different widths possible for each wardrobe: 1000mm, 750mm, and 500mm.

Using these initial criteria as a basis, we began to analyze what products a wardrobe contains and their placement within the wardrobe. Illustrated in the heat map of figure 3.3, there is an uneven distribution of products, to combat this uneven distribution two approaches were used: Wardrobes containing sparsely represented products were either removed completely from the data set, or artificially inflated by adding copies of such wardrobes. As seen in the figure the product called *Divider* is missing completely from wardrobes of width 500mm, this is expected as there is little use in dividing such a small space. Additionally, frame dividers create their own subgroups of nestled products using the product *Section Shelf* with their own positional boundaries. Regarding the above stated arguments wardrobes containing frame dividers were removed completely from the data set. A similar treatment fell to the product *Valet Hanger* as the occurrence within the data set was deemed too low.

An attempt was made to up-sample the two most infrequent product types (*Multi Hanger* and *Trouser Hanger*), without further boosting the most frequent product types (*Cloth Rail*, *Drawer*, and *Shelf*). However such attempts were deemed unsuccessful as such a subset was far too small. The final result is a slightly less imbalanced distribution with no missing values as seen in the heat map of figure 3.4.

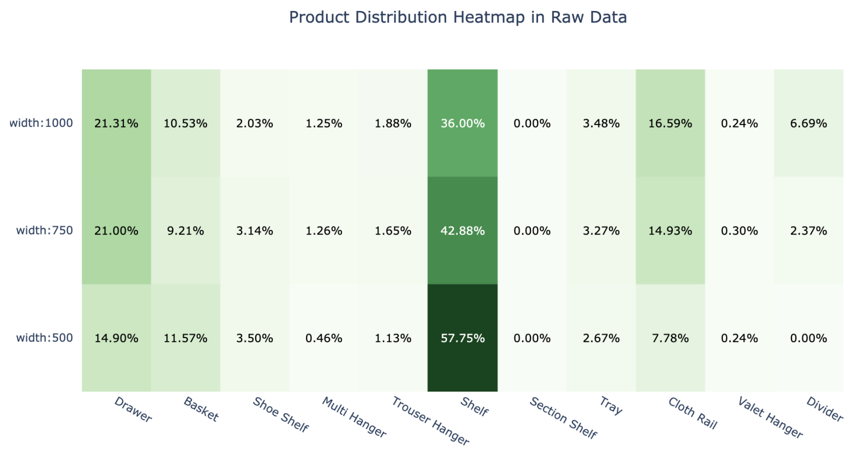


Figure 3.3: A heat map describing the representation of each product by name, within each width group. A high percentage is shown as deep green, and as the value approaches zero the respective square turns white.

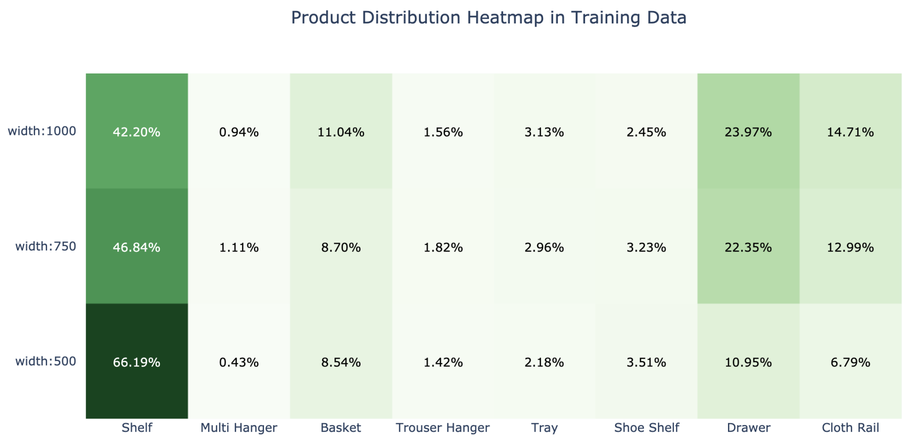


Figure 3.4: A heat map describing the representation of each product by name, within each width group. A high percentage is shown as deep green, and as the value approaches zero the respective square turns white.

Another defining characteristic of the end result is the positioning of each product within a wardrobe. Taking a look at the different height positioning of products, there are two products that stand out: the drawer and the cloth rail. The cloth rail has the highest density around the higher end of the spectrum together with a high density spike around the middle, as seen in figure 3.5. At the same time we see almost no drawers above the one meter mark in figure 3.6, as these would be out of reach for most people.

The 3D Tool used for rendering visual models of the wardrobes added an additional limiting factor for wardrobes with doors. Wardrobes rendered with doors, hid all products within. Due to this all doors were removed, which in turn lead to removal of all hinges, handles and knobs.

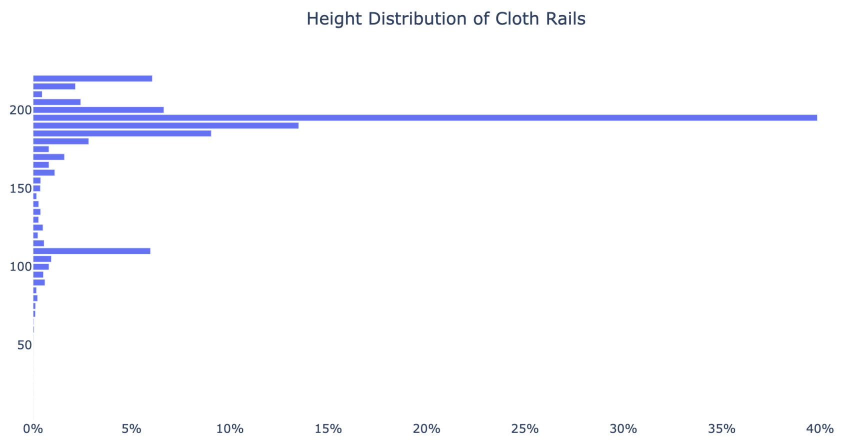


Figure 3.5: A horizontal bar graph describing the height (y-axis) distribution of cloth rails. Each height placement grouped within 5cm.

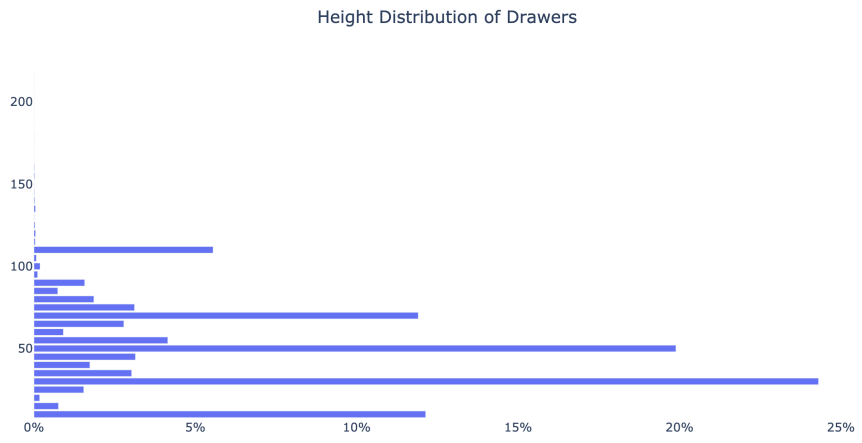


Figure 3.6: A horizontal bar graph describing the height (y-axis) distribution of drawers. Each height placement grouped within 5cm.

3.4 Model Design

This section describes how the model was developed. The model design conditions can be summarized as:

- Representing the wardrobe as a matrix.
- Using GAN as the architectural basis.
- Categorizing wardrobes based on width.

3.4.1 Network Design

One criteria for the solution was that the generated results should show innovation. This prevented our solution from being purely algorithmic or constraint based, since one input would either produce the same output every time, or be too random to consistently produce configurations of value. As mentioned in section 3.3.3, by structuring the wardrobe data as a matrix it gains properties similar to those of an image. Because of this we can adapt ANNs originally designed for images and apply them to our data set.

Using the design of Radford et al.[21] as a foundation we adapted their solution to fit our wardrobe matrices, and continued by adding other techniques to improve results. In order to fit our data the convolutional ratios of the model needed to be changed. We wanted to keep the ratios between the convolutions evenly distributed, i.e. based on the size of the network input each convolution would compress the 2D-space by the same factor. Both the generator and discriminator follow this design.

Following the approach of Radford et. al. we opted to implement our discriminator as an all convolutional model proposed by Springenberg et al. [26]. An all convolutional model does not have pooling layers, as opposed to the regular CNN for tasks such as image classification, described in section 2.1.2. To get the desired dimensions throughout the convolutions, a combination of kernel size (filter size) and strides is applied instead of pooling. However we did not see the same success using an all convolutional approach in the generator. Instead we used the more traditional combination of up-sampling and convolutional layer.

As defined in our scope the generated solutions should have the possibility of being tailored based on user input. In other words manual modification should change the properties of the wardrobe, e.g. from wide to narrow. Through labeling the data and combining regular classification with the GAN discriminator classification, Odena et al. propose an Auxiliary Classifier GAN (ACGAN) [19]. The ACGAN network changes the properties of a regular GAN network by introducing a discriminator optimized for both classifying the data as real or fake and assigning the correct category. Similarly the generator receives a categorical input in addition to the latent z-vector on which it should base the generated result. In effect this means that the input to the generator is distinguishable between categories, and can generate different solutions based on different categories. This was implemented as a weighted shift of the randomized z-vector distribution using an embedding layer. With this modifications our model can be described as an Auxiliary Classifier Deep Convolutional Generative Adversarial Network (ACDCGAN).

At this stage of the model the generator continuously overpowered the discriminator in an instance of over-fitting often referred to as mode collapse within GANs. Mode collapse occurs when the generator repeatedly tricks the discriminator by generating data within distributions where the discriminator is weaker. When the discriminator adapts the generator switches to another distribution. The result is that in any given state, the generator will most likely excel at generating output from one of the possible clusters, and fail at the others. To solve the problem of mode collapse the batch normalization technique developed by Ioffe et al. [11] was added to the generator model. Batch normalization normalizes the input to each layer in the network after each training batch. Additionally, with a lower risk of mode collapse the network should be able to handle a higher learning rate which as a result reduces the number of training epochs required significantly [11].

Another solution used to prevent over-fitting is the use of dropout in neural networks [27]. Dropout is a regularization technique used to force the network out of interdependent neuron connections, i.e. make individual neurons less reliant on specific outputs of previous neurons. Dropout was first implemented in both discriminator and generator, but our experiments showed that dropout in the generator had varied effect trending towards worse results. Since the generator already had the regularization technique batch normalization, we chose to only include dropout in the discriminator.

In an attempt to improve the convergence rate of the model using a technique referred to as instance noise by Sønderby et al. [25] was added. Instance noise should make it harder for the discriminator to separate between the real and fake wardrobes, i.e. making it easier for the generator. However this made the generator's convergence rate too fast, ending up overpowering the discriminator, and did not end up in the final design.

The activation functions used in all layers (excluding output layers) is the Leaky ReLU activation [18]. Leaky ReLU was chosen as it is empirically proven to be the better activation in convolutional networks [31].

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{3.1}$$

Leaky rectified linear unit (Leaky ReLU) applies linear function (i.e. multiplies by 1) on values greater than or equal to 0, and otherwise applies a multiplier alpha as seen in equation 3.1. The regular ReLU function works in the same way with the exception of $\alpha = 0$ thus every value below 0 is equal to 0.

Finally, since we want some features of the generated output to appear discrete a softmax function is applied. A softmax function is a squashing function which limits a set of values to be between 0 and 1, and the sum of all values to be exactly 1. The output from a softmax layer can therefore be interpreted as a probability distribution between the different features.

3.4.2 Constructing the Model

The final network model can be viewed in table 3.1 for the generator, and table 3.2 for the discriminator. The two networks follow a sequential design with each layer output being the input to the next, with the exception of generator input layers and discriminator output layers. The *Dense* layers are what Keras call fully connected layers. Note that many of these layers are activation or regularization layers and as such do not have weights manipulated during training.

The generator has five layers with trainable weights: *Embedding*, *Dense*, and three *Convolutional 2D* layers as seen in table 3.1. The first two convolutional layers are prefaced by

UpSampling 2D layers in order to up-sample the matrix dimensions. Most trainable layers are also followed by the Batch Normalization regularization layer, and the *LeakyReLU* activation layer. In order to achieve the classifier portion of the network the input to the generator is a combination of five layers, the first five in the table. First the category integer which is fed to the embedding layer. The embedding layer constructs a weighted representations of categories out of a maximum sized vocabulary. The output from the embedding layer is then flattened in the Flatten layer in order to remove redundant dimensions before multiplied cell wise with the latent z-vector input layer. The final *Convolutional 2D* layer is followed by a customized activation layer combining a tanh activation function and three softmax functions. The tanh activation scales data between $[-1, 1]$ and is applied to the y-position. The softmax functions are applied to the three feature attributes: type, color and width.

The discriminator has six layers with trainable weights: four *Convolutional 2D* layers, and two *Dense* output layers. All convolutional layers are followed by dropout regularization and *LeakyReLU* activation layers. The output of the discriminator is extracted by two different dense layers. Both dense layers receive their input from the *Flatten* layer above them, but compute it differently. The dense layer with output shape 1 is the layer responsible for classifying between real and fake data, and has a sigmoid activation as the output should be between $[0, 1]$. The dense layer with output shape 3 is responsible for the categorical classification and is followed by a softmax activation as the output should be a normalized categorical decision distribution.

Layer Type	**Output Shape**
Category **(Input)**	1
Embedding	1, 100
Latent **(Input)**	100
Flatten	100
Multiply **(Latent x Flatten)**	100
Dense	800
Reshape	2, 4, 100
UpSampling 2D	4, 8, 100
Convolutional 2D	4, 8, 50
Batch Normalization	4, 8, 50
LeakyReLU	4, 8, 50
UpSampling 2D	8, 16, 50
Convolutional 2D	8, 16, 25
Batch Normalization	8, 16, 25
LeakyReLU	8, 16, 25
Convolutional 2D	8, 16, 1
Custom Activation **(Output)**	8, 16, 1

Table 3.1: Layers used by the generator model.

Layer Type	Output Shape
Input	8, 16, 1
Convolutional 2D	8, 8, 25
LeakyReLU	8, 8, 25
Dropout	8, 8, 25
Convolutional 2D	4, 4, 50
LeakyReLU	4, 4, 50
Dropout	4, 4, 50
Convolutional 2D	2, 2, 100
LeakyReLU	2, 2, 100
Dropout	2, 2, 100
Flatten	400
Sigmoid Activation **(Classifier output)**	1
Softmax Activation **(Category output)**	3

Table 3.2: Layers used by the discriminator model

3.4.3 Training

As described in section 3.4.1 there is no clear cut way to design GANs, this includes the training setup. The training setup configuration was much like the model design an extensively iterative process through a mixture of grid search and manual search. Every training setup the model was saved with regular intervals to be able to analyze and compare the results retroactively between both different setups and different lengths of training (amount of epochs).

The final training setup uses the Adaptive Moment Estimation (Adam) optimizer [13]. The Adam optimizer is a type of Stochastic Gradient Descent (SGD) optimizer with a step size that decreases as the network approaches a minimum loss. The Adam optimizer has both been proven by research (e.g. Radford et al. [21]), as well as throughout our tests, as the suitable optimizer for our model.

The two networks are trained sequentially per batch, but simultaneously considering the entire training process. In every batch the discriminator was trained on generated results as well as real data, after which the generator was trained. During the training of the generator the two networks are linked together, as the generator itself has no concept of the real data. As such the back propagation runs through the entire network whenever the generator was trained, training both generator and discriminator.

The latent z-vector input of the generator consists of randomized values based on a normal distribution with expected value and variance as $z \in \mathcal{N}(0, 1)$. The categorical input to the generator during training was determined by a uniform randomization. The categorical labeling of the data was based on different aspects of the wardrobes throughout training iterations. The final training consisted of three labels describing the width: small, medium, and large.

Instead of optimizing the discriminator towards a strict binary value when classifying real wardrobes we used the one-sided label smoothing as proposed by Saliman et al. [24]. One-sided label smoothing means reducing the optimization value for the real data into something slightly lower (e.g. 0.9 instead of 1). The reduction of label intensity reduces the possibility of the discriminator extrapolating too confident classifications [7]. In effect when the discriminator classifies the real result too confidently it will be penalized which in turn gives the generator more leeway. This label smoothing was only performed on real versus fake classification and not the categorical labels. Additionally the label smoothing was not made while training the generator, as the generator's success is determined by the discriminator's output.

Instead of improving the convergence of the model via network design as mentioned in section 3.4.2, we attempted to improve the convergence rate by introducing a training centered technique we call one-sided label flipping. Label flipping means that in every batch the real versus fake label input to the discriminator was flipped with a low probability. One-sided meaning that it was only the generator's output, the fake data labels, that were flipped to real. The intention of this was to reduce the differences and divergence of the two data sources within the discriminator's interpretation. However this solution reached similar results as in the instance noise attempt: increasing the convergence rate by too much. As such one-sided label flipping was not included in the final training setup.

3.4.4 Evaluating Training Results

As mentioned at the start of this chapter, an iterative process was used to develop the training model by attempting to isolate the most prominent issues and counter them. This was done through continuous evaluation and analysis of the model output. In most ANNs the main factor of model convergence can be seen in the training progression values loss and accuracy. However GANs are two competing networks without a loss function of closed form, as such the loss values of the network does not follow the same principle as regular networks. I.e. even though the loss does not converge to zero, the two networks might still be learning from each other. By continuously saving the state of the model and taking samples of generated wardrobes at these points, we gained insight into how the model was evolving and could evaluate whether or not it was still improving. The generated wardrobes were primarily analyzed as raw data, but also by translating them to XML and examining the resulting wardrobes using the 3D tool. Through these methods it was possible to identify issues in an early stage of training, and determine whether the training cycle could be aborted due to patterns of instability.

Wardrobe attributes that were studied during evaluation were product relevance and positioning with a common sense approach as to how wardrobes usually look. A few examples of this are products that do not fit within the wardrobe frame, placement of cloth rails right above a shelf or pull out drawers placed too high to reach by an average person. While examining the wardrobes we also noted the variance, as in how different the generated wardrobes are to one another. Variance is an important metric since the model has to be able to produce wardrobes of different product configurations to fulfil the requirements of this project.

Finally we looked at confusion matrices of the discriminator's acceptance rate of both real and fake data. These results are arguably inconsistent with the networks performance, i.e. a low acceptance rate of fake data is not equivalent to the generator not progressing. However we considered the confusion matrix a relevant tool when analyzing the balance of strength between discriminator and generator.

Chapter 4
Result

This chapter will present the results of the study along with some of the authors' interpretations of them.

4.1 Training results

In our final solution we trained the network for 100 epochs using a batch size of 256 and saved the model weights every fifth epoch. We used three labels: small, medium and large referring to the width of the wardrobe frame. The loss for both discriminator and generator were saved after each epoch, and after the 100 epochs our model ended with a discriminator loss of 0.57 and a generator loss of 2.54, visualized in figure 4.1. After epoch 68 the loss of the generator started climbing rapidly while the loss of the discriminator declined. This behaviour occurs when the discriminator discovers a pattern that the generator has difficulties capturing. Further exemplified in figure 4.2, we see the differences in discriminator classifications generated and real wardrobes between epoch 65 and 100. During epoch 65 the discriminator has a harder time distinguishing between the two, as well as the peaks of both residing closer to 0.5. If the discriminator can easily distinguish between generated and real it could mean one of two things: either the discriminator progressed faster than the generator, or the generator's performance declined. The best possible end result would be the generator reaching a state of producing data of the same quality as the real data, giving us a complete overlap in the discriminator classifications. However if the discriminator reaches this point of classifying both data inputs equally would mean difficulties to progress further. In the end we would like a state where these graphs would have close, but not complete, overlap. As of epoch 65 the respective loss sits at 1.32 for the generator and 0.68 for the generator.

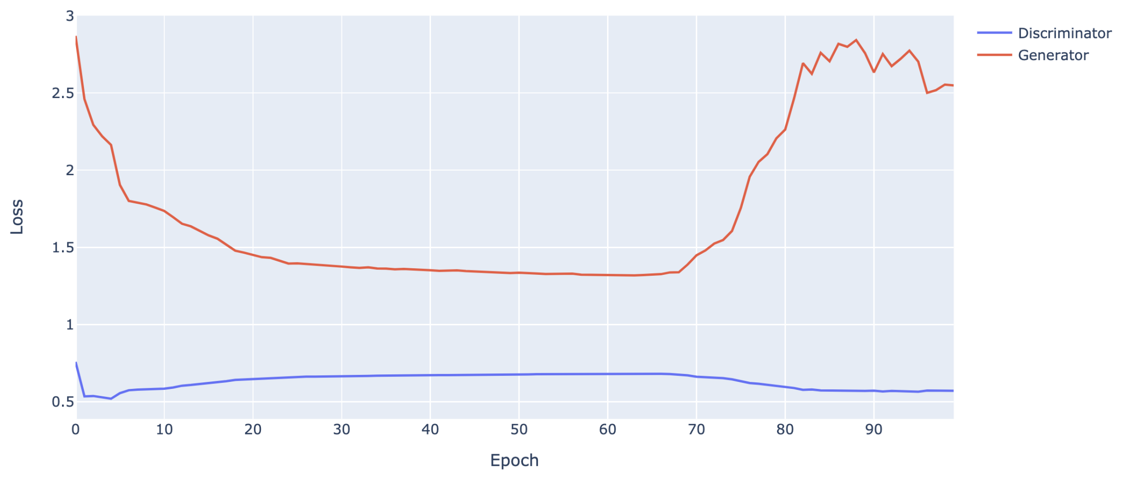
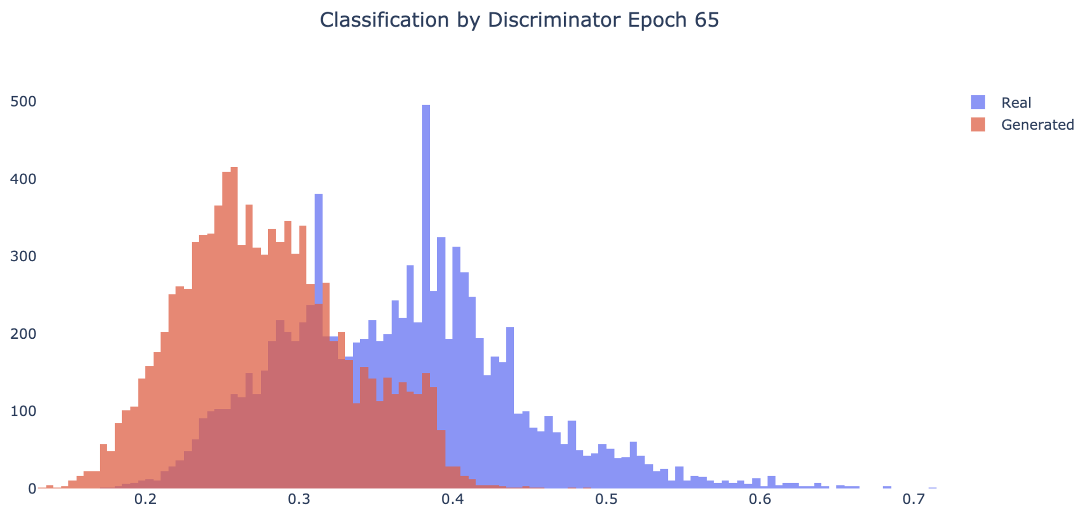


Figure 4.1: Graphs of discriminator and generator loss over epochs.

Figure 4.3 shows six examples of generated wardrobes using the model saved at epoch 65. These wardrobes follow the aforementioned characteristics of both variance and some degree of common sense product positioning. With this as basis the model from epoch 65 was selected as the final one.



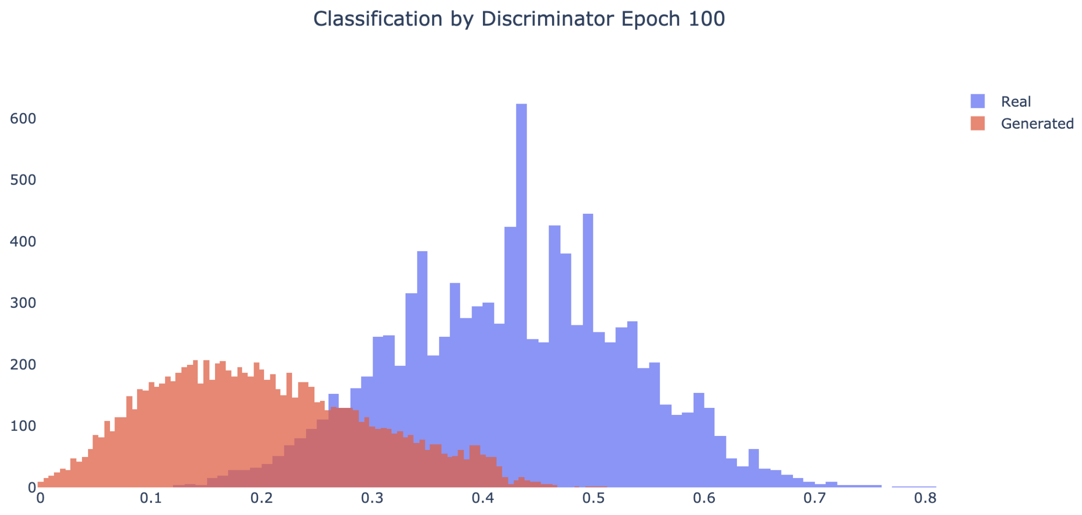
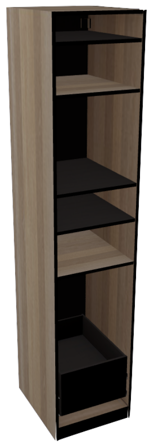
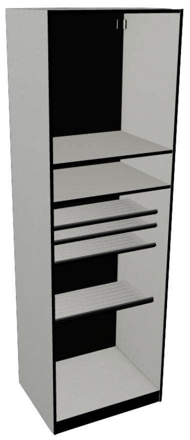
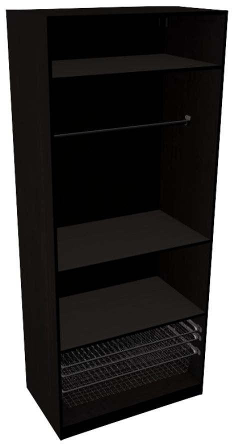
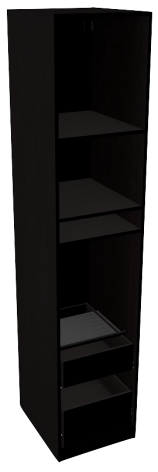


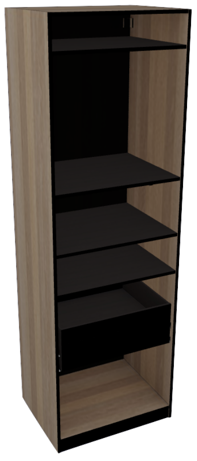
Figure 4.2: Histograms showing the discriminator classifications of wardrobes. The upper graph shows the output at epoch 65 having a higher overlapping density compared to the bottom one showing output at epoch 100.


(1) Label: small.


(2) Label: medium.


(3) Label: large.


(4) Label: small.


(5) Label: medium.

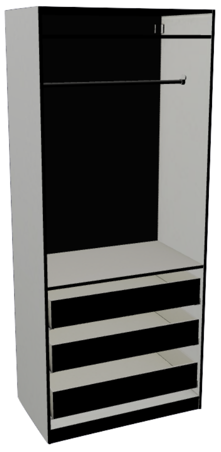

(6) Label: large.

Figure 4.3: Examples of generated wardrobes, rendered using 3D visual tool. The label shows the categorical input given to the generator.

4.2 Quantitative Results

As described in section 4.1, the final model used is the model trained over 65 epochs. For the quantitative results we generated a sample size of 10000 wardrobes evenly distributed between the three categorical sizes: small (500mm.), medium (750mm.), and large (1000mm.). The size of each generated wardrobe was compared with the expected wardrobe size based on the generator's input. Results show that **100%** of the wardrobes generated based on input category small, produces wardrobes of size small. The other two sizes show a slight bias towards smaller sizes (figure 4.4).

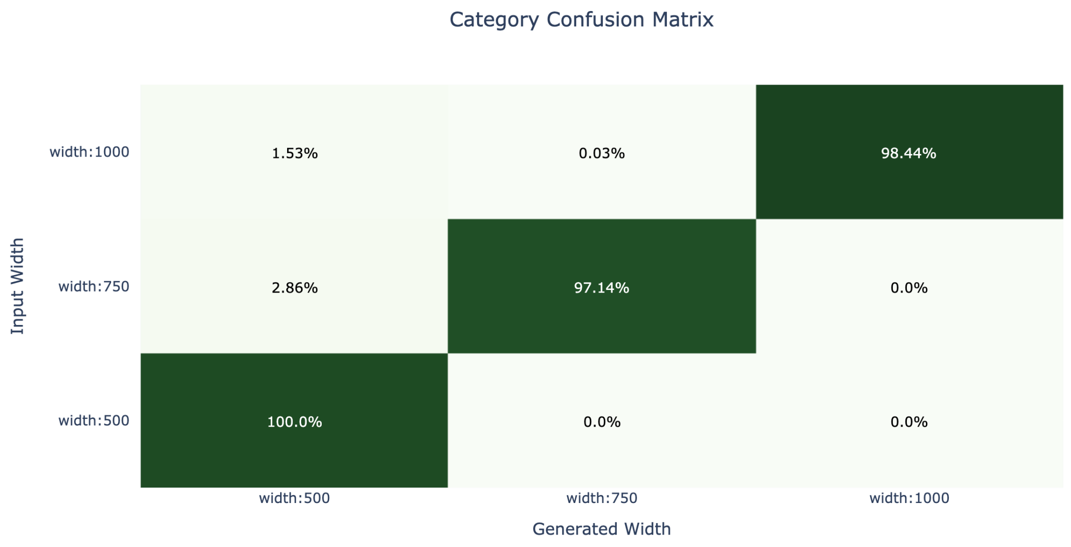


Figure 4.4: Confusion matrix showing input wardrobe size versus actual generated wardrobe size. Input consisted of 10000 generated wardrobes, evenly divided between the size categories: small, medium, large.

During the analysis of the training data Section 3.3.3 a concern arose regarding the low occurrence of a handful of products. Such cases can cause machine learning models to bias towards the more frequently occurring cases and potentially completely disregard some of the training data. However the heat map in figure 4.5 shows us that nothing is entirely omitted. In fact, for wardrobes of the smallest size the rarely occurring products have increased in frequency (figure 4.6). This means that there is an area reserved in the latent space for even the most rarely occurring combinations. If we were to instead bind the categorical input to e.g. specific products the generator could be told to generate such combinations on demand.

Described in section 3.3.3, drawers and cloth rails have specific positional rules. In figure 4.7 we see the comparison between training data and generated data positional distributions for cloth rails. The most frequent position for cloth rails in generated wardrobes is the same as in the training data (195 cm.). Although the distribution follows a similar shape, it has been smoothed out. Now the top peak represents **16%** of the generated cloth rails compared to the previous **40%**. Additionally the secondary peak at 110 cm. is now lost in the downwards trend of less frequent cloth rails closer to the ground. The positional distribution of drawers (figure 4.8) follows a similar pattern with spikes smoothed out into a more even distribution.

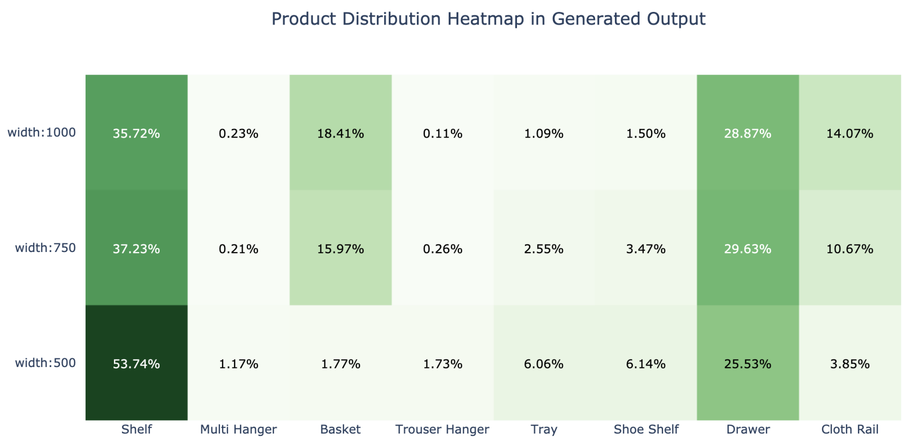


Figure 4.5: A heat map describing the distribution of each product, in the generated wardrobes, by name and width group.

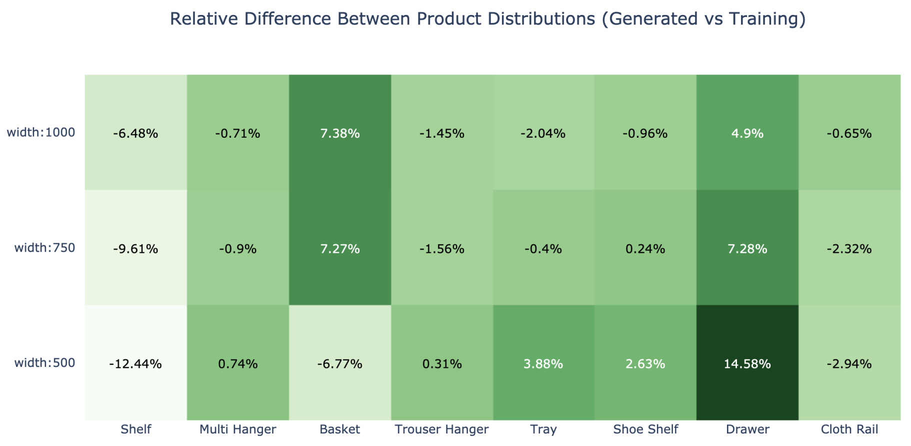


Figure 4.6: Shows the difference in distribution between the training data in 3.4 and generated data in 4.5. Numbers represent the percent unit difference calculated as $generated - training$

The two peaks of the training data are at 30 cm and 50 cm, which is also where the peak of the generated distribution lies. Worth noting is that, while hard to see in the visualizations, both cloth rails and drawers cover the entire positional y-axis (i.e. no position on the y-axis has 0% occurrences), though occurrences of drawers in high positions and cloth rails in low positions are very rare. Based on this we can conclude that the generator has noticed, and is generalizing towards, the positional rules of cloth rails and drawers, although not capturing the discrete peaks of prioritized positions.

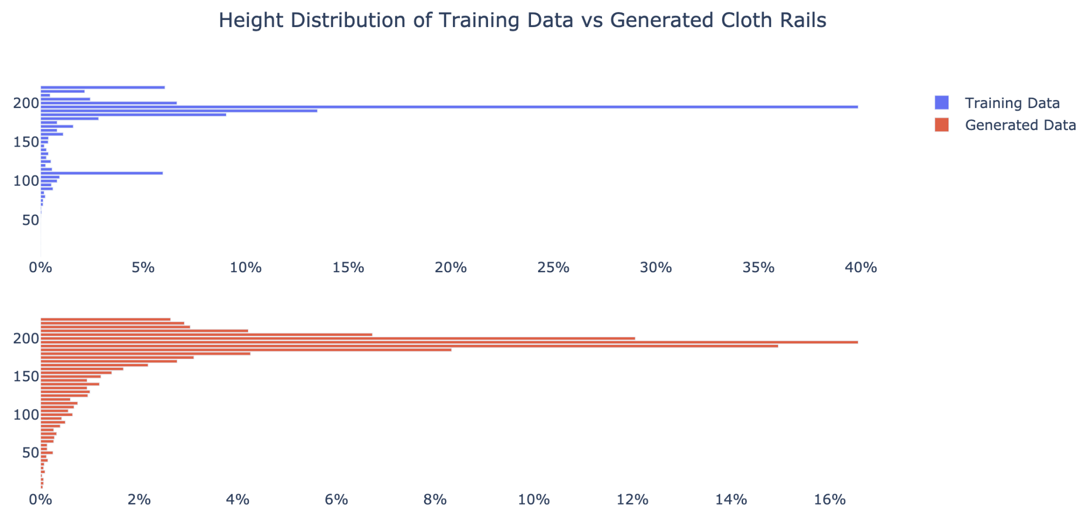


Figure 4.7: A horizontal bar graph describing the height (y-axis) distribution of cloth rails in training data (blue) and generated data (red). Each height placement is grouped in 5cm intervals.

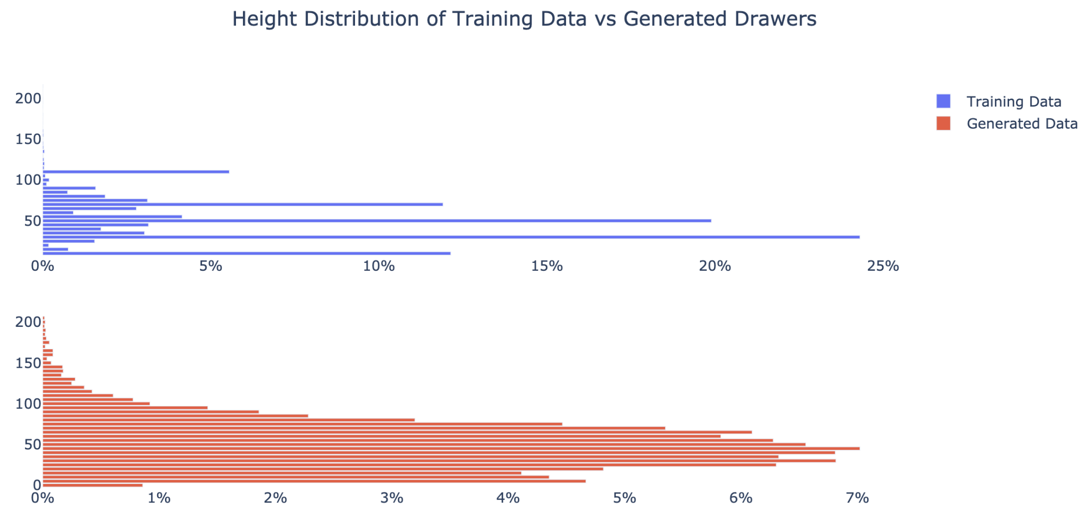


Figure 4.8: A horizontal bar graph describing the height (y-axis) distribution of drawers in training data (blue) and generated data (red). Each height placement is grouped in 5cm intervals.

4.3 Qualitative Results

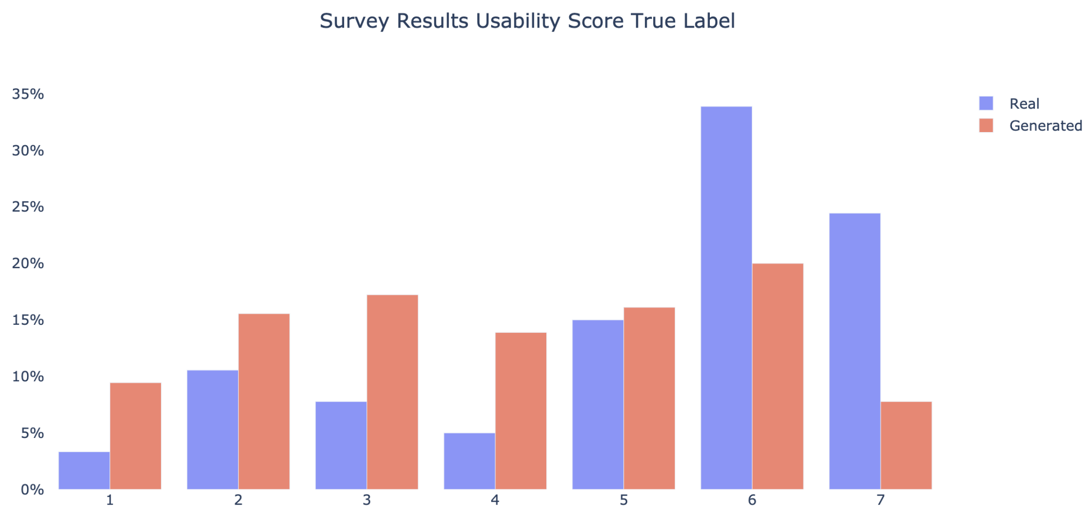
Using the same generator as mentioned in section 4.1 we generated some wardrobe samples of different specifications, the specifications differ depending on what label was used during generation i.e (small, medium, large). The output was converted to XML and loaded to a tool provided by company X which allowed us to produce 3D images of each wardrobe (see figure 4.3 for an example collection of such images). Using 20 generated images together with 20 wardrobes taken from the training set we constructed a survey where 9 subjects answered the following two questions about every wardrobe:

- Do you think the wardrobe is generated or real?
- How usable do you deem this wardrobe on a scale 1 to 7?

Figure 4.3 represents six of the generated wardrobes used in the survey and an example of the survey questions can be seen in Appendix B.2. It is also worth noting that the survey participants were not aware of the 50/50 split between generated and real wardrobes.

Comparing the two figures of usability scores between true labels and guessed labels (figure 4.9), we recognise a bias; wardrobes seeming usable were more likely to be considered real. Due to the survey structure of combining both usability score grading and guessing if a wardrobe is real or generated, this behaviour was expected. However according to the survey results from the true label graph relatively many of the real wardrobes (training data) exist on the lower end of the scale, emphasising the need for more rigorous data processing. Close to the end of the project we found erroneous data in our training set where the faults range from minor illegal product positioning to whole wardrobe configurations being invalid. One could argue that a few of these errors should not have an impact on the overall results, but when they become frequent it becomes a serious issue. In order to mitigate this a lot of effort was directed towards finding such faulty entries while cleaning the data, but without manually checking every wardrobe there is no way to prove that the data set only contains suitable entries. Our suspicion that the data set still contained faulty wardrobes was further enforced after analysing the results from the survey presented in 4.10, where almost 42% of the real wardrobes were marked as *generated* by the survey participants. Due to this we believe that our results would look better if the whole training set was manually verified, regrettably we did not have the time to do this. Regardless of these problems with the training data, the survey results exceeded our expectations in terms of positive results for several of the generated wardrobes. Out of the 20 generated samples, 5 of them were given a score of 7 in at least one of the responses. In addition to this the survey participants categorized almost 32% of the generated wardrobes as real, we consider this a success as only 58% of the real wardrobes were attributed as such.

Figure 4.11 shows three examples of some of the worst performing wardrobes from an average usability score perspective. In the first two images we see a similar trend that occurs in wardrobes from both the training data and generated collections, an impossible stacking of products. In the training data wardrobe (1) it is less noticeable as an image, but upon close inspection we see that multiple products are placed on the same height. If such occurrences are frequent enough it will have an impact on the generator, which we believe the worst performing generated wardrobe (2) is an example of. Comparatively there are examples only found among the generated wardrobes in the style of the rightmost wardrobe (3).



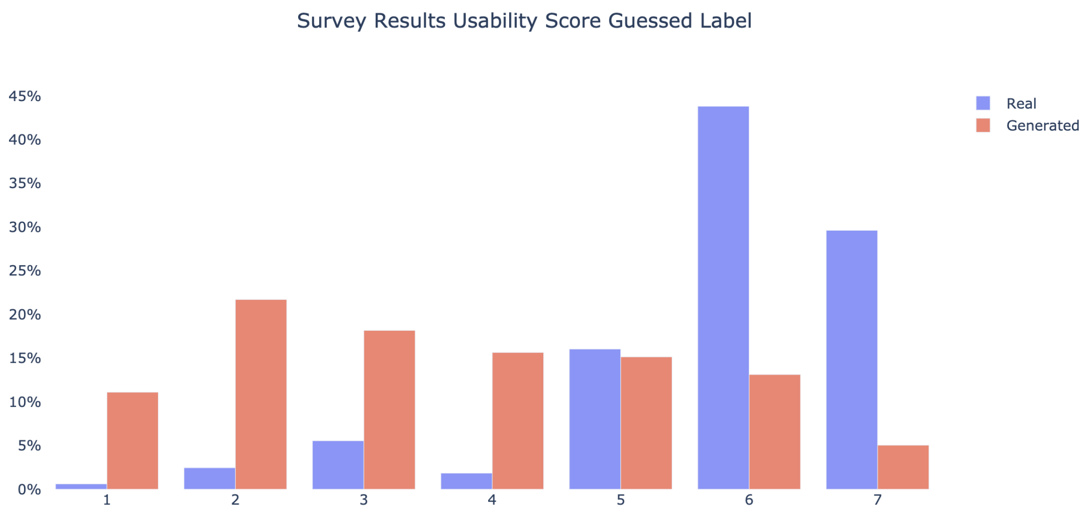


Figure 4.9: Two bar graphs visualizing the distribution usability scores derived from the survey responses. In the survey a score of 1 was referenced as "Not usability at all in current state", and a 7 as "Realistically usable in current state". The upper graph is built measuring the usability against the true label (generated/real) and the bottom one is measured against the guessed label.

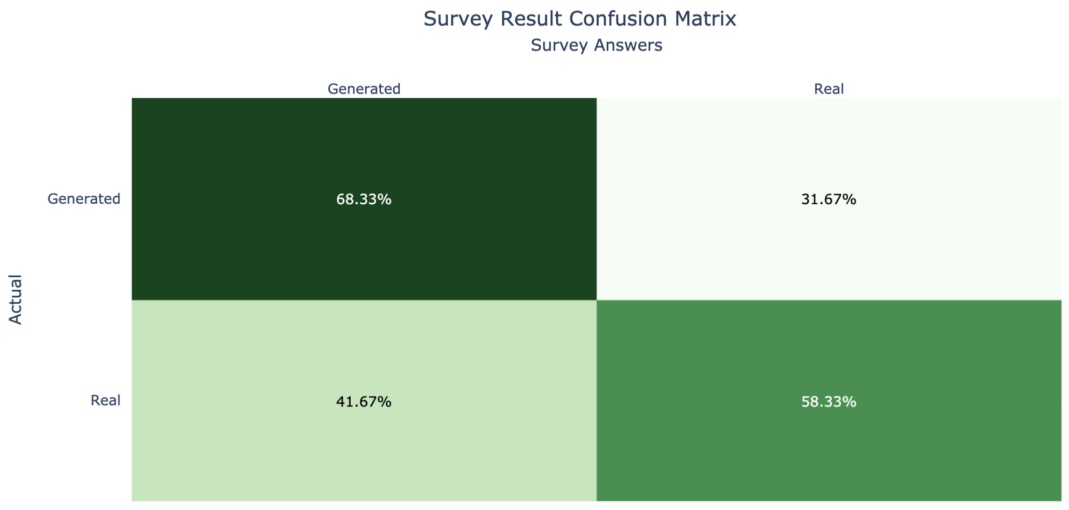
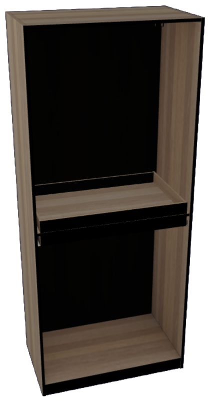
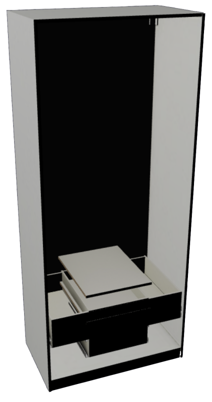


Figure 4.10: Confusion matrix visualizing the distribution of survey answers. E.g. the number in the bottom right corner represents that 58.33% of the real wardrobes were also flagged as real.

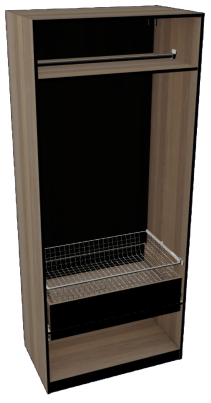
More specifically at the top of the wardrobe there is a shelf beneath the cloth rail rendering traditional use of the cloth rail impossible. This flaw can be viewed in two ways: the result is almost correct having only missing a few points in the y-axis or this mistake is grave because there is a shelf beneath the cloth rail. Our guess as to why this wardrobe received such a low score is due to the latter reasoning, that the survey participants focused on the impossibility of using the cloth rail. Our ambition with the convolutional design of the model was that such conditions would be captured but evidently the precision is not quite there.


(1) Wardrobe from training data.

Average score: 2.9


(2) Generated wardrobe.

Average score: 1.3


(3) Generated wardrobe.

Average score 2.7

Figure 4.11: Examples of generated wardrobes, rendered using 3D visual tool. The wardrobes presented are ones that received some of the lowest average scores from the survey.

Chapter 5
Discussion

This chapter will discuss our results, our methods, and possible alternative solutions. It will also present future work recommendations for further exploration of the area.

5.1 Training the Model

One of the biggest concerns we have had during the project has been the difficulty that generative networks have with producing discrete data. The most prominent one is the structural differences between continuous and discrete data, and how the networks adjust themselves to minimize their loss. In our model's case this is performed by assisting the parameters a little bit at a time, to close in on the correct value. In the case of discrete data there is no space between two points in which to move, and the parameters have to jump all the way to the next available option. Available options for categorical features are unlikely to be optimally ordered. Consider the colour feature, an intuitive ordering could be to order colours from darkest to brightest, but the data shows that white shelves sometimes occur in black wardrobes. This means that in order for the network to decrease its loss, it has to make a jump to a point likely unreachable by the stochastic gradient decent, thus trapping the network in a local minimum. In an effort to mitigate this problem we tried several methods such as modifying learning rate, adding batch normalization and introducing dropout. These methods improved results and in the final model we have not encountered patterns of this issue. Additionally there is no sure way of verifying when the global minimum has been reached, which leads us to the next problem. GANs, opposed to many other neural networks, lack a loss function of closed form making it harder to know what changes might have a positive impact on the results. This drives development towards a trial and error tactic which requires many iterations with small alterations for each cycle.

The width categorisation using the auxiliary classifier component was introduced as a proof of concept to show that one can provide input to the generator and specify the scope of its distribution. ACGANs have shown promising results even though they have primarily been used to specify the output scope of image generation. The difference between an image of a house and of a lake are structurally a lot larger than the change of width in a wardrobe, but this study shows that they can perform well in other applications as well. Although the width was partially selected due to the ease of labeling, one could add labels for other traits and thus customise the generation based on specific demands.

The first metric we are studying to make sure the model is progressing correctly is the loss of the model. To reiterate in short, we want to progress towards a lower loss score as this means the model is getting closer in the predictions and learning. During training the generator had a sudden increase in its loss score (seen in figure 4.1). The explanation of such behaviour ties back to the competing nature of GANs, and that the calculation of the loss is tied to their adversary's performance. This builds on the same principles as the "Zero-Sum Game" theory. These behaviours are how the GAN models improve, and a sudden spike in loss for one party is not necessarily a bad thing, as long as it recovers. However, although the loss scores seem static throughout much of the training we still saw a clear improvement in the generated output. This indicates that smaller improvement spikes might still exist, though hidden in the plot due to aggregations of the loss scores.

All of this raises the question of: when do you stop the training of the model? Due to time and hardware limits we were unable to continue the training longer than the 100 trained epochs. That brought us to a decision between the most trained model and the "best performing" model loss-wise. Optimally the generated wardrobes reach a state where they are indistinguishable from the real wardrobes. During training, this is established by the discriminator which poses a problem, since if such a state is reached before the generator has reached a hypothetical best state the adversarial model is unable to further improve. As mentioned in 4.1 what we are searching for is a state where the wardrobes are close to, but not completely indistinguishable from the training wardrobes from the discriminators perspective. This does however not mean that a human will be able to tell them apart, which is the actual end goal.

5.2 Data processing

As mentioned in the introduction the data provided by Company X was derived from digital wardrobe configurations assembled by an expert and sold. We had therefore assumed that the provided wardrobes would follow the business rules of product placement and usage. As proven by the survey, and mentioned in section 4.3 this was not the case. Even though the received data was analysed in its raw state, and several steps were taken to prune wardrobes not following business rules, a substantial amount of invalid wardrobes remained in the training data set. One of the most noticeable issues is the stacking of products, where all products in a wardrobe are placed on the same height. We encountered wardrobes with this problem while studying the generated wardrobes, but attributed it to mistakes made by the model, rather than underlying issues with the data. We did not discover the occurrences within the training data before the random sample was extracted and rendered to be used in the survey. Since we had used the building tool ourselves, we knew that the current version of it prevented this type of pattern from being made, and therefore we did not think of searching for them. Had we done that, these wardrobes could easily have been removed from the data set before training. In retrospect we should have identified the complete set of business rules posed on a wardrobe, and removed all wardrobes violating these rules. This would have guaranteed a data set void of rule breaking entries, which likely would have improved results.

5.3 Future Work

Considering the overarching goal of scalability and reliably being able to personalize solutions, we have not reached that point in the project's current state. We judge that a certain generalization has been reached (presented in the quantitative and qualitative results). Even so, the model does not produce functional wardrobes with high enough frequency and precision to be viable for the consumers market. At this point we have considered two options: supporting the model via a set of business rules to secure precision, or improving the model and sacrificing some precision for generalization. The choice between these two options is primarily a business decision, whether the precision is worth sacrificing for a higher level of personalization. We believe that the results are promising enough to be able to reach a level where a purely model driven generation can out-benefit the time consuming task of manually built example solutions.
As mentioned in the report our choice of using a GAN was primarily based on the problem description specifying a requirement of simulated innovation. This decision was made despite the difficulties GANs have to work with distrete data. Kusner and Hernández-Lobato [16] claim to have successfully mitigated the discrete data problem by exchanging the softmax output layer with a Gumbel-softmax distribution. The Gumbel distribution creates a continuous approximation of the discrete softmax distribution, which could facilitate the stochastic gradient decent. We believe that this would work quite well for our model as well, but we did not have the time to study this.

Another way of tackling the difficulty of generating discrete results the network could be modified into a semi-generative network. This could be achieved through letting an algorithm partially construct the wardrobe, while letting a neural network generate the innovative/creative aspects. Generative networks designed for the *"in-painting"* problem [32] could fit this description.

One of the main factors to the improvements between early and later iterations lay within the data modelling and not neural network modelling. One idea to take this even further is with a completely new data model, one that is less dependant on the generator building discrete positions from continuous numbers. Instead of a row in the matrix representing a product, the row position within the matrix could represent the position itself. I.e. each wardrobe matrix would have an y-axis with a representative number of rows corresponding to the max height of the wardrobes. The x-axis, similarly to the current data model, would correspond to a one-hot encoded product. The end result as such is a very sparse one-hot encoded matrix where most rows would just be flagged with the encoding for nothing, i.e. empty. This would result in the entire data model to be more image-like, which we have seen in previous work is the most successful field of GANs. Additionally such a model also allows for a variable number of products.

Chapter 6
Conclusion

In this thesis we have constructed a neural network designed for generating wardrobe configurations based on a collection of products. The results show that it is possible, but the quality does not meet the market requirements. We believe that the use of generative neural networks can prove useful for the task, but more time and research is needed in order to reach acceptable results. The results from the survey strongly suggest that the training data used was not of the expected quality, and given better data the model could have proven more successful. Thus we reached the conclusion that even though it is possible to generate adequate wardrobes using generative adversarial networks (GANs), it is unlikely that it is the optimal way to go. GANs have a lot to offer in the field of generative networks, but their inability to generate qualitative discrete data at this time cripples their performance for tasks such as the one in this study.

Bibliography

[1] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237. Barcelona, Spain, 2011.

[2] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jurgen Schmidhuber. Convolutional neural network committees for handwritten character classification. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1135–1139. IEEE, 2011.

[3] Simon Colton, Geraint A Wiggins, et al. Computational creativity: The final frontier? In *Ecai*, volume 12, pages 21–26. Montpelier, 2012.

[4] Richard D De Veaux and Lyle H Ungar. A brief introduction to neural networks. *Unpublished: http://www. cis. upenn. edu/˜ ungar/papers/nnet-intro. ps*, 1997.

[5] Martı́n Abadi et. al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[6] Thomas Kluyver et. al. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[7] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[9] Alex Graves. Generating sequences with recurrent neural networks. 2013.

[10] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[12] JSON. Introducing json. Last checked March 16, 2018. `http://json.org/`.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[14] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84, 2017.

[16] Matt J. Kusner and José Miguel Hernández-Lobato. GANS for Sequences of Discrete Elements with the Gumbel-softmax Distribution. *arXiv e-prints*, page arXiv:1611.04051, Nov 2016.

[17] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. June 2015.

[18] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[19] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org, 2017.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 2015.

[22] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[24] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.

[25] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. Amortised MAP inference for image super-resolution. *CoRR*, abs/1610.04490, 2016.

[26] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. 2014.

[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[28] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[29] Cornell University. arxiv.org. `https://www.arxiv.org/`.

[30] W3. Extensible markup language (xml) 1.0 (fifth edition). `https://www.w3.org/TR/REC-xml/`.

[31] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[32] Raymond A. Yeh, Chen Chen, Teck-Yian Lim, Mark Hasegawa-Johnson, and Minh N. Do. Semantic image inpainting with perceptual and contextual losses. *CoRR*, abs/1607.07539, 2016.

Appendices

Appendix A
Tools and Technologies

The following software, environments, tools and technologies were used during the project.

- Mac OS X 10.12.6
- Python 3.6.3
- Tableau 10.4
- PyCharm Community Edition 2017.2.4, Build #PC-172.4343.24
- Java Runtime Environment (JRE): 1.8.0_152-release-915-b12 x86_64
- Java Virtual Machine (JVM): OpenJDK 64-Bit Server VM by JetBrains s.r.o
- Jupyter Notebook 4.3.0
- Scikit Learn 0.19.1
- NumPy 1.13.3
- Keras 2.1.2
- Tensorflow 1.4.0
- Google Form

Appendix B
Code and Output Examples

B.1 Generator Output

Following is an example of how the direct output from the generator looks like, before it is scaled and converted to XML. Note that the 8 center columns are not displayed. Below is an image of the same wardrobe.

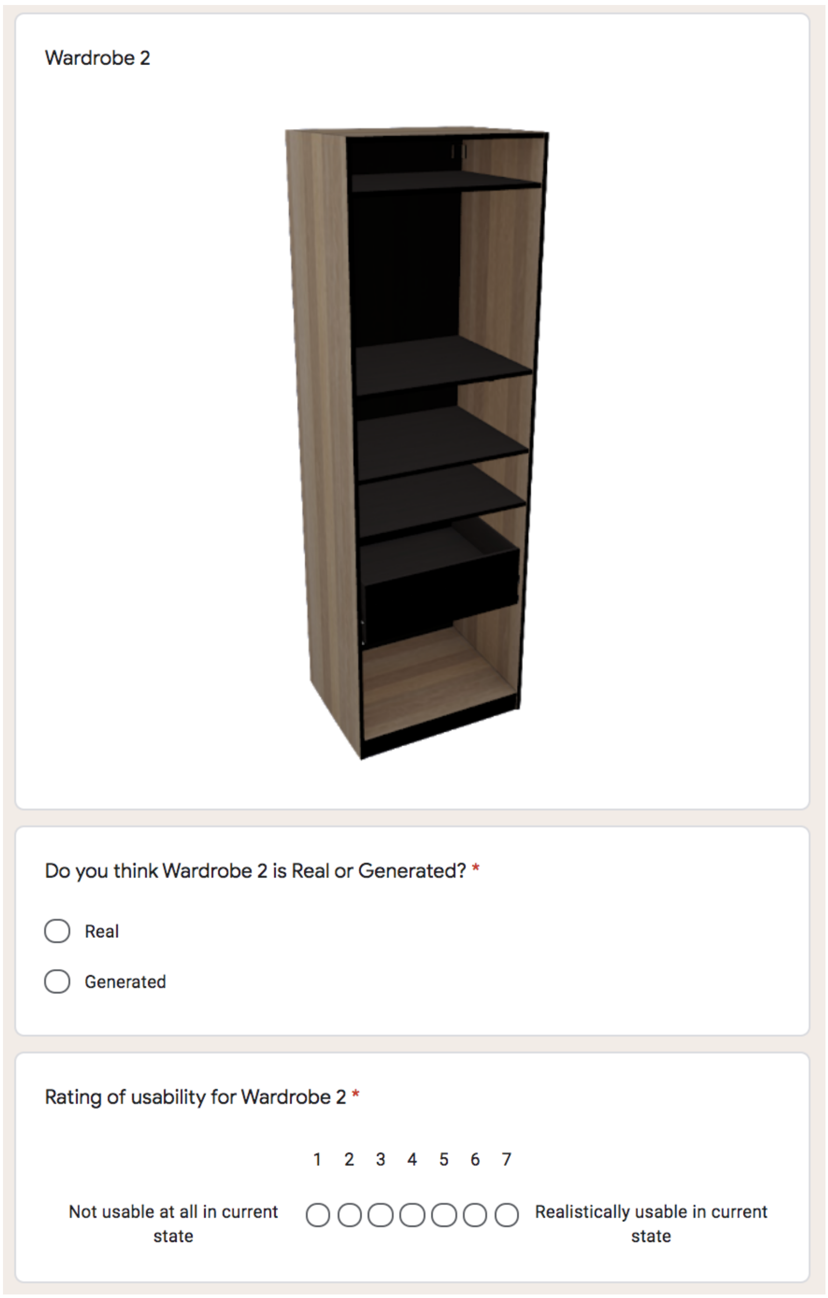
```
-3.71809602e-02, 4.87553303e-13, 7.87177079e-09, 1.29972889e-07, ..., 2.42082462e-01, 3.02922621e-04, 5.64710035e-06, 9.99691486e-01
8.48478615e-01, 3.97179721e-15, 1.46243229e-09, 2.50168094e-12, ..., 9.99999523e-01, 1.61330693e-03, 6.10423854e-08, 9.98386621e-01
8.04138362e-01, 2.87558130e-16, 1.00000000e+00, 1.76447927e-23, ..., 9.42171395e-01, 2.65818549e-08, 1.26187408e-10, 1.00000000e+00
-2.99196362e-01, 7.79346764e-01, 2.20653236e-01, 3.95262684e-16, ..., 9.90304530e-01, 3.10970347e-11, 2.82561084e-12, 1.00000000e+00
-8.79240751e-01, 2.19561975e-04, 3.75325340e-13, 9.99780476e-01, ..., 9.77258027e-01, 1.36921245e-15, 4.57425108e-15, 1.00000000e+00
-7.73638010e-01, 3.27341812e-19, 1.04511204e-31, 1.00000000e+00, ..., 1.22948829e-03, 6.63907940e-18, 1.31685542e-13, 1.00000000e+00
-5.70617080e-01, 2.74025242e-13, 1.09596401e-27, 1.00000000e+00, ..., 8.09822450e-06, 7.00269931e-17, 8.88199182e-13, 1.00000000e+00
-4.72170383e-01, 6.99883048e-16, 6.66205977e-27, 3.11586884e-13, ..., 8.92700627e-03, 7.43582539e-12, 1.47129219e-06, 9.99998569e-01
```

B.2 Wardrobe Evaluation Survey

Wardrobe evaluation
This survey was created by Oskar Holmberg and Erik Munkby as a part of a Masters Thesis in Computer Science at Lund University. The results from the survey will be used to evaluate the results of a Neural Network generating wardrobe configurations.

You will be presented to 40 images of wardrobes, some of which are generated by a computer, and some have been taken from a dataset containing wardrobes that have been sold in a store. For each of these wardrobes we ask you to:

- Answer whether or not you think it is a generated wardrobe, or one from the dataset (real)
- Rate the "usability" of each wardrobe from 1 to 7.



Wardrobe 2

Do you think Wardrobe 2 is Real or Generated? *

◯ Real

◯ Generated

Rating of usability for Wardrobe 2 *

	1	2	3	4	5	6	7	
Not usable at all in current state	◯	◯	◯	◯	◯	◯	◯	Realistically usable in current state

Figure B.1: Example of one of the entries from the form.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-06-01

EXAMENSARBETE Wardrobe Design using Generative Neural Networks

STUDENTER Oskar Holmberg, Erik Munkby
HANDLEDARE Elin Anna Topp (LTH)
EXAMINATOR Jacek Malec (LTH)

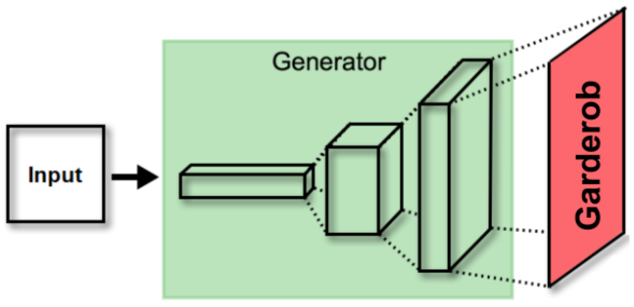
Skapa innovativa garderober anpassat till kundens behov med hjälp av Neurala Nätverk

POPULÄRVETENSKAPLIG SAMMANFATTNING **Oskar Holmberg, Erik Munkby**

En stor fördel med garderober i mindre delar är att de kan anpassas till att täcka ett särskilt behov. Att pussla ihop en anpassad möbel kräver dock en viss nivå av kunskap och erfarenhet som alla inte har. Genom maskininlärning kan vi träna ett neuralt nätverk till att ge förslag på installationer anpassade till uttryckta behov.

Företag världen över har aldrig suttit på så mycket information om sina kunder som de gör idag och alla letar efter möjligheter att använda informationen för att skapa mer nytta. Ett sådant exempel är ett möbelföretag som vill förbättra kunders upplevelse när de utforskar garderobslösningar på företagets hemsida. Utifrån data bestående av sålda garderobsinstallationer som skapats tillsammans med en expert har vi med hjälp av maskininlärning tränat ett neuralt nätverk till att kunna skapa förslag på innovativa installationer. Kunden kan också ange kriterier som input vilket garderoberna måste uppfylla och nätverket kommer att begränsa sitt skapande för att passa innanför dessa ramar. Detta ger kunder en inspirationsväckande upplevelse när de ska välja ut deras nya förvaringsutrymmen.

Ett intressant fynd under projektets gång var att nätverksarkitekturen som valdes, Generative Adversarial Network (GAN), fungerade så bra som den gjorde. GANs har huvudsakligen använts till kreativt skapande av bilder och musik, alltså kontinuerliga format. Det har däremot inte varit lika lyckat inom skapande av diskreta format så som text eller garderober. Generering baserad



på diskret data resulterar ofta i instabil träning, vilket inte ger särskilt bra resultat. Vi lyckades överkomma många av de problem som kan uppstå, och i en enkätundersökning tilldelades genererade garderober väldigt liknande betyg som faktiska exempel. Resultaten från enkäten antyder att att kreativ generering av diskret data med hjälp av GANs kan fungera väldigt bra. Detta betyder också att möjligheterna för att expandera från att generera garderober till andra liknande områden är stora. Att använda ett automatiserat system, istället för att manuellt bygga installationerna minskar arbetsbördan och kan samtidigt förbättra kundupplevelsen.