Generating African Inspired Fashion Designs

Lindiwe Malobola (452205)

Supervisor(s):
Dr. Shakir Mohamed
Dr. Negar Rostamzadeh

Research Report in Data Science
submitted in partial fulfillment of the requirements for the degree of Master of Science in the field of e-Science

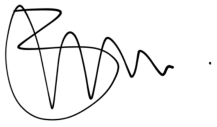
in the

School of Computer Science and Applied Mathematics
University of the Witwatersrand, Johannesburg

22 August 2022

Declaration

I, Lindiwe Malobola (452205), declare that this research report is my own, unaided work. It is being submitted for the degree of Master of Science in the field of e-Science at the University of the Witwatersrand, Johannesburg. It has not been submitted for any degree or examination at any other university.



Lindiwe Malobola (452205)

22 August 2022

Abstract

Fashion is one of the ways in which we show ourselves to the world. It is a reflection of our personal decisions and one of the ways in which people distinguish and represent themselves. Fashion is a form of cultural expression and story telling. In this research report, we focus on the fashion design process and expand computer vision for fashion beyond its current focus on western fashion. We discuss the history of Southern African Seshweshwe fabric fashion and the curation of a Seshweshwe dataset. In addition, we demonstrate an example use case of the Seshweshwe dataset: sketch-to-image translation task for affordable fashion-design. We use popular Pix2pix methodology to generate fashion images from sketches, evaluated using FID scores. The application of generative models to fashion raises both technical questions of training with small amounts of data, and also important questions for computer vision beyond fairness, in particular ethical considerations on creating and employing fashion datasets, and how computer vision supports cultural representation and might avoid algorithmic cultural appropriation. In this work we also propose responsible and proper ways of practice when dealing with a cultural fashion dataset.

Acknowledgements

I would like to thank the National e-Science Postgraduate Teaching and Training Platform for their financial support and all the support structures they have put in place for the success of this research. A special thank you to my supervisors Dr. Shakir Mohamed and Dr. Negar Rostamzadeh for their heartfelt guidance and support. Thank you to Tafara, my family and friends for their support and encouragement throughout the duration of this research in this difficult year.

Contents

List of Figures

Chapter 1

Introduction

Fashion has drawn a lot of attention from researchers in computer vision in recent years with a growing number of papers and workshops dedicated to this topic. There has been rapid development in fashion-related work ranging from retail sales forecasting [76, 79, 20], fashion trends analysis [10, 81, 31], fashion synthesis and recommendation [39, 8]. Fashion trends analysis often involves identifying patterns and predicting future fashion demands based on cities, season and runway fashion. Fashion image synthesis involves using generative models such as Generative Adversarial Networks (GANs) [27] and Variational Autoencoders (VAEs) [42] to generate new samples of fashion images. Fashion recommendation focuses on recommending clothing pieces or outfits given some conditions such as users' preferences, occasion and weather conditions.

This rapid development has led to a number of publicly available fashion datasets that are suitable for the development and application of machine learning [70, 47, 88, 85, 90, 48]. However, to the best of our knowledge, existing works on machine learning for fashion have not yet considered applications to African-inspired fashion and are limited to western forms of fashion. This limited diversity in existing data is linked to the general under-representation of non-western cultures and art forms, and contributes to the general harms that arise from perpetuating single stories about people and cultures.

In addressing this limitation our contributions in this work are:

- Firstly, we curate the first dataset that considers African fashion, in particular, the creation of a fashion dataset representing different designs of Southern African modern Seshweshwe fashion dresses;

- Secondly, we make initial contributions in the training of sketch-to-design GANs from small datasets, to explore the problem of generating African-inspired fashion from sketches;
- Thirdly, we explore important questions for computer vision beyond fairness, in particular questions of AI in relation to cultural representation and algorithmic appropriation.

1.1 Research Aims

The aim of this research is to generate images of African inspired fashion given design sketches, introduce the concept of *algorithmic cultural appropriation* and provide a guideline for prevention. This goal will be reached by training a generative model on images of African inspired fashion, discussing potential harms of *algorithmic cultural appropriation* and explore preventative measures.

1.2 Objectives

- Curate the first dataset that consists of African fashion, in particular, create a fashion dataset representing different designs of Southern African modern Shweshwe fashion dresses.
- Discuss ethical considerations in relation to AI, culture and fashion and propose responsible and proper ways of practice when dealing with a cultural fashion dataset.
- Demonstrate the usability and provide an example use case of our new Seshweshwe dataset by generating African-inspired fashion from sketches.

1.3 Research Questions

- Where do we situate machine learning for fashion in the history of cultural expression?

- can we train a GAN model on a very small dataset and still get reasonable translation results?
- Can we generate African fashion (Seshweshwe) images from line drawings using Generative Adversarial Networks(GANs)?
- What are the most significant parameters to pay close attention to when training a GAN on our custom dataset?
- How does the Pix2pixGAN perform under various conditions and paramenter values?
- What are the optimal learning rates for each of our models?
- What impact does the batch size have on the training efficiency of our models?
- Given the size of our dataset, how can we avoid over fitting on the training set?
- Are there any improvements in the images generated when we update the Discriminator more than Generator?
- How important is the choice of loss function and how the different losses affect model performance.
- How does the patch size of the discriminator receptive field affect our outputs?
- Where does the boundary of algorithmic cultural appropriation and representation sit?

1.4 Limitations

Due to the limited time and lack of readily available African fashion datasets, we limited the scope of our work to Southern African Seshweshwe fashion, which is only one of many cultural fashion forms in the Southern African region. Although we did a case study on Southern African Seshweshwe, the practice we introduce in our work is applicable to other fashion datasets. We also did not implement our models and adapt them to real fashion sketches.

1.5 Overview

In Chapter 2 we describe the background of Seshweshwe fashion and deep learning. In chapter 3 we contextualise our study and explain its relevance by providing an overview and discussion of existing work on Generative models for fashion. We detail the process of curating the Seshweshwe dataset that was used in this work. We also discuss ethical considerations in machine learning for fashion and art in general. Chapter 4 gives a detailed discussion of the Pix2pixGAN method and the results of its implementation on the Seshweshwe dataset. Finally, in Chapter 5 we provide a summary of our findings and the resulting conclusions .

1.6 Pronunciation Guide

We present a short guide on pronunciation of certain words used in the research report.

- se-Shweshwe is Pronounced: ci-sh-wesh-where.
- Batswana is Pronounced:Baat-swa-na.
- Basotho is Pronounced:Baa-soo-to.
- uMakoti is Pronounced: oo-mark-oot-ee.
- Moshoeshoe is Pronounced: mo-sh-wesh-where.
- Xhosa pronunciation: https://www.youtube.com/watch?v=Trq_gIe1v04.

1.7 Mathematical Notation

On all accounts, we adopt the mathematical notation specified in [56]. Vectors and scalars are denoted by bold and plain symbols respectively. A function f of variables $\boldsymbol{x}$ is represented by $f(\boldsymbol{x})$. p is a symbol for a probability distribution and $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ symbolises a distribution over some random vectors $\boldsymbol{x}$ with distributional parameters $\boldsymbol{\theta}$. The notation $\hat{\boldsymbol{x}} \sim p(\boldsymbol{x})$ represents a sampling of variables $\hat{\boldsymbol{x}}$ from a distribution $p(\boldsymbol{x})$. We use $\mathbb{E}_p[f]$ to denote the expectation of the function f under the distribution p.

Chapter 2

Background

In this chapter, the research is given context with a discussion of cultural fashion and deep learning concepts. In the first section we introduce real-world problems that the research is intended to address, the history of the Seshweshwe fabric and the state of South African fashion from a high level. In the final section, we detail fundamental concepts in Deep Learning as a tool for fashion generation in this work.

2.1 Cultural Fashion

Fashion represents one of the long-lasting modes of preserving and celebrating culture and history. In parts of the world, fashion is used to signify different social groups and status [16, 5]. In different parts of the African continent people use fashion to distinguish the many different cultural traditions [14, 3], and it is also used as a celebration of their unique histories [13], on special occasions, and on a daily basis for some. African traditional wear has had a large influence on global fashion brands, although often not acknowledged nor given recognition of its rooted cultural significance. Our focus in this paper is on the most popular Southern African traditional wear, Seshweshwe.

2.1.1 Cultural Fashion Industry in Southern Africa

We look at the fashion design process, which often begins with conceptual drawings. Fashion designers translate these concept drawings into a pattern, create a basic version of the garment with inexpensive cotton, and then finally samples are

made with desired fabrics. This process is costly and can stifle productivity in the cultural fashion sector where, in a country like South Africa, 70% of cultural fashion enterprises are informal [2] with an average annual turnover of R96,439[1] [75]. This poses a significant challenge for the cultural fashion sector to take advantage of the growing market in cultural fashion for everyday use [75].

In this work, we develop models that can be used as tools for generating African inspired fashion images from conceptual drawings. A sample generating system will help designers in informal markets communicate their concepts faster and secure orders from customers before creating a sample garment. This blend of product creativity and digital advancement can result in a strong competitive edge for the struggling South African textile industry as a whole.

2.1.2 Seshweshwe fabric and its history

In this research report we focus on the Seshwshwe traditional wear. Seshweshwe is a dyed cotton fabric characterised by elaborate flower arrangements, square, circular, stripes or diamond geometric prints. The fabric was originally dyed indigo but is now being produced in different colours. Seshweshwe is one of the fabrics that are central to Southern African traditional clothing. It is commonly worn at traditional ceremonies such as weddings and lobola ceremonies.

Historically in African culture Seshweshwe has been used to make several cultural apparels. These apparel range from aprons, skirts and dresses among others. The African cultures in question that mostly make use of the Seshweshwe are the Batswana, AmaXhosa and the Basotho. Seshweshwe is mostly worn by newly married women known as uMakoti. Further, in the Xhosa culture, the Seshweshwe has been embraced to be a part of ochre-coloured blanket clothing. The Herero women in Namibia use Seshweshwe to make some of their dresses [57]. In addition Seshweshwe is used in modern South African fashion clothing design for all genders from different cultures.

[1] Where R96,439 = US$6,441 on average

The Seshweshwe fabric has a long and rich history. The intellectual contribution can't be attributed to a single individual, but results from, and defines, the cultural evolution of specific groups of people. Seshweshwe's earliest origin can be dated all the way back to the trend of colourful, floral Indian cotton called Indienne that spread very quickly in the mid-16th century through Europe [58]. The cloth has been imported from Europe to South Africa. The trademarked cloth is being manufactured today by Da Gama Textiles in the Zwelitsha township in the Eastern Cape since 1982 [15]. Da Gama textiles has bought rights to the most popular brand called Three Cats.
Seshweshwe is currently a raw material in processes that produce clothes and accessories at large , medium and small scale. The Seshweshwe is specifically and predominantly made use of by small non-formal entities manufacturing personalized clothes for social functions such as marriage ceremonies.

The name Seshweshwe originated from the cloth being linked with the former King of Lesotho, King Moshoeshoe[2]. King Moshoeshoe received a Seshweshwe as a gift from a French missionary in the 1800s and he went on to make it popular among his people. In SeSotho they call it the sejeremane or seshoeshoe and in Afrikaans tarentaal. In IsiXhosa it is known as ujamani or isiShweshwe and se-Shweshwe in seTswana. The se-Shweshwe was further rooted in South Africa and Lesotho by German and Swiss settlers who made their own clothes from the imported fabric.

2.2 Deep Learning

To introduce the Seshweshwe fabric to the machine-learning-for-fashion community we explore the problem of generating Seshweshwe fashion images from drawings. In this section we introduce fundamental concepts in Deep Learning that form building blocks for the generative models used in this work.

2.2.1 Feedforward Neural Networks

A feedforward neural network, which may be termed as a multiplayer perceptron (MLP), is an arithmetic representation that resembles a directed acyclic graph with

[2]Moshoeshoe is Pronounced: mo-sh-wesh-where

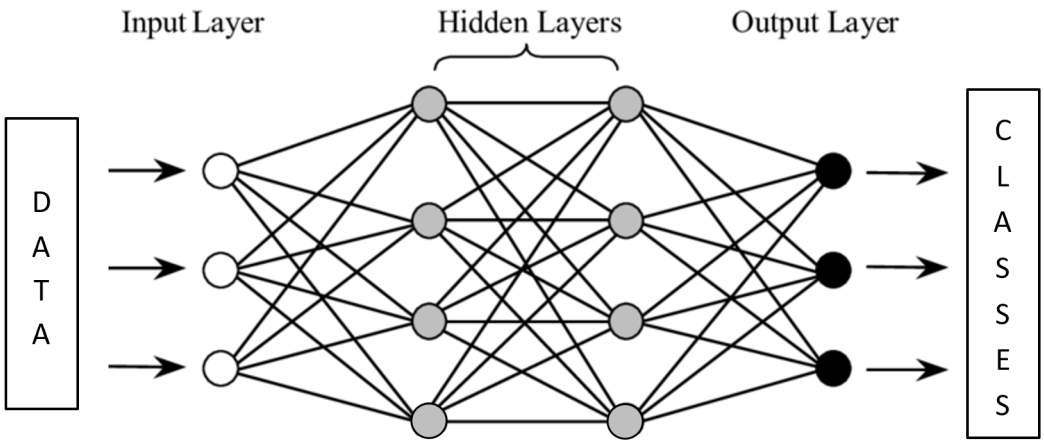


FIGURE 2.1: An illustrative diagram of a three layer feedforward neural network architecture with three inputs, two hidden layers with four neurons, and three output neurons.

a minimum of three layers of stacked nodes (termed "neurons") joined by edges (termed "weights") laid out in a network [69]. The networks are referred to as feedforward networks since information runs through the network from the first layer (referred to as the input layer), past the middle layer(s) and lastly to the final layer (termed the output layer). This process takes place without any feedback or repeated connections. Feedforward neural networks, bound by their weights W, are used to estimate any function $f(\boldsymbol{x} : \boldsymbol{W})$ with some inputs x. For instance, given a classifier with k number of classes $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{W})$ maps an input vector $\boldsymbol{x}$ to a probability distribution $\hat{\boldsymbol{y}}$ where $\hat{y}_k$ represents the network's estimated likelihood of the input being a member of class k. A feedforward neural network is made up of artificial neurons laid out in "fully connected" or "dense" layers, and each neuron in a single layer is joined to every other neuron in the next layer. The input of a neuron in each layer is an output of the preceding layer up to the last output layer. The layers that are in-between the input (first) and output (last) layers are termed hidden layers. A network with a minimum of two hidden layers is referred to as a deep neural network (DNN). Figure 2.1 is an example of a fully connected Deep neural network.

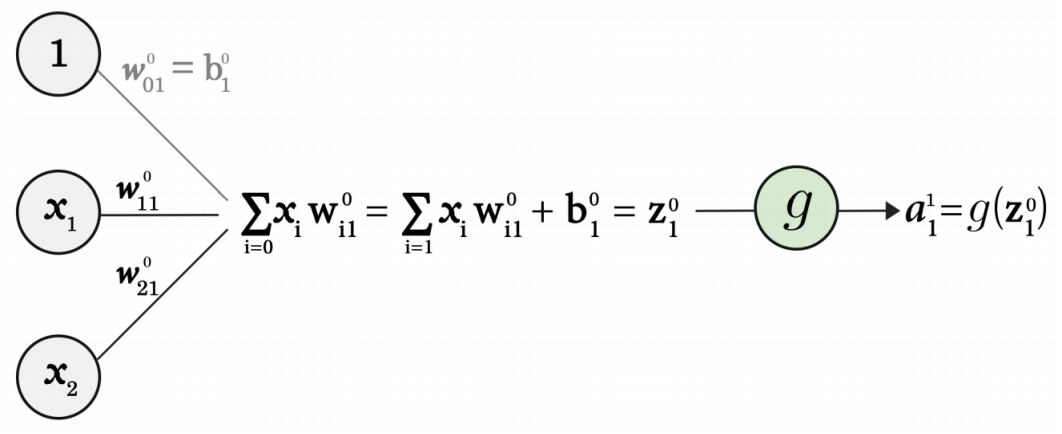


FIGURE 2.2: An artificial neuron with labeled inputs x_i, weights w_{ij} , bias b_j , pre-activation z_j , activation a_j and activation function g.

Artificial neurons

Artificial neurons are mathematical functions that model biological neurons [26]. They form basic units in an artificial neural network. Similar to a biological neuron, an artificial neuron computes and transfers information. A neuron computes information by processing the linear combination of its input, followed by applying a non-linear function referred to as an activation function. This activation function then computes the output of a neuron. A neuron's output serves as the input for any neuron joined to it in the next layer of the neural network.

The first artificial neuron model, developed in the 1950s by Frank Rosenblatt, was called the perceptron [69]. The perceptron takes a number of binary inputs $x_1, x_2, \cdots, x_n$ and produces a single binary output. It is parameterized by real-valued weights $w_1, w_2, \cdots, w_n$ that express the significance of the respective inputs to the output. To compute the output, a threshold function is applied to the weighted sum of the inputs and weights:

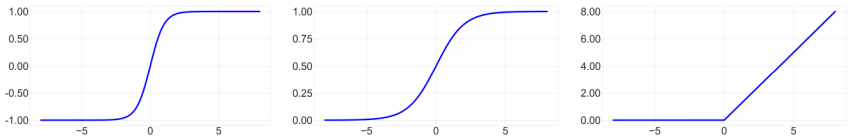
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold}, \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold}. \end{cases} \tag{2.1}$$

This can be re-written by re-arranging and replacing the threshold with a "bias"

term b, where $b = -\text{threshold}$, which is represented in the artificial neuron model as a constant input 1 and measures how difficult it is for the perceptron to return a positive value ("fire" in the language of biological neurons) [26]. Small changes in the inputs or weights of any perceptron in a network of perceptrons can lead to a completely different output as the threshold is passed. Sigmoid neurons are similar to perceptrons but relax the constraint that the inputs have to be binary and instead accept inputs with any value between 0 and 1. In the case of sigmoid neurons, the threshold activation function of the perceptron is also replaced with the sigmoid activation function (described in the next section). The advantage of sigmoid activation is that small changes in the input of the sigmoid neuron lead to small changes in its output. Modern neural networks accept any real-numbered input and use a range of different activation functions [26].

Activation Functions

Activation functions are always non-linear because a multi-layer neural network with linear activation functions could be re-parameterized as a single layer network due to the fact that the product of weight matrices $W_1, W_2, \cdots, W_N$ could be rewritten as W. Non-linear activation functions enable neural networks to learn complex mappings from input to outputs. They also typically have derivatives that are easy to compute to speed up the process of learning optimal network parameters during backpropagation.



(A) Tanh (B) Sigmoid (C) ReLU

FIGURE 2.3: Common Activation Functions.

Sigmoid

The sigmoid activation function (also called the logistic function) maps any real-valued input x into the range $[0, 1]$:

$$g(x) = \frac{1}{(1+e^{-x})}, \tag{2.2}$$

$$g\prime(x) = g(x)(1-g(x)) \tag{2.3}$$

The sigmoid activation functions was very popular in the early neural network literature because they tend to perform well when networks are very small but they are seldom used today [26]. They exhibit 3 issues. First, sigmoid outputs are centered at 0.5 so if all the components of an input vector are positive then all of the weight updates will have the same sign and will all increase or decrease together which slows down network convergence [7]. Second, sigmoid activations "saturate" in the sense that large and very large positive or negative pre-activation values produce activation values near 0 or 1. This is a problem because, as we will see in the section on backpropagation, this implies that the weights in these neurons do not update and neurons connected to saturated neurons only update slowly. Finally, in contrast to the ReLU activation function, the exponential function is computationally expensive to compute in comparison to a piecewise linear function [60].

Hyperbolic tangent

The hyperbolic tangent (tanh) activation function maps any real-valued input x into the range $[1, 1]$:

$$g(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{2.4}$$

$$g\prime(x) = 1 - g(x)^2 \tag{2.5}$$

The tanh function is sigmoidal and thus experiences the same saturating and computationally expensive issues as the sigmoid function but it is centered at zero and as such is always preferable to the sigmoid activation function [25].

Rectified Linear Unit

The ReLU unit clips any input value $x < 0$ at 0:

$$g(x) = max(0, x), \tag{2.6}$$

$$g\prime(x) = \begin{cases} 0 & \text{for } x < 0, \\ 1 & \text{for } x \geq 0 \end{cases} \tag{2.7}$$

The Rectified Linear Unit (ReLU) activation function is a simple piecewise linear activation function that does not saturate like the sigmoid and tanh and is much faster to compute. It is the most commonly used activation function in modern ANN architectures. It was introduced in early neural network models and dates at least as far as the Neocognitron [23] but it was largely ignored in favour of the sigmoid due to a belief that activation functions with non-differentiable points must be avoided [26]. It was popularized by two publications [59, 43] whose AlexNet CNN architecture was a breakthrough improvement over the state-of-the-art in image classification in 2012, almost halving the previous error rate in the ImageNet classification challenge. The use of ReLU activation functions was deemed by the AlexNet authors to be the most important factor in the success of their model.

Output Units

The neurons in the output layer of a neural network typically have a different activation function to the neurons in the previous layers. In the case of regression where the network seeks to predict a real-valued continuous numerical output, the output layer typically contains a single neuron with a linear activation function. In a 1-of-K class classification context, the desired output is a probability distribution over the K classes being predicted and the softmax function (which can be seen as a multi-class version of the sigmoid function) is used to transform a network's real-valued activation values into a probability distribution with values between 0 and 1 that together sum to 1. The softmax function $\mathbb{R}^K \mapsto \mathbb{R}^K$ applied to a vector $\boldsymbol{a} = [a_1, \cdots, a_K]^T$ is defined as:

$$g(a_i) = \frac{e^{a_i}}{\sum_{k=1}^{K} e^{a_k}} \tag{2.8}$$

2.2.2 Forward propagation

The process by which information propagates and flows forward through a feed-forward neural network from the inputs x to produce a predicted output $\hat{y}$ is called forward propagation.

Algorithm 1 The Forward propagation Algorithm

1: **Input:** Vector of inputs $\mathbf{x}$
2: **Input:** Neural network model function $f(\mathbf{x}; W)$ parameterized by $\mathbf{W}$
3: **Input:** Activation function $g(\cdot)$
4: **for** l in $1, 2, \ldots, L$ **do**
5: Compute neuron pre-activations $z_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)}$
6: Compute neuron activations a $a_j^{(l)} = g(z_j^{(l)})$
7: **return** $\hat{\mathbf{y}} = \mathbf{a}^{(\mathbf{L})}$

2.2.3 Cost Functions

A cost function, also known as an "objective", "loss" or "error" function, measures how well a predictive model's outputs correspond to known ground truth observed values. We will see in the next two sections how a neural network's weights can be updated using backpropagation and an optimization algorithm to make the network produce more accurate predictions and central to this process is the concept of a cost function. Cost functions can take many forms, for example the mean-squared error between observations and prediction values is a commonly used cost function for regression problems. A cost function $C(\hat{y}, y)$ maps pairs of model output predictions $\hat{y}_i$ and ground-truth known values y_i to a single scalar number that should be greater than or equal to zero and returns low values when a model's predictions are accurate and higher values otherwise. The cost is computed for each

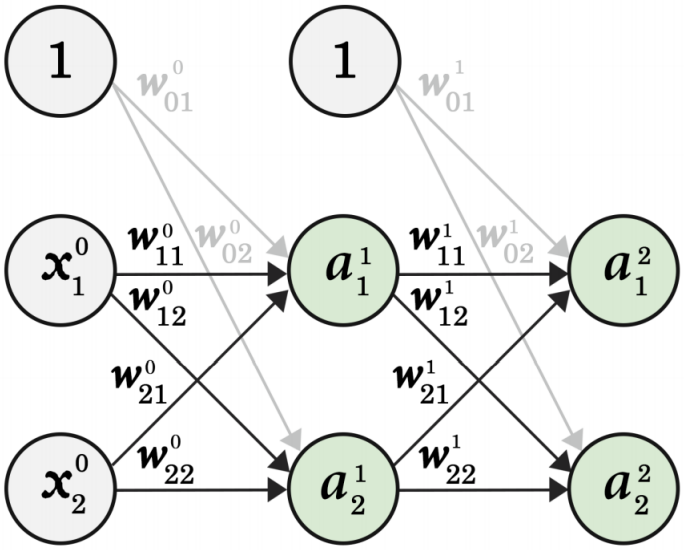


FIGURE 2.4: An example forward propagation algorithm calculation.

observation in a dataset x_i and then averaged to calculate an overall cost:

$$C(\hat{y}(x), y) = \frac{1}{n} \sum_i C_i(\hat{y}_i, y_i) \tag{2.9}$$

Cross-entropy Cost

The kind of cost function used is closely related to the type of output units. For classification problems with a softmax function in the final layer, the categorical cross-entropy cost function is typically used [60]. The categorical cross-entropy cost function for a K-class classification problem with N observations is defined as follows:

$$C = -\frac{1}{n} \sum_{n=1}^{N} \sum_{k=1}^{K} [y_k \log \hat{y}_k + (1 - y_k) \log(1 - \hat{y}_k)] \tag{2.10}$$

The categorical cross-entropy cost function derives from the principle of maximum likelihood estimation (MLE) in statistics which is a method of using data observations to estimate the parameters of a model. The log-likelihood function

$\mathcal{L}(\theta|x)$ for a discrete random variable is a function of the parameters of a model $\boldsymbol{\theta}$ given the data $\boldsymbol{x}$ defined as follows:

$$\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{x}) = \log P_\theta(\boldsymbol{x}) = \log P_\theta(X = \boldsymbol{x}) = \log\left(\prod_{n=1}^{N} p(\boldsymbol{x_i};\boldsymbol{\theta})\right) = \frac{1}{n}\sum_{n=1}^{N}\sum_{k=1}^{K}\log p(y_n = k|\boldsymbol{x_i},\theta) \tag{2.11}$$

2.2.4 Backpropagation

The process of training a neural network involves initializing its weights randomly, using forward propagation to compute the network's outputs given a training dataset, measuring the errors in these outputs using a cost function and the training dataset, and then updating the weights to try to reduce the error of the network [71]. Backpropagation, which is shorthand for "the backward propagation of errors", is the name for the method of adjusting the weights of the network using information from the cost function. After producing an output from the network and measuring how far or close this output is from the ground truth, that measure is used to proportionately adjust the weights so that the error gets smaller and the output gets closer and closer to resembling the ground truth of the data. But different weights have varying effects on the final output and hence the cost. In order to adjust the weights appropriately, we first have to measure the magnitude and direction (negative or positive) of each weight's influence on the cost function (usually referred to as gradients). This is done through differentiating the cost function with respect to the weights.
Given that neural networks are composite function (functions of functions), in calculus to differentiate a composite function we make use of the chain rule. Therefore, we can simply say that backprobagation is computing gradients of the cost function with respect to weights using chain rule and then using the same gradients to adjust the weights appropriately in order to produce outputs that are closer to ground truth [22, 65, 26].

In the general case with vectors $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$ and functions $g : \mathbb{R}^m \mapsto \mathbb{R}^n$ and $f : \mathbb{R}^n \mapsto \mathbb{R}$, if $y = g(x)$ and $z = f(y) = f(g(x))$ then the chain rule of differentiation is defined as follows:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x_i} \tag{2.12}$$

Defining the intermediate quantity δ_j^l to represent the error in the j neuron in the l layer, backpropagation uses the chain rule of differentiation to compute the *error* δ_j^l for each neuron which is then used to compute the gradient of the network's cost function C with respect to the network's weights w_{ij}^l given an arbitrary activation function g as follows:

$$\frac{\partial C}{\partial w_{ij}^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \times \frac{\partial a_j^{(l)}}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} \times x_i^{(l-1)} \quad \text{for all } i, j, l. \tag{2.13}$$

The error in the output layer is defined as:

And the error in the layers is defined recursively as:

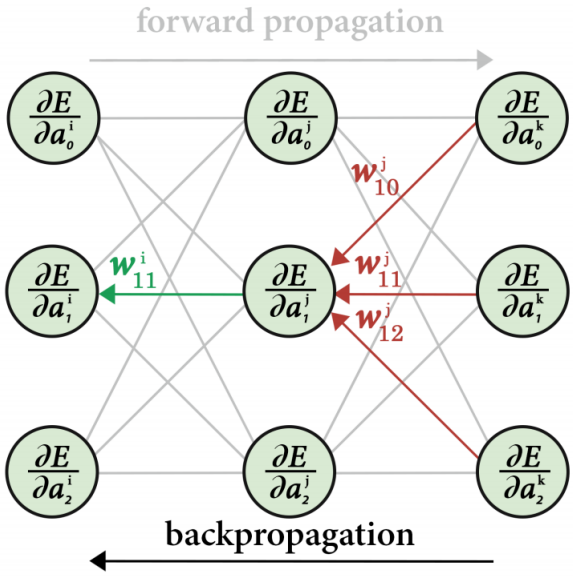


FIGURE 2.5: 5: An example backpropagation algorithm calculation.

Using the $\odot$ Hadamard product symbol to indicate element-wise multiplication and applying activation function g element-wise, the back-propagation algorithm

can be summarized as follows:

Algorithm 2 The Backpropagation algorithm

1: **Input:** Vector of inputs $\mathbf{x}$ with a corresponding ground-truth label y
2: **Input:** Neural network model function $f(\mathbf{x}; W)$ parameterized by $\mathbf{W}$
3: **Input:** Activation function $g(\cdot)$
4: **Input:** Cost function C
5: Compute current prediction using forward propagation $\mathbf{a}^L = \hat{\mathbf{y}}_\mathbf{i} = f(\mathbf{x}_i; \mathbf{W})$
6: Compute cost $C(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y}_\mathbf{i})$
7: Compute error for each neuron in output layer $\boldsymbol{\delta}^L = \nabla_{\mathbf{a}^L} C \odot g'\left(\mathbf{z}^L\right)$
8: Compute weight gradients in output layer $\nabla_{\mathbf{W}^L} C = \delta^{(k)} \mathbf{a}^{(k-1)}$
9: **for** l in $L-1, L-2, \ldots, 1$ **do**
10: Compute error for each neuron in l^{th} layer $\boldsymbol{\delta}^l = \mathbf{W}^{l+1}\boldsymbol{\delta}^{l+1} \odot g'\left(\mathbf{z}^l\right)$
11: Compute weight gradients for neurons in l^{th} layer $\nabla_{\mathbf{W}^l} C = \boldsymbol{\delta}^l \mathbf{a}^{l-1}$
12: **return** Weight gradients $\nabla_\mathbf{W} C$

2.2.5 Gradient Descent Optimization

Gradient descent is an iterative numerical optimization algorithm for finding the local minimum of a function by taking steps in the function's parameter space proportional to the negative of the function's gradient at the current point. Training a neural network involves initializing its weights randomly and then iteratively updating the weights using gradient descent on the network's cost function using gradients computed using backpropagation. As the neural network's weights are updated, its hidden units which are not part of the input or output come to encode important feature transformations of the input that enable the network to more accurately produce outputs [71]. In gradient descent, the gradient of the cost function computed over the entire dataset. In practice a variant called Stochastic Gradient Descent (SGD) which samples subsets of the dataset to compute gradients and tends to converge faster than gradient descent is more commonly used. The goal of the SGD algorithm is to update the network's parameters W to minimize the total cost $\sum_{i=1}^{n} C(\hat{y}, y_i)$ over the set of observation. It works by repeatedly sampling a random training example and computing the gradient of the cost of the example with respect to the parameters W (line 9). The network's parameters are then updated in

the opposite direction of the gradient by multiplying the gradient by the learning rate η that controls the size of the steps taken during gradient descent. A training epoch is defined as a set of SGD steps during which each training observation has been used exactly once to compute gradients.

Algorithm 3 Stochastic Gradient Descent for Training a Neural Network

Input: Training dataset of n input/output pairs $\{(\mathbf{x_1}, \mathbf{y_1}), \ldots, (\mathbf{x_n}, \mathbf{y_n})\}$
Input: Neural network model function $f(\mathbf{x}; W)$ parameterized by $\mathbf{W}$
Input: Cost function C
Input: Learning rate η
for $epoch$ in $1, 2, \ldots, epochs$ **do**
    Sample a single training dataset observation $(\mathbf{x_i}, \mathbf{y_i})$
    Compute current prediction using forward propagation $\hat{\mathbf{y}}_\mathbf{i} = f(\mathbf{x_i}; \mathbf{W})$
    Compute the cost $C(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y_i})$
    Compute gradients using backpropagation $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} \leftarrow$ gradients of $C(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y_i})$ w.r.t $\mathbf{W}$
    Update parameters $\mathbf{W^*} \leftarrow \mathbf{W} + \eta \frac{\partial \mathbf{C}}{\partial \mathbf{W}}$
return $\mathbf{W^*}$

The gradient of a neural network's parameters with respect to its cost function computed from a single observation can be noisy so a common alternative to SGD called minibatch SGD is often used. Minibatch SGD tends to produce smoother gradient updates at each step by computing the cost function and gradients based on a sample of m observations (m) is often referred to as the "batch size") rather than a single datapoint as in SGD.

The non-linearities in a neural network cause the cost function to be non-convex and therefore gradient descent with backpropagation is not guaranteed to find a global minimum of the network's cost function, only a local minimum which may not be the overall lowest cost possible for the model. This issue caused by the non-convexity of the cost function was long thought to be a major drawback of neural networks but in practice it is not [44]. One explanation due to Dauphin et al. (2014) for why local minima are not an issue is that in high dimensional space, stationary points in the error surface are not local minima but saddle points since in high dimensional space there is almost certainly at least one dimension in which the cost surface can be reduced. It can be difficult for SGD as a first-order gradient descent

Algorithm 4 Minibatch Stochastic Gradient Descent for Training a Neural Network

Input: Training dataset of n input/output pairs $\{(\mathbf{x_1}, \mathbf{y_1}), \ldots, (\mathbf{x_n}, \mathbf{y_n})\}$
Input: Neural network model function $f(\mathbf{x}; W)$ parameterized by $\mathbf{W}$
Input: Cost function C
Input: Learning rate η
for *epoch* in $1, 2, \ldots, epochs$ **do**
    Sample a minibatch of m training dataset observations $\{(\mathbf{x_1}, \mathbf{y_1}), \ldots, (\mathbf{x_m}, \mathbf{y_m})\}$
    Initialize gradient for this minibatch $\hat{\mathbf{g}} \leftarrow 0$
    for $i = 1$ to m **do**
        Compute current prediction using forward propagation $\hat{\mathbf{y}}_\mathbf{i} = f(\mathbf{x_i}; \mathbf{W})$
        Compute the cost $C(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y_i})$
        Update minibatch gradients using backpropagation $\hat{\mathbf{g}} \leftarrow \hat{\mathbf{g}} + \frac{1}{m} \frac{\partial \mathbf{C}}{\partial \mathbf{W}}$
    Update parameters $\mathbf{W}^* \leftarrow \mathbf{W} + \eta \frac{\partial \mathbf{C}}{\partial \mathbf{W}}$
return $\mathbf{W}^*$

algorithm to escape these seeming local minima and as such several second-order approaches have been proposed.

Drawbacks of Fully-connected DNNs: Image classification

Fully-connected DNNs are not well suited to classifying images directly from image pixel data for 3 reasons:

- A single fully-connected layer with just 100 neurons connected to each colour channel of each pixel in a 224×224 colour input image would have 1 500 520 800 trainable weight parameters which can cause computational challenges and likely cause the model to overfit its training data.
- Fully connected networks ignore the spatial topology of image data with pixels tending to be correlated with nearby pixels to form salient features within an image.
- Fully-connected networks have no invariance with respect to local distortions or translations causing them to fail on classification problems outside of controlled environments where objects out-of-sample often appear distorted or in different parts of an image than they did in training data.

In theory, due to the universal approximation theorem, a fully-connected network of sufficient size could learn to produce outputs that are translation invariant but this would require many units with identical weight patterns and an impractically large number of training examples to cover the space of possible variations [45]. In practice, fully-connected DNNs applied to image classification tasks tend to overfit the training data due to their large number of parameters and do not generalize well out-of sample [46].

2.2.6 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a special class of networks that are used mainly for data that has a grid-like structure. Images and time-series data are an example of such data. There has been a great success in visual applications of Convolutional networks ever since they first surfaced in 1989. CNNs have become the basic building block of most modern computer vision systems.
Convolutional networks use a type of linear operation called convolution in mathematics.

Convolution takes advantage of three important architectural ideas:

Local Receptive Fields

Each convolutional unit receives input from a set of units located in a small neighbourhood of the previous layer, called the neuron's receptive field, which enables it to extract elementary local spatial features such as edges and corners.

Shared Weights

Each convolutional unit is scanned across the extent of the previous layer to compute an output plane called a feature map (see Fig. 2.7). Since features that are useful in one part of an image are likely to be useful across the entire image, each convolutional unit is set to have identical weight vectors at each position that is scanned. This weight sharing introduces translation invariance and significantly reduces the number of trainable parameters in the network. A convolutional layer is composed of several convolutional units each computing a feature map so that different types of features are extracted at each location.

Subsampling

After a feature is extracted, its exact location becomes less important. Subsampling layers lower the resolution of preceding feature maps while ensuring discriminative features persist through the network. This lowers the network's sensitivity to distortions in its inputs. Successive alternating convolutional and subsampling layers are stacked to reduce the spatial resolution of the network and extract higher level hierarchical features.

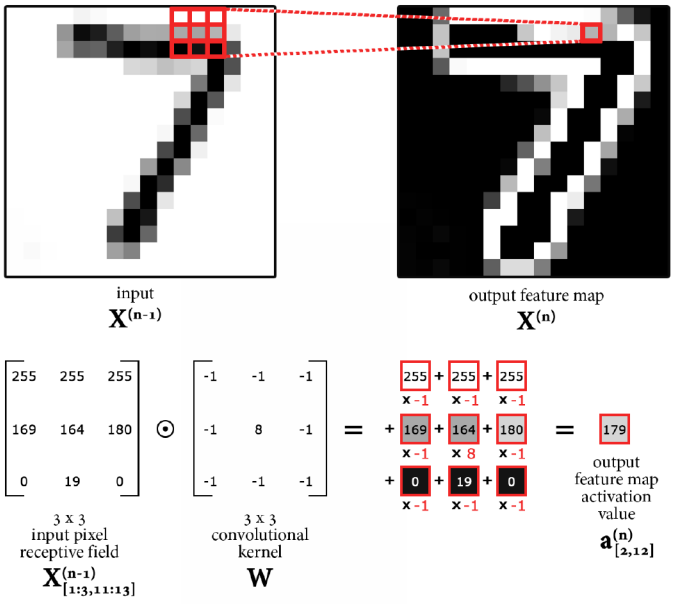


FIGURE 2.6: An example convolutional feature map computed by scanning a 3×3 convolutional unit (highlighted in red) across a 16×16 pixel grayscale input image (left) to produce output feature map (right). The calculation of the zoomed pixel highlighted in red on the output is shown beneath the images. The bias weight is set to zero and ReLU activation function is omitted for clarity. Black padding was used at the border to produce a valid convolution. [63].

Convolutional Layers

A convolutional layer contains sets of filters, also called convolutional filters. Each convolutional unit is scanned across its input using the same set of weights in each position similar to a mathematical convolution operation, hence the name "convolutional layer". The first convolutional layer in a CNN scans across the input

image but convolutional layers are usually stacked with later convolutional layers scanning across the outputs of earlier convolutional and subsampling layers (for example, see Fig. 2.7).
A mathematical convolution essentially computes an arbitrary-dimensional weighted average with a weighting function being applied across the domain of an input function to produce an output. Convolutions are commonly used in image processing to produce filter effects, for example the 3×3 convolutional filter illustrated in Fig. 2.7 - called an "outline filter" in image processing software [63] - highlights edges in the input. In the image processing context, kernel weights are typically hand-chosen to produce a desired effect but in CNNs these kernel weights are randomly initialized and then learned using backpropagation so that each convolutional layer's output feature maps contain discriminative feature transformations that are useful for classification.

Forward Propagation Through a Convolutional Layer

Forward propagation through a convolutional layer proceeds as illustrated above by scanning the convolutional kernel across each point in the input to compute a feature map output.

Backpropagation Through a Convolutional Layer

Backpropagation through a convolutional layer is similar to backpropagation in fullyconnected layers but takes into account the local weight sharing property of convolution units so that gradients propagate through all weights in the kernel as it is applied to each position of its input.

Subsampling Layers

Subsampling in a CNN refers to the loss of some position information between layers as the input is transformed through the network, reducing the number of parameters in the network while enabling it to learn higher level hierarchical features. Subsampling layers follow convolutional layers in a CNN to increase the effective receptive field of its feature maps and reduce the resolution of successive convolutional layers, making the network more robust to shifts and distortions in

the positions of objects in the input image [45]. This helps the network generalize better since objects may appear at different positions out-of-sample than were present in the training data images.
A subsampling layer (also called a "pooling layer") is similar to a convolutional layer in that it consists of several subsampling units that each slide a window function over its input. Unlike a convolutional layer, the outputs of a subsampling layer have a lower dimensionality and the units in a subsampling layer typically do not have any trainable parameters therefore all the subsampling units in a subsampling layer are typically identical. Subsampling units typically use the same stride as window size so that they do not overlap.
Two common types of functions used in pooling layers, max pooling and average pooling, use max and average window functions respectively. Average pooling was the first subsampling operation used in CNNs [45] but max pooling is more commonly used in modern CNNs [43] and typically outperforms average pooling because averaging dilutes signal from previous layers [72].

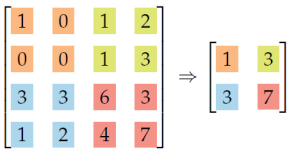


FIGURE 2.7: This example illustrates a max pooling operation with a 2×2 filter size and a window stride of 2 applied to a feature map to reduce its resolution by a factor of 4.

Forward Propagation Through a Pooling Layer

Forward propagation through a pooling layer proceeds as illustrated in the example above by applying the subsampling operation to each point in the input feature map to compute an output feature map.

Backpropagation Through a Pooling Layer

In backpropagation, the backward pass for a max pooling unit only passes the gradient to the input from the previous layer that had the maximum value in the filter window from the forward pass [26]. Backpropagation through an average pooling

unit averages the gradient in the subsequent layer into the corresponding inputs in the pooled layer.

2.2.7 Generative Models

Generative models are a subset of unsupervised learning models where given training data, the goal is to generate new data from the same distribution. Fundamentally, generative models try to address the problem of density estimation like other unsupervised learning methods like clustering, feature learning, dimensionality reduction. In addition to learning the underlying hidden structure of the data, generative models go a step further by generating novel examples that resemble the training data in someway.

Unlike supervised discriminative models where given any input data (X, Y) where X is a feature set and Y a set of labels, the model learns a probability distribution $p(y|x)$, generative models are a class of machine learning models where given any input x the model learns $p(x)$. This makes generative models and unsupervised learning in general very special in that we can now move beyond mapping features to labels to fundamentally understanding our world, its evolution and its objects. This makes it possible to us to re-imagine our would, stimulate progress and give birth to evolution in many areas of application.

Some examples of applications of generative models that are already shaping our would include generating realistic samples for artwork, improving cybersecurity [73], Video Prediction [82], text-to-image translation [70] and health care [6, 21, 74].

Generative models can be organised into explicit and implicit density estimation models.

The goal of Explicit Density Estimation is to explicitly compute $p(x) = f(x, W)$. Given a dataset $x^{(1)}, x^{(2)}, \cdots, x^{(N)}$ a model is trained by solving:

$$W^* = \arg\max_W \sum_i \log f(x^{(i)}, W), \tag{2.14}$$

which will be the loss function trained with gradient descent. Implicit density estimation models are models that do not explicitly compute $p(x)$, but can sample

from $p(x)$.

Examples of generative models that seek to compute an explicit density function include autoregressive models and variational autoencoders. Autoregressive models compute a tractable density while variational autoencoders can compute an approximation of $p(x)$.

Explicit tractable density estimation: Autoregressive

Autoregressive models are derived from the probability chain rule. Assuming $\mathbf{x}$ consists of multiple sub-parts: $\mathbf{x} = (x_1, x_2, \cdots, x_T)$, the probability of $\mathbf{x}$, $p(x)$ is defined by:

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t | x_1, x_2, \cdots, x_{t-1}). \tag{2.15}$$

This means if we minimize

$$-\log p(\mathbf{x}) = -\sum_{t=1}^{T} \log p(x_t | x_1, x_2, \cdots, x_{t-1}). \tag{2.16}$$

by training a recurrent neural network we can directly maximize the likelihood of the input ($\mathbf{x}$ and model $p(x_t | x_{1:i1})$.
Autoregressive models are very impressive density estimators but, the input data is required to be broken down into an ordered list of sub-components. This works well for certain data types like audio and text and not so well for images and other data types where the order of the sub-components of an input is not clear. This can have an impact on how well the model performs based on the network architecture chosen. In addition, given that sampling with autoregressive models is a step-by-step process where the probability of a component is dependent on the probabilities of all the previous components, this can be extremely slow on high dimensional data.

Explicit approximate density estimation: Variational autoencoders

Variational Autoencoders (VAE) define an intractable density that we cannot explicitly compute or optimize but we will be able to directly optimize a lower bound on the density. VAEs are derived from another type of generative model called Autoencoders. An autoencoder is an unsupervised method for learning feature vectors from input data x, without any labels. An autoencoder consists of two neural networks $f(\mathbf{x})$ and $g(\mathbf{z})$, usually called an encoder and a decoder. An encoder inputs raw data $\mathbf{x}$ and outputs a feature vector $\mathbf{z}$, which the decoder takes in as input and outputs a reconstruction of the input data $\hat{\mathbf{x}}$ as shown in the diagram in Fig. 2.8. The Feature vector $g(\mathbf{z})$ needs to be lower dimensional than the data. To extract these feature representations $\mathbf{z}$ of the data without any help from labels, the functions f and g are trained to minimize the difference between the input data $\mathbf{x}$ and the reconstructed data $\hat{\mathbf{x}}$ by minimizing the loss given by: $||\mathbf{x} - \hat{\mathbf{x}}||_2^2$.

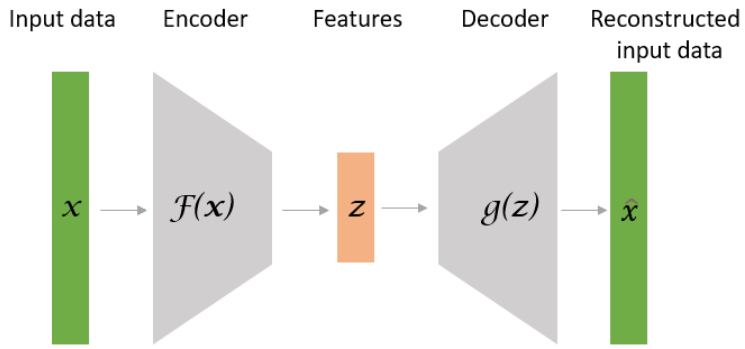


FIGURE 2.8: Autoencoders illustrative diagram

As generative models, autoencoders can only reconstruct input data. For other uses, the encoder part of the autoencoder is sometimes used as an initial feature extractor to initialize a supervised model. This is as far as they go, since they are not probabilistic, there is no way to sample new data from the learned model. To solve for this, Variational Autoencoders are a probabilistic modification of autoencoders: they can learn latent features $\mathbf{z}$ from raw data and sample from the model to generate new data.

Variational Autoencoders seek to maximise the likelihood $p(\mathbf{x})$ of the training data $\mathbf{x}$ given by:

$$p(\mathbf{x}) = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z})\, d\mathbf{z}, \tag{2.17}$$

but this is intractable for every $\mathbf{z}$, since we only have specific $\mathbf{z}$ vectors that only relate to the input data. A solution for this is to learn the distribution of $\mathbf{z}$ given input data $\mathbf{x}$: $p(\mathbf{z}|\mathbf{x})$ given by:

$$p(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) / p_\theta(\mathbf{x}), \tag{2.18}$$

but, even this density is intractable. To solve for this problem, instead of trying to maximize $p(\mathbf{z} \mid \mathbf{x})$ directly, it is approximated by defining a lower bound $q_\phi(\mathbf{z}|\mathbf{x})$.

As a result, a Variational Autoencoder model is made up of two probabilistic networks: an encoder $q_\phi(\mathbf{z}|\mathbf{x})$ which takes input data and outputs a mean $\mu_{\mathbf{z}|\mathbf{x}}$ and (diagonal) covariance $\sum_{\mathbf{z}|\mathbf{x}}$ and a decoder $p_\theta(\mathbf{x}|\mathbf{z})$ which takes the vector $\mathbf{z}$ as input and outputs a mean $\mu_{\mathbf{x}|\mathbf{z}}$ and (diagonal) covariance $\sum_{\mathbf{x}|\mathbf{z}}$. See illustrative diagram in Fig. 2.9

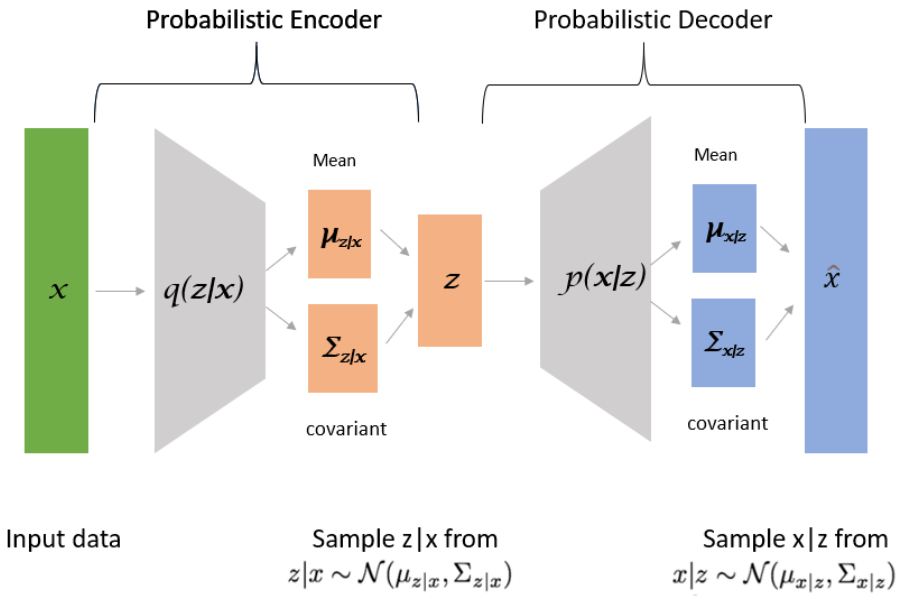


FIGURE 2.9: Variational Autoencoders illustrative diagram

VAEs are trained by maximizing the a loss function given by:

$$\mathcal{L}(x^{(i)}, \theta, \phi) = \mathbb{E}_z[log p_\theta(x^{(i)}|z)] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))], \tag{2.19}$$

The first term, forces the generated samples to be as close as possible to the input data while the second term forces the learned distribution of $z|x$ to be as close as possible to the prior $p_\theta(z)$, which is generally assumed to be a Gaussian distribution.

VAEs provide a principled approach to generative networks and enable inference of $q(z|x)$, which can be useful in feature representation. However given that it maximizes a lower bound of the likelihood function, evaluation is not as good as autoregressive models. Another drawback of VAEs, is that samples are generally more foggy and of poor quality when compared to the latest Generative Adversarial Networks. Ongoing research include more versatile approximations. For example, instead of a diagonal Gaussian have a richer approximation of the posterior and have better structured latent variables.
It turns out we need not to explicitly model density, in order to sample from it. The next section will expand on this.

Implicit density estimation models: Generative Adversarial Networks (GANs)

GANs don't work with any explicit density function. Instead, they take a game-theoretic approach of learning to generate from the training distribution through a two-player game [27]. Suppose we wish to sample from a complex, high-dimensional training distribution. There is currently no direct way to do this. GANs help us solve this problem by first sampling from a simple distribution, e.g. random noise and then learn a transformation to the training distribution which is usually very complex. This complex transformation is represented by a neural network. A basic GAN is made up of two neural networks famously referred to as the Generator network(G) and the Discriminator network(D). During training the Generator network tries to fool the discriminator by generating real-looking images that are similar to images from the training set and the Discriminator network tries to differentiate between real and generated images. See illustrative diagram in Fig. 2.10.

G and D are trained together in an alternating procedure to minimize and maximize the loss function given by Equation (2.20).

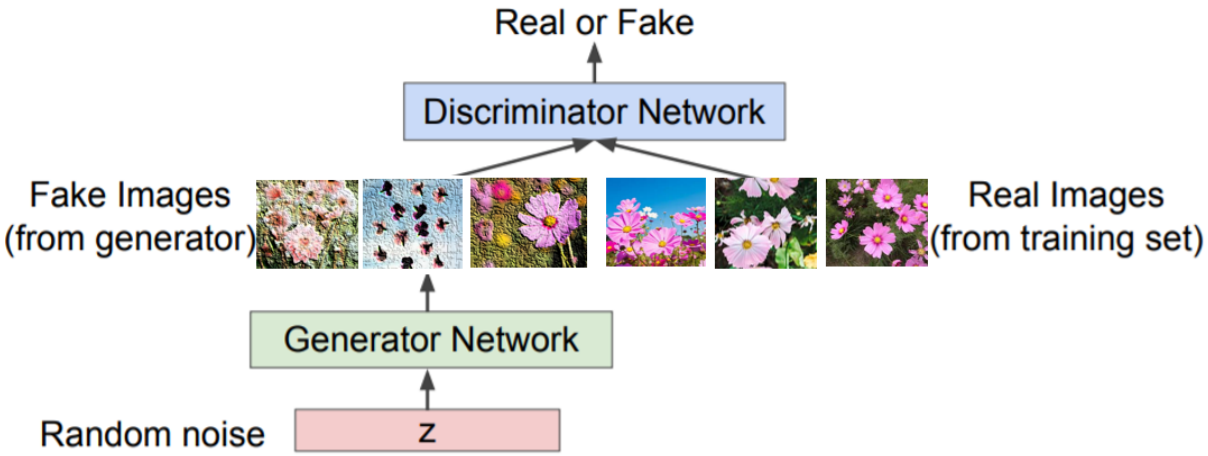


FIGURE 2.10: GAN training process illustration.

$$\min_{\theta_g} \max_{\theta_d} [\mathbb{E}_{x \sim p_{data}} log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} log(1 - D_{\theta_d}(G_{\theta_g}(z)))] \tag{2.20}$$

The discriminator outputs a likelihood of its input being real (i.e a value between 0 and 1). The output of the Discriminator for real data x is denoted by $D_{\theta_d}(x)$ while output for generated fake data $G(z)$ is denoted by $D_{\theta_d}(G_{\theta_g}(z))$.

The Discriminator D_{θ_d} seeks to maximize the objective function so that $D_{\theta_d}(x)$ approximates to 1 (real) and $D_{\theta_d}(G_{\theta_g}(z))$ approximates to 0 (not real/synthetic). In contrast, the Generator network G_{θ_g} seeks to minimize the loss so that $D_{\theta_d}(G_{\theta_g}(z))$ approximates to 1 (thus fooling the discriminator into identifying the synthesized data $G(z)$ as real).

In order to train, we alternate between gradient ascent on the discriminator:

$$\max_{\theta_d} [\mathbb{E}_{x \sim p_{data}} log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} log(1 - D_{\theta_d}(G_{\theta_g}(z)))] \tag{2.21}$$

and gradient descent on the generator:

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} log(1 - D_{\theta_d}(G_{\theta_g}(z))). \tag{2.22}$$

However, in practice, optimizing this generator objective does not work well.

The gradient of the function $log(1 - D_{\theta_d}(G_{\theta_g}(z)))$ is very low when the generator still needs more training (when $D_{\theta_d}(G_{\theta_g}(z))$ is near 0) and has a very high gradient when the generated samples are good i.e. when $D_{\theta_d}(G_{\theta_g}(z))$ is near 1. This slows down training when we need it the most and speeds up training when we should be slowing down. Therefore, as standard in practice, instead of minimizing the likelihood of discriminator being correct, we maximize likelihood of the discriminator being wrong. This is still the same objective of tricking the discriminator, but with a higher gradient signal for bad samples which is more efficient. Hence, in practice, to train the two networks we alternate between gradient ascent on the discriminator:

$$\max_{\theta_d}[\mathbb{E}_{x\sim p_{data}} log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} log(1 - D_{\theta_d}(G_{\theta_g}(z)))], \tag{2.23}$$

and gradient ascent on the generator:

$$\max_{\theta_g} \mathbb{E}_{z\sim p(z)} log(D_{\theta_d}(G_{\theta_g}(z))). \tag{2.24}$$

Finally after training, the generator network is used to generate new samples.

GANs have been further used in super resolution, human pose approximation, age estimation and 3D object reconstruction. Deep neural networks have shown great success in discriminative classification tasks in domains with big annotated data sets. An example of this is the ImageNet dataset which is made-up of over 1M images [43]. For many tasks, the quantity of annotated data available is not enough to train the millions of parameters that are often in these deep neural networks. It has however been discovered that information carried in a model trained on a big dataset can be used for other computer vision tasks. This can be done by making use of these models as ready-made feature extractors or by fitting them to other domains through fine-tuning. The pre-trained model is made to initialise the weights for new tasks. These weights can then be fine-tuned with the training images from the target domain.

In the past few years we have seen that training a new model using a pre-trained model or pre-trained weights requires much lesser images. GANs are generally

trained from the beginning. Transfer learning is commonly applied to classification models and not used often for GANs. Like classification models, the number of parameters in GANs is enormous. Take the popular DC-GAN model [64] for instance, it takes 36 million parameters to produce an image of 64 X 64 pixels. In areas that lack sufficient training images, making use of pretrained GANs could make the quality of generated images better.

GANs were brought about by Goodfellow et.al. in the year 2014 [27], the original model that was introduced is made up of fully connected layers in a sequence which limits it to simple datasets. In the production of real images of higher complexity, convolutional architectures have been seen to be more fitted. Soon after the first paper on GANs, Deep Convolution GANs (DC-GANs) became the base GAN built for image production problems [64]. Within the DC-GAN, the generator up-samples input features in series through fractionally strided convolutions. The discriminator makes use of normal convolutions to categorise input images. Recently, some work has been done on multi-scale architectures showing how they can effectively produce high resolution images [17, 89, 40]. We have also seen that ensembles can be used to better the quality of generated distributions [83].

There are some challenges associated with GANs in-line with their training. These challenges include convergence properties, model collapse or stability issues. The original GAN loss [27] is unable to work with unsuitable distributions, for example those with disjoint supports, regularly seen when training a GAN [52]. These challenges have been addressed by making use of the Wassertain GAN [53]. The Wassertain GAN uses the Wassertain distance as a loss yet needing the generator to be 1-Lipschitz. A more stable solution is adding a gradient penalty term to the Wassertain GAN loss [29].

There are also conditional GANs (cGANs) [54] which use certain features as a prior to train conditional generative models. These conditions could be text [67, 89, 70], class labels [61, 62, 28], another image, for example in the case of style transfer [19] or image translation [41, 91].
cGANs often apply their condition in both the generator and the discriminator by

combining it to the input of layers. It is either combined with the random vector on the first layer or combined with learned attributes in the later layers. There are other conditional GANs that do not have this common design. For example, there are GANs in which the conditioning is incorporated in the batch normalization layer [19]. The AC-GAN [61] has an interesting design in that it allows for the reconstruction of the class-conditional information by adding an auxiliary decoder to the discriminator. In the same fashion, InfoGAN [11] reconstructs a part of the latent vectors from which the samples are produced. Another variation of a GAN discriminator has been brought forward where the projection layer uses the dot product of the conditional information and the output in between layers to calculate loss [55].

In the following chapter we see how generative models have been used for fashion use cases.

Chapter 3

Generative Models and Datasets for Fashion

Here we provide an overview of the current literature in Generative Models for Fashion and discuss some existing fashion datasets. We motivate our research by highlighting existing gaps in the literature and introduce the Seshweshwe fashion dataset as our contribution towards cultural representation in generative works. In this chapter we also discuss ethical considerations in relations to algorithmic cultural appropriation and make a recommendation for preventative measures and responsible use of cultural fashion datasets.

3.1 Generative Models for Fashion Synthesis

Generative models have been used as a tool to solve important problems in the online shopping fashion industry. For example, a virtual try-on network (VITON) [32, 18] can transfer an image of a clothing item to an image of a person wearing the item; pose transfer generative model [93] which can have rich applications in the e-commerce industry.

Generative models have also been used in interesting fashion communication applications like the Fashion++ [34] which makes visual changes to a given outfit to improve its overall fashionability. However, this model's notion of fashionability does not have diverse cultural contexts. A cultural clothing item can be changed and presented in an inappropriate manner that strips off its cultural significance.

Generative models have also been used as a tool in fashion and style generation. These advancements in generative networks have tremendous potential to provide

fashion designers with instruments to easily re-imagine and scale their design generation. For example, GANs have been used to synthesize high-resolution images without input text specifications [70]. They have also been used to produce synthetic images that look real relying only on text specifications using a large Fashion dataset of 260,480 high-quality images as training data [70, 92]. However, not all fashion domains have large datasets of high-resolution for training, it would still be very challenging to generate high-quality images in such cases.
Style transfer algorithms have been used for generating fast fashion [66] and personalised fashion [24, 38]. Personalised clothing has been generated based on a user's fashion taste by discovering the user's style from their existing clothing. This approach limits fashion creation to the user's past decisions and does not allow for evolution in style and keeping up with current fashion trends.
Although recent breakthroughs in generative modeling can be used to produce images of unparalleled realism, in our knowledge there is currently no published work on African fashion generation.

3.2 Fashion Datasets

In this section we give an overview of some existing fashion datasets, identify some gaps and also introduce the Seshweshwe dataset as an exemplary contribution towards filling existing gaps in machine learning fashion datasets.

3.2.1 Existing Fashion Datasets

There are multiple fashion datasets that are created either in generative fashion or in fashion image retrieval. Fashion-Gen [70] is a dataset on text to image translation, helpful for generating fashion designs from descriptions. Fashion-IQ [30] is a dataset for natural language based image retrieval systems. DeepFashion [47] is a large scale dataset with fine-grained annotation helpful for a variety of tasks related to fashion. Geostyle [51] is a fashion dataset that was used to try to model fashion migration and influence in some parts of the world [4]. In all these fashion datasets,

the origins of the fashion items are not given, and they mainly consist of the western clothing styles.

In the following section we introduce the first dataset that considers African fashion, in particular, we present the curation process of a fashion dataset representing different designs of Southern African modern Seshweshwe fashion dresses.

3.2.2 Seshweshwe Dataset

The dataset is made up of images of modern African fashion items and corresponding sketches split into train and test subsets. The fashion items, most of which are dresses, are made of the popular Seshweshwe fabric.

FIGURE 3.1: Examples of paired images in the Seshweshwe dataset.

Data Collection

We collected photographs of Seshweshwe dresses from an image search engine called Google Images[1] where the resulting photos are from social media websites, forums, weblogs, and other materials created by users. The query keywords We

[1]https://www.google.com/imghp

used are 'se-Shweshwe dresses', 'Batswana traditional dresses' and 'Basotho traditional dress'. We fed these query keywords to Google Images, and downloaded the returned photographs. A total of 640 photographs with noisy background were obtained from Google Images.

Data Cleaning

To clean the dataset, we identified and removed duplicates, removed unusable images of low resolution or irrelevant objects. A total of 500 se-Shweshwe clothing images of varying designs were retained to make up the se-Shweshwe dataset.

Models trained on images with noisy background generated poor quality images. While reliance on background in classification tasks is still an open problem [84], we have seen that a standardized background on images can significantly contribute to the quality of output images, particularly for translation tasks [70]. This motivated the removal of the noisy background in our images.
To remove background from images we made use of online background removing tools[2,3] with some human interaction for visual analysis and making manual corrections.

Creating Sketches

A sketch is a series of lines that roughly imitate the borders and inner textures of an object. Sketches are traditionally drawn by humans using a pencil, pen, crayons, digital tools etc. Human made sketches take a lot of time to make and are generally costly.

To train our model to translate a sketch of a dress to an image of a Shweshwe dress we needed a dataset of paired images and sketches. This means we needed to ask a human sketch artist to draw a corresponding sketch for each of the 500 images we had collected. This was not practical given our limited and the high cost of commissioning an artist.

[2]https://www.remove.bg/
[3]https://removal.ai/

To create a dataset of paired sketches and dress images, we used an off-the-shelf method that converts images to pencil sketches[4]. This is done by first converting the original BGR image to gray scale, inverting the gray scale image, followed by blurring it, then inverting the blurred image, finally, dividing the gray scale image by the inverted blurred image to get the pencil sketch image.
At the end of this process we retained 500 sketches and 500 corresponding images. These two sets were divided into subsets of 400 elements for training and 100 elements for testing in a paired manner.

In the following section we discuss existing problematic practices in the non-AI fashion industry. We argue that without proper use, the use of generative models on cultural fashion data will exacerbate these problematic practices. Therefore, we also propose practical guidelines towards responsible and proper use of cultural fashion datasets.

3.3 Ethical Considerations in Generative Fashion

Generative models could have a positive impact on society in general by helping with cultural and identity representations. Aljowaysir, in Salaf[5], uses StyleGAN to reveal the political and social aspects of her past generations that are missed from the collective memory. Jake Elwes, by "Zizi and Me"[6], relying on DeepFake, addresses challenges and discriminations against Queers and drag artists.
Seshweshwe inspired fashion generation also represents a significant part of Southern African cultures, to broader communities. However, using generative models for representing cultures in the context of fashion, adds a layer of complexity to a well-known problem in the non-AI fashion industry, known as cultural appropriation.

[4] https://www.youtube.com/watch?v=vS2ubdiAXvg&t=11s
[5] http://www.aiartonline.com/highlights-2020/nouf-aljowaysir/
[6] https://www.jakeelwes.com/project-zizi-and-me.html

Defining cultural appropriation: Culture is defined as something that a group of people share that is characterised by shared ideologies, practices and way of life. Culture is often rooted on ethnicity, religion, geography, or social environment. Appropriation is taking for one's own use something that which belongs to others without permission or acknowledgement. Appropriation can be seen as social plagiarism.
Putting the two terms together, cultural appropriation is adopting elements of a culture without acknowledging their origins. Cultural appropriation involves a lack of understanding of or appreciation for the historical context of the said cultural elements. As such, cultural appropriation often results in cultural elements being adapted out of context, or re-purposed by members of another culture or identity. [86, 80]. The harm can be amplified, when members of a dominant culture appropriate from disadvantaged minority cultures or historically oppressed or marginalized groups [87].

3.3.1 Cultural appropriation in non-AI fashion industry.

The fashion industry (non-AI) has a long history of appropriating cultural elements. There has been a lot of reported cases of different fashion houses repeatedly taking elements of cultures and decontextualizing them, stripping them of their cultural significance and misrepresenting them as their own new trendy fashion for their own financial gain and aesthetic indulgence.
Traditional designs are an art that transcend aesthetics, they often carry a lot of historic information, symbolism and significance. Take for example the case of the basotho blankets:

A variety of blankets are used by the Sotho and South African people in important initiation events. An example is the Moholobela blanket that is used to symbolise fertility when boys transition into men. After this initiation from boys to men, the now men will put on a different blanket called Lekhokolo as an assertion of them being men. For the women, a motlotlehi blanket is worn on the day of their marriage ceremony. Further upon the birth of a woman's first child, she is given a Serope blanket. There is also blankets that are used by the King and his chiefs and it is called Seana Marena which means Chief's blanket. This blanket has the highest

rank of all the Basotho blankets.[7].
A French fashion house and luxury goods company, Louis Vuitton in their recent 2017 menswear collection turned the basotho blanket into a fashion trend for men[8]. Normally these basotho blankets retail for R500 ($35) on average, while Louis Vuitton sells them for R33, 000 ($2317).

This case of the basotho blanket is not exceptional and was certainly not the first. Amongst the earlier cases, a traditional Romanian blouse was misappropriated by French fashion designer, Yves Saint Laurent in his 1981 haute couture Fall-Winter collection [36]. The 1981 collection sparked a widespread interest and unleashed a trend of further appropriations of the blouse. Today this Romanian blouse has a very common presence in the fashion industry with anonymous origins. "I argue that Saint Laurent's appropriation of the Romanian blouse inadvertently set off a series of misappropriations. As the original cultural context became fused and confused with nations of loosely defined Eastern Europe or Russian "peasant blouses," the Romanian blouse became hybridized, and its cultural context once again became historically erased"[36].

Amongst many other recent cases, Louis Vuitton misappropriated the Shuka cloth traditionally worn by the Masai community in East Africa. In the year 2018 South African cultural sustainability fashion designer Laduma Ngxokolo battled cultural appropriation in a legal dispute with a Spanish fast-fashion retailer Zara where Zara introduced socks that featured a design similar to that of Ngxokolo's signature designs, representing the cultural bead-works by the Xhosa community in South Africa[9].

What makes cultural appropriation problematic in most cases is that cultural elements are often presented in a manner that strips off their cultural significance and meaning, and this can be specifically amplified using automated algorithms such as generative models.
"Textile craftsmanship is part of cultural heritage and has been an important element in building cultural identities. This is reflected in the traditional garments of

[7] The history of the basotho traditional-blanket
[8] Louis Vuitton's appropriation of the basotho traditional blanket
[9] How Laduma Ngxokolo battled cultural appropriation

different communities and indigenous people worldwide."[10]
With this understanding that cultural textile craftsmanship is very closely tied with cultural identity, misrepresentation of cultural elements harms the sustainability of that culture and thus robbing future generations of the capacity of understanding and living the meanings and values of their heritage[11].

3.3.2 Algorithmic Cultural Appropriation and Prevention

With the long history of cultural appropriation and economic exploitation in the fashion industry, the ease of use of off the shelf generative models and the magnitude of data that is easily available on the internet will only exacerbate the problem. For example, a cultural clothing item can be taken and used in an inappropriate way using style transfers or generative models. How can we prevent this algorithmic cultural appropriation?

One of our contributions in this work is to make a first step towards cultural representation in fashion AI research. Working with Seshweshwe fashion generation brings a significant and important question. Where does the boundary of *algorithmic cultural appropriation* and representation(or appreciation) sit?

How can an artist draw an inspiration and depict a cultural other? In words of Helen Fang[12], an artist can still tell a story through informed ways that do not limit creativity. She also adds that appreciation and representation have to come from a respectful intent, with a thorough consideration of impact and awareness of contextual complexities such as commercialization, power dynamic and cultural sustainability.

Given the rate at which algorithms can quickly generate designs when compared to non-digital ways of creating designs and the scale at which this can be

[10] https://www.culturalintellectualproperty.com/cultural-sustainability-in-fashion

[11] https://www.youtube.com/watch?v=twHCsVPupXo, Monica Boța-Moisin, founder of Cultural Intelectual Property Rights initiative, which influenced a lot of our thinking about the concept of cultural sustainability in generative fashion. More of her work can be found here: https://www.culturalintellectualproperty.com/library

[12] https://www.youtube.com/watch?v=4wY5S3pfPis

achieved, algorithmic fashion designers are in a position of power and advantage. This power and privilege comes with great responsibility.

Use of cultural elements for fashion generation should be regulated. Our recommendation for preventing algorithmic cultural appropriation is as follows:

- Present the origins, history and cultural significance of the fashion items you generate and train models with.
- The cultural fashion items should be adopted within the same context that is meant to be used.
- Make it part of their economic activity: In fashion design there is often a financial aspect, we should make sure that people who are part of that culture are the main beneficiaries of proceeds from the sale of any of their cultural elements.
- In cases where consent was not given, the dataset used for research purposes should not be released or distributed.

By mentioning the origins and history of cultural fashion items one would be celebrating and appreciating the said culture and contributing to cultural sustainability. Proper adoption will ensure that cultural elements do not lose their cultural significance and future generations can still inherit their culture.

3.3.3 The use of Seshweshwe fashion dataset

The images of the dataset, are obtained from an image search engine[13] using specific keywords.
This dataset and application are formed to introduce and represent the cultural aspects of the Seshweshwe fabrics and fashion. We contribute towards cultural sustainability by presenting the history and origins of the Seshweshwe fabric and also did not use it out of its cultural context. Our application also introduces a relatively affordable design method for the benefit of the informal cultural fashion sector.
The use of this dataset in this work also serves as a case study for introducing the concept of algorithmic cultural appropriation, potential harms and how we can prevent it from happening. We don't intend to release the current version of the dataset

[13]Google Images https://www.google.com/imghp

for ethical considerations related to the release of the dataset and how it was collected. In addition to this, all human faces were removed from each of the images used.

Now that we have reflected on some potential harms that might come as a result of misusing a cultural fashion dataset and have proposed some responsible and proper ways of practice when dealing with a cultural fashion dataset, we proceed to present the generative aspect of our work.

Chapter 4

Sketch to Image Translation with Sesheshwe Dataset

To generate African fashion images from line drawings we adopt image-to-image translation techniques Pix2pix [37]. In this chapter we describe this method in detail. We discuss the architectures, objective functions, model evaluation metrics and other relevant training details. Lastly, we present and discuss results.

Image-to-image translation is a transformation of an image from one domain to another. As defined in [37]. Similar to language translation, image-to-image translation is the task of changing the representation of a scene in an image to another possible representation in a different domain.

The Pix2pix GAN methodology enables automated translation with the requirement of paired instances of images. The images from a source and target domain have to resemble the same scene in each instance. For example, daytime photographs and night-time photograph of the same scene make up a good pair, black and white images and colourised images of the same scene also make a good pair. The idea is that the overall scene should be common although the textures can come from different domains.

In the following sections we describe the Pix2pix architecture and give details of the objective function.

4.1 Pix2pix Architecture

Pix2Pix GAN is made up of a generator network and a discriminator network much as a regular GAN would. The generator model is presented with an input image (from the source domain X) and produces a transformed form of the input image. The discriminator model is presented with an input image A joined with a real image Y (from the target domain) or a predicted image Y′ (produced by the generator model) and must decide if the joined pair is real or false. This relation is further illustrated in Fig. 4.1. Ultimately, the generator model is learned simultaneously to trick the discriminator model and to reduce the loss between the predicted image and the target image.
Hence, the Pix2Pix model must be learned on datasets that consist of a paired source domain and target domain.

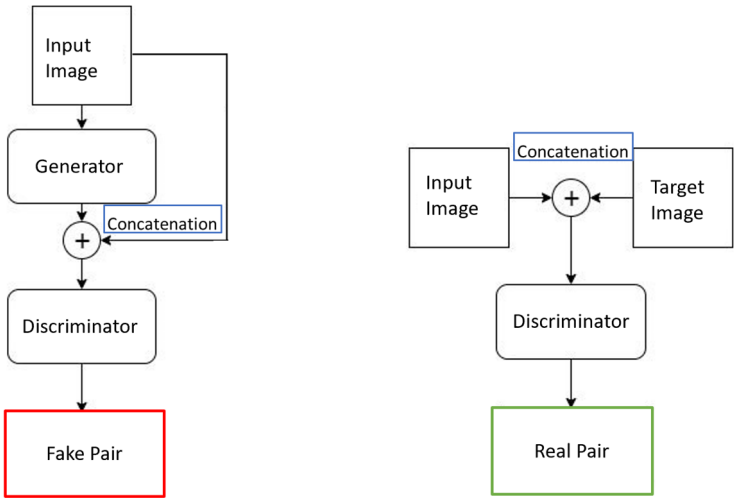


FIGURE 4.1: A summary of the Pix2pix GAN architecture [91].

The Pix2Pix model architecture consists of a detailed configuration of the generator network, the discriminator network, an objective function and an optimization technique. The usual Convolution-Batch Normalization-ReLU [35] layers form part of the building blocks of the generator and discriminator networks. Detailed layer specifications are given in Section 4.4. In the following subsections we look carefully at the two model architectures and the objective function used to refine the

parameters of the model.

In the following section we give details of the generator architecture.

4.1.1 Generator Architecture

For the generator, instead of the famous encoder-decoder architecture, the Pix2pix GAN uses the U-Net architecture [68], which was originally designed for biomedical image segmentation. There are two parts to this architecture. The first is the contraction famously known as the encoder. The encoder is just a conventional stack of layers of convolution and max pooling operations. This part of the architecture helps the model extract the context of the image, that is to identify "WHAT" is on the image. Up to this point a lot of information about "WHERE" the object is located on the image is lost.

The second part is the expansion symmetric to the contraction and is famously known as the decoder. The decoder consists of transposed convolutions which help with the construction of the new image. There are skip connections between the encoder and decoder which allow accurate localization. Thus aiming to restore information about "WHERE" the object is located on the image.

To archive this restoration of information the inputs of the transposed convolution operations are concatenated with feature maps from the encoder. More Specifically, every layer j of the network is concatenated with layer $n - j$ on the decoder part of the architecture, with n being the total number of layers. This creates skip connections between the encoder and decoder which enables localization information sharing.

These skip connections are very important to this research. The aim was to translate a sketch of a Shweshwe dress into an image of the same dress design. This means we needed the design of the dress to stay the same while transforming the appearance from a sketch to an image. Figure 4.2 below shows an illustrative U-net

architecture.

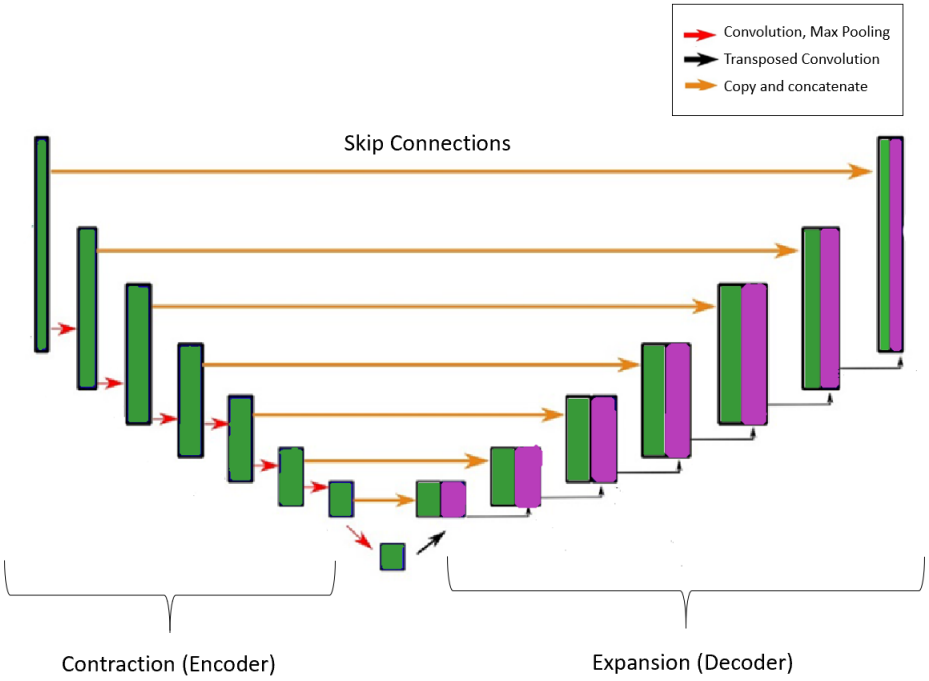


FIGURE 4.2: An illustrative U-net architecture.

Next, we discuss the discriminator architecture.

4.1.2 Discriminator Architecture

The Pix2Pix model uses a PatchGAN for its discriminator model. Instead of classifying the whole input image as real or fake, the PatchGAN is a deep convolutional neural network model used to classify patches of an input image as real or fake. The output of regular GAN discriminator is one number between 0 and 1. The PatchGAN outputs a 2-dimensional array (a matrix) of numbers of values between 0 and 1 (see Fig. 4.3).

Each element on the matrix represents a specific area on the input image, that is smaller than the image size hence the name patch, if the input image is of size $N \times N$ the patch is set to size $M \times M$ where $M < N$.

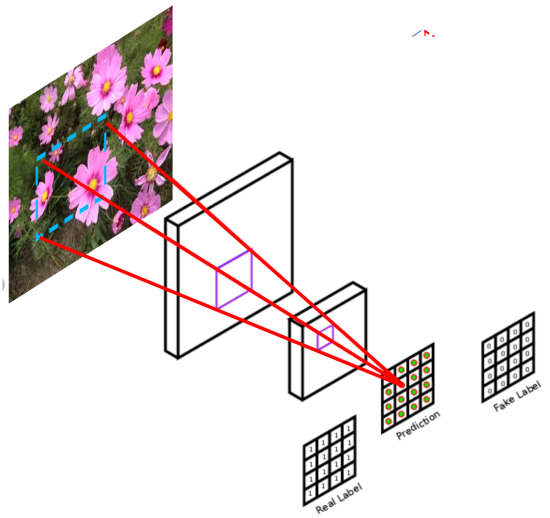


FIGURE 4.3: An illustrative PatchGAN architecture[1]

This 2D array is then averaged into a single value to get a fake or real prediction for the whole input image. These patch-by-patch independent predictions play a very critical role in helping the generator to capture the textures that are present in the input images.

In the following section we discuss the Pix2pix GAN Objective function.

4.2 The Pix2pix GAN Objective function

The discriminator network is trained separately to distinguish the generated images from the authentic ones. The discriminator loss is the same as the standard conditional adversarial loss which minimizes the negative log probability of the difference between the generated image and the authentic target images. Given two domains X and Y where X is the source domain and Y the target domain, we denote the discriminator and generator by D and G respectively. Then, the adversarial loss is given by

$$\mathcal{K}_{cGAN}(G,D) = \mathbb{E}_{x,y}[logD(x,y)] + \mathbb{E}_{x}[log(1 - D(x,G(x))] \tag{4.1}$$

The discriminator model has fewer parameters when compared to the generator thus it learns much faster than the generator. As a result, the discriminator is slowed down by multiplying its loss by 0.5, $\mathcal{L}_{cGAN}(G, D) = 0.5 * \mathcal{K}_{cGAN}(G, D)$. The adversarial loss from the discriminator and the L1 loss (mean absolute pixel difference) between the generated image and the target image, are summed up together to update the generator network. The adversarial loss directs the generator towards generating images that are likely to come from the target domain, while the L1 loss helps to restrict the generator to an output that resembles a specific image in the target domain.

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y}[||y - G(x)||_1] \tag{4.2}$$

Overall, the objective function is given by:

$$\mathcal{L}^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \tag{4.3}$$

Where λ controls the relative importance of the L1 loss.

4.3 Evaluation Metrics

To measure the quality of images produced we made use of the Frechet Inception Distance (FID) [33]. The FID is a measure generally used to assess the quality of produced data. It has been developed especially to assess the performance of GANs. For this experiment FID measure has been used to calculate the distance between the original and regenerated data. The lower the FID score is, the better the generated images are. In a case where two sets of images are the same, the FID score is expected to be zero.

The FID score uses the inception v3 model [78], a convolution neural network used in image recognition. The FID makes use of the last pooling layer prior to the output classification of images of Inception Net [77] to compare low-dimensional, learned continuous vector representations of the real and the generated data. The FID views these vector representations as a continuous multivariate normal distributions. The mean and the co-variance are estimated for both the synthesised data

and the target data. Then, the Frechet distance between these two distributions is computed as [33]

$$distance = ||\mu_s - \mu_t||^2 + Tr(C_s + C_t - 2 \cdot \sqrt{C_s \cdot C_t}) \tag{4.4}$$

Where μ_s and μ_t are the means of the said vector representations from the target data distribution and the synthesised distribution, C_s and C_t are corresponding covariance matrices.

In the following section we present our experimental setup which provides specific information about the libraries used, data prepossessing techniques, detailed layer specifications for each of our models and the optimization strategies used.

4.4 Experimental Setup

All computations carried out in this work were carried out with the python programming language. The neural network models were implemented using the TensorFlow [1] library with Keras [12] which uses automatic differentiation to enable the user to define a neural network model as a computational graph and have backpropagation computed automatically.

Given the very small size of our data, to improve our model performance we increased the variety of the training data by resizing the training images to bigger height and width of size 286×286 then randomly cropped them to the target size of 256×256. We also applied random horizontal flipping, random ± 15 degree rotations and added salt-pepper noise.
The testing images were only resized to the target size of 256×256. Finally both training and testing images were normalized to the range $[-1, 1]$ before being fed into the models.

Next, we detail layer specifications for each of our models.

4.4.1 Network architectures

We adopt network architectures from [37]. Let Kj be a Convolution-BatchNorm-ReLU layer where j is the number of kernels. KDj represents a Convolution-BatchNorm-Dropout-ReLU layer where the dropout rate is varied between 30% and 60%. Here all convolution operations are applied with stride 2 and kernels of size 4×4 unless stated otherwise. Convolutions in the encoder, and in the discriminator, downsample by a factor of 2 and in the decoder they upsample by a factor of 2.

Generator architecture

The U-Net encoder-decoder architecture consists of:
encoder:
$K64 - K128 - K256 - K512 - K512 - K512 - K512 - K512$
decoder:
$KD512 - KD1024 - KD1024 - K1024 - K1024 - K512 - K256$

Following the last layer of a decoder is a convolution operation which maps to 3 channels and lastly a tanh function. As a slight deviation from the notation above, for the first K64 layer we do not apply Batch-Norm. The encoder has leaky ReLUs with a slope of 0.2 and the decoder has ReLUs that are not leaky (slop of zero)

Discriminator architecture

The 70×70 discriminator architecture is: K64-K128-K256-K512

Following the last layer is a convolution operation which maps to a single value and lastly a Sigmoid function. As a slight deviation from the notation above, for the first K64 layer we do not apply Batch-Norm. We use leaky ReLUs with a slope of 0.2 in every instance of ReLus on the network. To modify the receptive field size for other discriminators we keep the basic architecture the same and vary the depth of the network:

1×1 **discriminator**:
$K64 - K128$ (note, in this special case, all convolutions were 1×1 spatial filters)
15×15 **discriminator**:
K64-K128

140×140 **discriminator**:
K64-K128-K256-K512-K512

In this final section of the chapter we present our model outputs and measure their quality under different circumstances.

4.5 Results and Discussion

In this section, we present and discuss key results of the generative aspect of our work. This section serves to demonstrate an example use case for our seShweshwe dataset that we introduced in section 3.2.2. We report the FID scores of the test set and output images generated through the Pix2pixGAN methodology trained on our seShweshwe dataset.

4.5.1 Results

Figure 4.4 shows the visual results of our efforts in exploring the problem of generating African inspired fashion from sketches using a small dataset. We found that when customised well, Pix2pixGAN produced near realistic looking images even though it was trained on only 400 images. The model worked very well in the task of translating the structure and texture of the input images. The choice of learning rate, batch size, loss, receptive field size of the discriminator and number of discriminator steps for each generator step had a significant impact on the performance of the model.
Even though the model worked exceptionally well in producing outputs that resemble the general structure and texture of the input images, the colourization does need improvement. In sparse regions of the image there tends to be some color spillovers.

FIGURE 4.4: Example Pix2pix results. Each column set of images shows an input on the first row, output on the middle and a target on the bottom row. The model is trained up to 100 epochs. We used AdamW optimizer with Warm Restart [50, 49] between a learning rate range of 3e−4 and 2e−2 on both the Discriminator and Generator network. We used the batch size of 1. Training procedure: We update the discriminator twice then update the generator once in every training iteration. We use a receptive field size of 140 x 140 on the discriminator network. We use the L1 loss in combination with the conditional GAN loss and set the lambda parameter to 100 (see equation 4.3).

Using an AdamW optimizer with Warm Restart (AdamWR) between a learning rate range of 3e−4 and 2e−2 on both the Discriminator and the Generator far out performs the AdamW optimizer with a learning rate of $2e-4$ (without restarts). This can be seen is Fig. 4.5.

For each experiment we apply a 10-fold cross validation process, then compute FID scores which we use to run 1000 bootstraps on each sample of 10 FID scores.

Using AdamW we recorded the lowest avarage FID score of 45.2 after 140 epochs of

training. Introducing warm restarts improved performance throughout the training stages and obtained the lowest avarage FID score of 37.1 after only 100 ephochs. This significantly reduced training time by 29% and dropped the average FID score by 18%. Given that there was almost no overlap in confidence intervals, these results are statistically significant. Although we trained both models up to 150 epochs, this was only for the purpose of comparing the efficiency of AdamWR and AdamW optimizers. For our final output we stopped training at 100 epochs seeing that the test FID scores begin to increase after 100 epochs using the AdamWR optimizer.

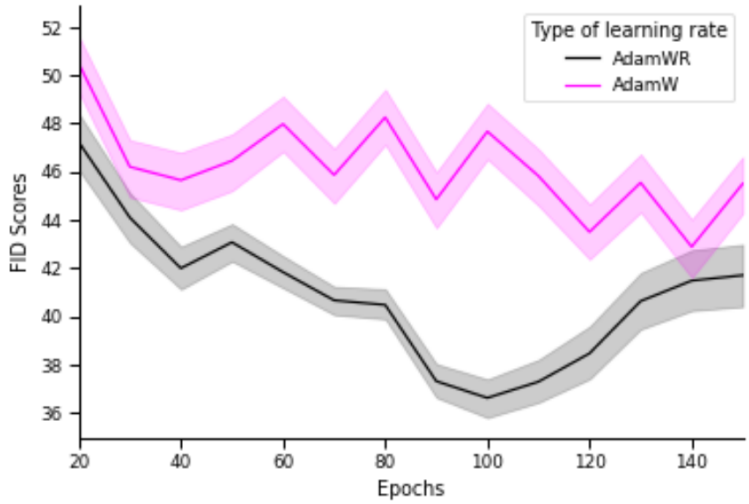


FIGURE 4.5: Test results for AdamW optimizer with Warm Restart between a learning rate range of $3e-4$ and $2e-2$ on both the Discriminator and the Generator vs AdamW optimizer with a learning rate of $2e-4$ (without restarts). The line graphs represent average FID scores of 10 runs and the shaded areas represent the 95% confidence intervals.

The batch size of 1 did not show any unexpected behaviour. It was relatively much slower to train but converged faster than the larger batch sizes. Increasing the batch size for the baseline model resulted in faster progress in training but greatly harmed the model performance. Fig. 4.6 shows that simply increasing the batch sizes (and similarly increasing the learning rate) by a factor of 5 results in higher FID mean scores after 40 training epochs. Instances where we increased the batch sizes without increasing the learning rate resulted in poorer performance as can be

expected. We note that there was no significant difference between the batch size of 2 and 1.

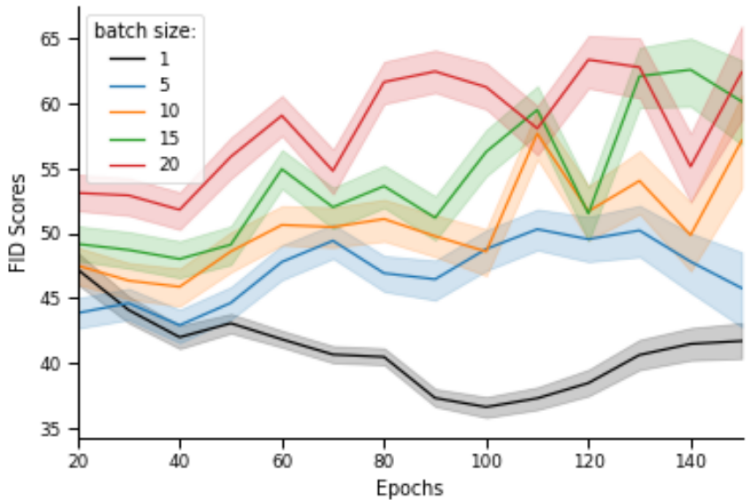


FIGURE 4.6: Test results for increasing batch sizes.

We did a study to analyse the effect of updating the discriminator more than the generator network. In each experiment, the generator is only updated once before updating the discriminator network. The discriminator is given 1, 2, 4 & 6 updates before updating the generator once in each iteration. We refer to these model updates as training steps. In Fig. 4.7 we see that increasing the number of **D** steps per **G** step from 1 to 2 increased learning speed and dropped the FID scores at all the training stages before 120 training epochs. Further increasing the number of **D** steps per **G** step by a factor of 2 dramatically harmed the model performance where FID scores increased by an average of 30% at all the training stages.

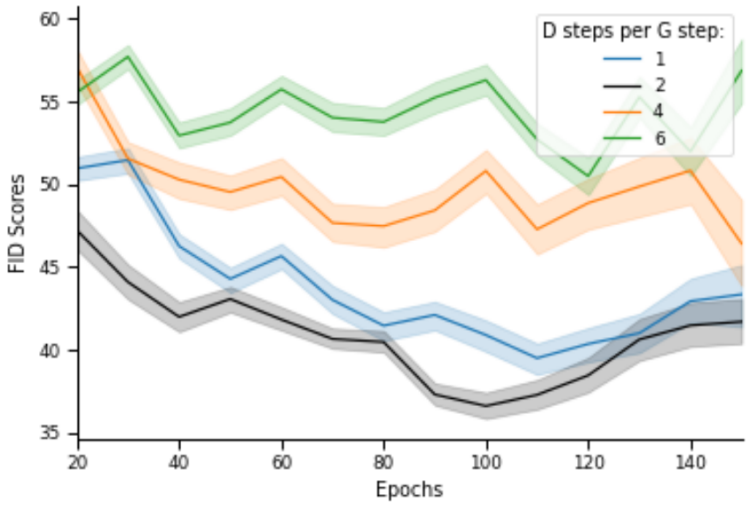


FIGURE 4.7: Number of **D** steps per **G** step experiment.

To understand what role each component of the objective function in Eqn. 4.3 plays and their relative importance we ran an ablation study. In addition to isolating the loss components we also varied the patch size N of our discriminator receptive field. This exercise was effective in comparing the effect of varying the loss versus varying the patch size N of our discriminator receptive field. What was very interesting was seeing how each of the losses fare when paired with different sizes of the discriminator receptive field (namely 1 x 1, 15 x 15, 70 x 70 and 140 x 140). To get the different patch sizes we adjusted the depth of the GAN discriminator. More layers of convolution increase the patch size and vice versa. Figure 4.8 shows qualitative outcomes of the different combinations on our sketch-to-image translation task. L1 on its own results in very blurry images across all the sizes of the discriminator receptive field. The cGAN alone outputs very sharp images across all the receptive field sizes however introduces some discoloration for models with a receptive field size of 1 x 1, 15 x 15, 70 x 70. Having both L1 and cGAN loss with λ set to 100 significantly reduces the discoloration across receptive field sizes. Interestingly, outputs from the model with a receptive field of size 70 x 70 did not have any discolorations across all the losses.

To quantify these outcomes we represented the corresponding mean FID scores on a heat map in Fig. 4.9 and highlighted their statistical significance with a use of a

bar graph in Fig. 4.10. Even though the size of the discriminator receptive field has a considerable impact on quality of the output, it is clear to see that the choice of loss is significantly more important. Increasing the size of the discriminator receptive field improves the FID scores across all the loss choices. These improvements that are however not evident across all the losses on the qualitative results. A combination of the L1 + cGAN loss and a discriminator receptive field of size 140 x 140 gives the best FID scores and has a better looking visual output.

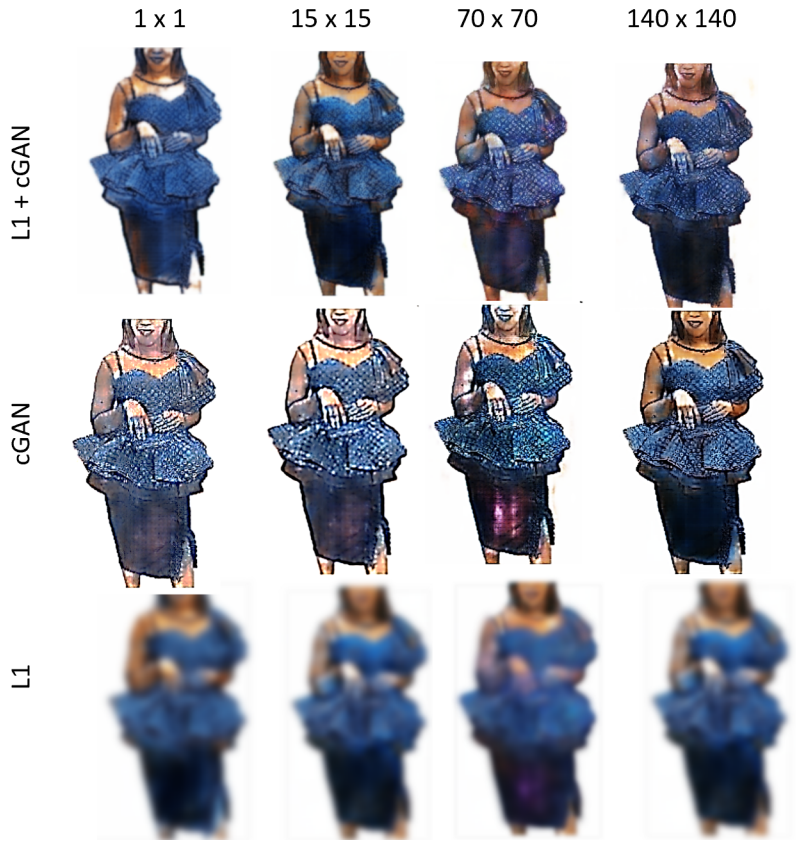


FIGURE 4.8: Example of qualitative results for a combination of different receptive field sizes of the discriminator and different losses

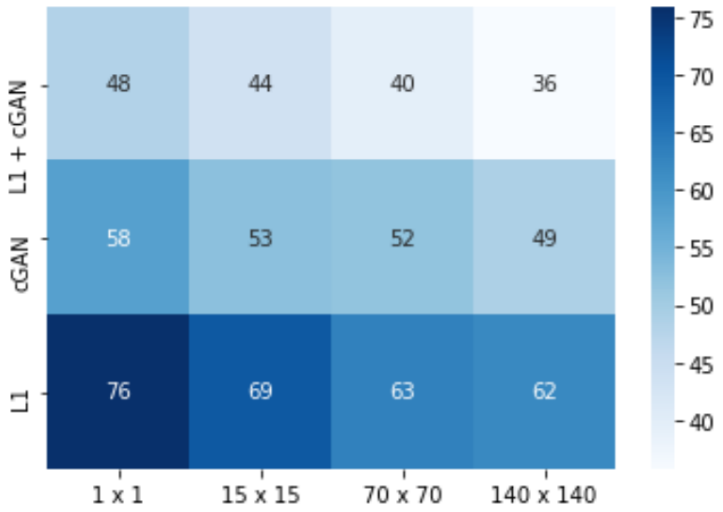


FIGURE 4.9: FID scores heat map for a combination of different receptive field sizes of the discriminator and different losses. For each experiment we apply a 10-fold cross validation process, then compute FID scores which we use to run 1000 bootstraps on each sample of 10 FID scores to obtain the mean FID scores represented on the heat map.

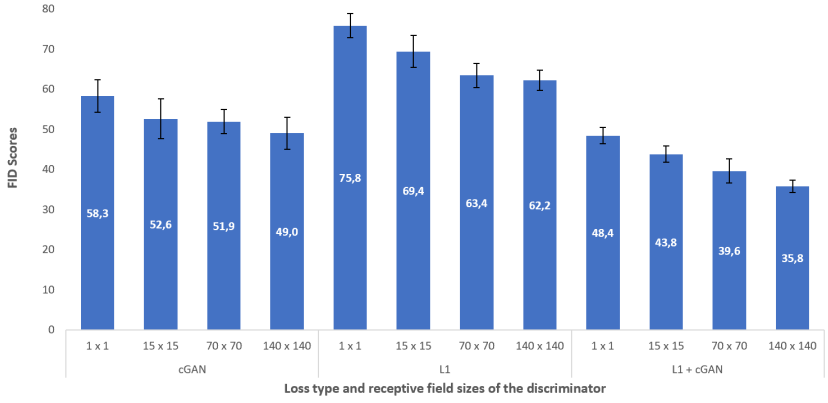


FIGURE 4.10: FID scores bar graph with confidence intervals for each combination of receptive field size of the discriminator and loss type

Although we were able to customize this conditional GAN method to our dataset and produce reasonable outcomes, there were some failure cases (see Fig. 4.11).

Typical failure cases have artifacts in regions where the input image is sparse or where there are rare colours and uncommon inputs.

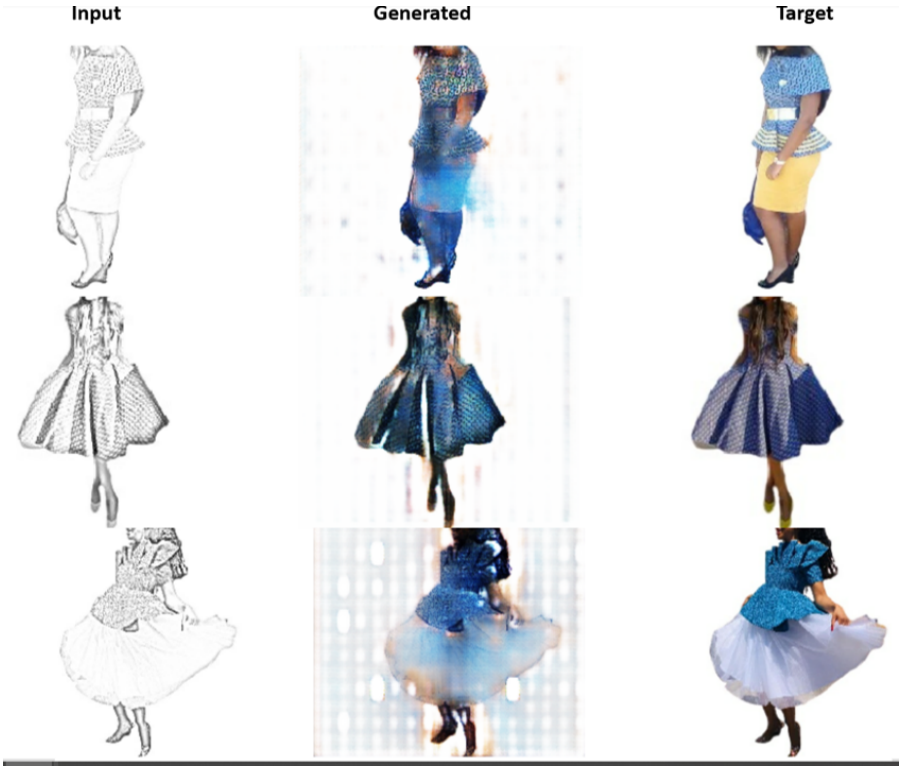


FIGURE 4.11: Examples of failure cases of generated images. The first column shows inputs, followed by generated outputs on the middle column and target images on the third and last column. These results are an examples of some of the worst outputs on our translation task.

4.5.2 Discussion

Our results confirm that with minor tweaks, the pix2pixGAN model can indeed be applied to a variety of paired translation tasks involving diverse datasets. Although we apply random jittering, mirroring and add salt and pepper noise to pre-process the training set, our data is still susceptible to small datasets challenges given that deep learning models require very large amounts of data to train. Our results show that training a deep learning model with a small dataset makes the model very sensitive to the training batch sizes, with larger batch sizes significantly worsening

performance.

Learning rate restarts also played a significant role in improving model performance at all the training stages. This is not a surprising outcome as it is consistent with the results presented in other studies on learning rates [50, 49].
In addition, our results suggest that having a slightly more robust discriminator relative to the generator helps both models to learn faster and perform better, which confirms findings in other studies where updating the discriminator twice before updating the generator worked well [9].
We saw that the L1 loss on its own results in very blurry images while cGAN alone outputs very sharp images with some colour distortions. These results are consistent with the results from the Pix2pix paper even though the experiments were carried out with a different dataset from ours. While the pix2pix paper states that a discriminator receptive field size of 70 x 70 yeilds the best results, we have found that with the Seshweshwe dataset, increasing this size to 140 x 140 further improves the performance of the model. Although deeper model architectures introduce more parameters to the model which require more data to train, in this case decreasing the depth of the discriminator model actually harmed performance and resulted in an increase in FID scores. This can be attributed to the level of complexity of the training data requiring more complex functions to model the data.
These results highlight the relative importance of the depth of our discriminator and the choice of loss function. From the FID scores it is clear that choosing the right loss function for the translation task results in better FID scores even with a shallow discriminator. This is an important contribution of this study towards understanding the best strategies for translation tasks.

Chapter 5

Conclusions and Future Work

Our work makes important contributions towards the diversity of fashion datasets in the field of computer vision, highlights important considerations for image-to-image translation tasks, advocates for a responsible use of cultural fashion datasets towards cultural sustainability and appreciation.
We set out to generate African inspired fashion images using Generative models. In the process we curated the first of its kind, Seshweshwe dataset, the first dataset that consists of African fashion designs. We discussed ethical considerations in relation to AI, cultural fashion and propose responsible and proper ways of practice when dealing with a cultural fashion dataset. We demonstrated the usability of our dataset and provided an example use case for our dataset: sketch-to-image translation task which can be very beneficial for small scale fashion designers to better communicate their ideas (at a much lower cost) and grow their businesses. Finally we presented key results and discussions from our sketch-to-image translation experiments where we customised the pix2pix methodology for our dataset, successfully generated images that are close to real images, analysed different components of the model and highlighted important components that significantly impact performance.
Given that the model worked very well in the task of translating the structure and texture of the input images, this makes it ideal for the purpose of translating a sketch of a fashion design to a colourised fashion image without needing to to do an actual photo shoot that is typically resource intensive.
Instead of propagating the problem of cultural appropriation through algorithmic appropriation we can contribute to cultural sustainability by using the power of deep learning and generative models particularly for promoting representation of

under represented communities and cultures around the world. By simply changing our way of practice and interaction with cultural artifacts and adopt an appreciation approach we can empower under-represented communities with algorithms and celebrate their culture and diversity. Generative models can play a role in helping small and up-coming fashion designers amplify their work and reach a wider audience in a very short amount of time in the absence of financial resources and access to high-end technology often used by big fashion houses.

5.1 Future Work

Even though there are significant contributions out of this study, we also note that there is still a lot of need for further research. We recommend that future work focuses on:

- Further investigation of learning rate efficiency, for example compare warm restarts to a 1-cycle policy for super convergence.
- Explore other model architectures that would be more suitable for translation tasks e.g a residual U-net for the generator would allow for an ever deeper network without loss of information from earlier layers of the decoder and encoder part of the network.
- Explore other generative methods for efficient high-resolution image-to-image translation for small datasets.
- Creation of a fashion dataset that is representative of the cultural fashion in all the regions of the African continent and other regions beyond the continent.
- A very interesting fashion application of translation GANs would be translating a fashion image to a video format that resembles a runway.
- Generating multi-cultural fashion images given text inputs.
- Representation of other cultures within the southen African region.
- Implement the models and test them on real human drawn fashion sketches.

- Besides the significant problem of cultural appropriation, when the training data consists of a massive number of images created by other artists, credit assignment is a crucial question. For future work it would be valuable to tackle questions about the difference between memorization (copying an art piece) and sourcing inspiration and credit be attribution in this case.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-scale machine learning on heterogeneous systems*. 2015.

[2] Small Enterprise Development Agency. "SMME quarterly update 1st quarter 2018. Small Enterprise Development Agency." In: *www.seda.org.za/Publications* (2018), pp. 1–15.

[3] Thessy Yemisi Akinbileje. "Symbolic values of clothing and textiles art in traditional and contemporary Africa". In: *International journal of development and sustainability* 3.4 (2014), pp. 626–641.

[4] Ziad Al-Halah and Kristen Grauman. "From Paris to Berlin: Discovering fashion style influences around the world". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10136–10145.

[5] Malcolm Barnard. *Fashion as communication*. Psychology Press, 2002.

[6] Yuemin Bian and Xiang-Qun Xie. "Generative chemistry: drug discovery with deep learning generative models". In: *Journal of Molecular Modeling* 27.3 (2021), pp. 1–18.

[7] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[8] Christian Bracher, Sebastian Heinz, and Roland Vollgraf. "Fashion DNA: merging content and sales data for recommendation and article mapping". In: *arXiv preprint arXiv:1609.02489* (2016).

[9] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large scale GAN training for high fidelity natural image synthesis". In: *arXiv preprint arXiv:1809.11096* (2018).

[10] KuanTing Chen et al. "Who are the devils wearing prada in new york city?" In: *Proceedings of the 23rd ACM international conference on Multimedia*. 2015, pp. 177–180.

[11] Xi Chen et al. “Infogan: Interpretable representation learning by information maximizing generative adversarial nets”. In: *Advances in neural information processing systems*. 2016, pp. 2172–2180.

[12] François Chollet et al. *keras*. 2015.

[13] Jill Condra. *Encyclopedia of National Dress: Traditional Clothing around the World [2 Volumes]*. ABC-CLIO, 2013.

[14] Justine M Cordwell et al. *The fabrics of culture: The anthropology of clothing and adornment*. De Gruyter Mouton, 1979.

[15] dagama. *Fabric, Fashion and Identity - The story of IsiShweshwe*. URL: https://www.dagama.co.za/five-popular-shweshwe-trends-this-season/. (accessed: 03.08.2020).

[16] Fred Davis. *Fashion, culture, and identity*. University of Chicago Press, 2013.

[17] Emily L Denton, Soumith Chintala, Rob Fergus, et al. “Deep generative image models using a laplacian pyramid of adversarial networks”. In: *Advances in neural information processing systems*. 2015, pp. 1486–1494.

[18] Haoye Dong et al. “Towards multi-pose guided virtual try-on network”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9026–9035.

[19] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. “A learned representation for artistic style”. In: *arXiv preprint arXiv:1610.07629* (2016).

[20] Celia Frank et al. “Forecasting women’s apparel sales using mathematical modeling”. In: *International Journal of Clothing Science and Technology* (2003).

[21] Stefan Frässle et al. “Generative models for clinical applications in computational psychiatry”. In: *Wiley Interdisciplinary Reviews: Cognitive Science* 9.3 (2018), e1460.

[22] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2001.

[23] Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. “Neocognitron: A neural network model for a mechanism of visual pattern recognition”. In: *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 826–834.

[24] Ashwinkumar Ganesan, Tim Oates, et al. “Fashioning with networks: Neural style transfer to design clothes”. In: *arXiv preprint arXiv:1707.09899* (2017).

[25] Yoav Goldberg. “Neural network methods for natural language processing”. In: *Synthesis lectures on human language technologies* 10.1 (2017), pp. 1–309.

[26] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[27] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[28] Guillermo L Grinblat, Lucas C Uzal, and Pablo M Granitto. “Class-splitting generative adversarial networks”. In: *arXiv preprint arXiv:1709.07359* (2017).

[29] Ishaan Gulrajani et al. “Improved training of wasserstein gans”. In: *Advances in neural information processing systems*. 2017, pp. 5767–5777.

[30] Xiaoxiao Guo et al. “Fashion IQ: A New Dataset towards Retrieving Images by Natural Language Feedback”. In: *arXiv preprint arXiv:1905.12794* (2019).

[31] Ahyoung Han, Jihoon Kim, and Jaehong Ahn. “Color Trend Analysis using Machine Learning with Fashion Collection Images”. In: *Clothing and Textiles Research Journal* (2021), p. 0887302X21995948.

[32] Xintong Han et al. “Viton: An image-based virtual try-on network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7543–7552.

[33] Martin Heusel et al. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Advances in neural information processing systems*. 2017, pp. 6626–6637.

[34] Wei-Lin Hsiao et al. “Fashion++: Minimal Edits for Outfit Improvement”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[35] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[36] Daniela Ionescu. “THE ROMANIAN BLOUSE: FROM MATISSE TO QUEEN MARIE OF ROMANIA AND YVES SAINT LAURENT”. MA thesis. 1250 Bellflower Blvd, Long Beach, CA 90840, USA: California State University, Long Beach, school of art, Dec. 2018.

[37] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[38] W. Kang et al. “Visually-Aware Fashion Recommendation and Design with Generative Image Models”. In: *2017 IEEE International Conference on Data Mining (ICDM)*. 2017, pp. 207–216. DOI: 10.1109/ICDM.2017.30.

[39] Wang-Cheng Kang et al. “Complete the look: Scene-based complementary product recommendation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10532–10541.

[40] Tero Karras et al. “Progressive growing of gans for improved quality, stability, and variation”. In: *arXiv preprint arXiv:1710.10196* (2017).

[41] Taeksoo Kim et al. “Learning to discover cross-domain relations with generative adversarial networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1857–1865.

[42] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[44] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[45] Yann LeCun, Yoshua Bengio, et al. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[46] Yann LeCun et al. “Generalization and network design strategies”. In: *Connectionism in perspective* 19 (1989), pp. 143–155.

[47] Ziwei Liu et al. “Deepfashion: Powering robust clothes recognition and retrieval with rich annotations”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1096–1104.

[48] Babak Loni et al. “Fashion 10000: an enriched social image dataset for fashion and clothing”. In: *Proceedings of the 5th ACM Multimedia Systems Conference*. 2014, pp. 41–46.

[49] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017).

[50] Ilya Loshchilov and Frank Hutter. “Sgdr: Stochastic gradient descent with warm restarts”. In: *arXiv preprint arXiv:1608.03983* (2016).

[51] Utkarsh Mall et al. “Geostyle: Discovering fashion trends and events”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 411–420.

[52] Arjovsky Martin and B Lon. “Towards principled methods for training generative adversarial networks”. In: *NIPS 2016 Workshop on Adversarial Training. In review for ICLR*. Vol. 2016. 2017.

[53] SC Martin Arjovsky and Leon Bottou. “Wasserstein Generative Adversarial Networks”. In: *Proceedings of the 34 th International Conference on Machine Learning, Sydney, Australia*. 2017.

[54] Mehdi Mirza and Simon Osindero. “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784* (2014).

[55] Takeru Miyato and Masanori Koyama. “cGANs with projection discriminator”. In: *arXiv preprint arXiv:1802.05637* (2018).

[56] Shakir Mohamed et al. “Monte carlo gradient estimation in machine learning”. In: *arXiv preprint arXiv:1906.10652* (2019).

[57] Iziko Museum. *Fabric, Fashion and Identity - The story of IsiShweshwe*. URL: https://artsandculture.google.com/exhibit/fabric-fashion-and-identity-the-story-of-isishweshwe-south-african-national-gallery/PQJSfVGhOBzHKg?hl=en. (accessed: 03.08.2020).

[58] Iziko Museum. *Fabric, Fashion and Identity - The story of IsiShweshwe*. URL: https://artsandculture.google.com/exhibit/fabric-fashion-and-identity-the-story-of-isishweshwe-south-african-national-gallery/PQJSfVGhOBzHKg?hl=en. (accessed: 03.08.2020).

[59] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Icml*. 2010.

[60] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, 2015.

[61] 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*. JMLR. org. 2017, pp. 2642–2651.

[62] Guim Perarnau et al. "Invertible conditional gans for image editing". In: *arXiv preprint arXiv:1611.06355* (2016).

[63] V Powell. "Image Kernels - Explained Visually Image Kernels - Explained Visually." In: *article* (2019).

[64] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[65] Louis B Rall and George F Corliss. "An introduction to automatic differentiation". In: *Computational Differentiation: Techniques, Applications, and Tools* 89 (1996).

[66] Abhianv Ravi et al. "Teaching DNNs to design fast fashion". In: *arXiv preprint arXiv:1906.12159* (2019).

[67] Scott Reed et al. "Generative adversarial text to image synthesis". In: *arXiv preprint arXiv:1605.05396* (2016).

[68] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[69] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[70] Negar Rostamzadeh et al. “Fashion-gen: The generative fashion dataset and challenge”. In: *arXiv preprint arXiv:1806.08317* (2018).

[71] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[72] Dominik Scherer, Andreas Müller, and Sven Behnke. “Evaluation of pooling operations in convolutional architectures for object recognition”. In: *International conference on artificial neural networks*. Springer. 2010, pp. 92–101.

[73] Haichao Shi et al. “SSGAN: secure steganography based on generative adversarial networks”. In: *Pacific Rim Conference on Multimedia*. Springer. 2017, pp. 534–544.

[74] Jung-Eun Shin et al. “Protein design and variant prediction using autoregressive generative models”. In: *Nature communications* 12.1 (2021), pp. 1–11.

[75] Jen Snowball and Aviwe Mapuma. “Creative industries micro-enterprises and informality: a case study of the Shweshwe sewing industry in South Africa”. In: *Journal of Cultural Economy* (2020), pp. 1–15.

[76] Zhan-Li Sun et al. “Sales forecasting using extreme learning machine with applications in fashion retailing”. In: *Decision Support Systems* 46.1 (2008), pp. 411–419.

[77] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[78] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[79] Les M Sztandera, Celia Frank, and Balaji Vemulapali. “Predicting women’s apparel sales by soft computing”. In: *International Conference on Artificial Intelligence and Soft Computing*. Springer. 2004, pp. 1193–1198.

[80] Biodiversity Unit. “MAKING THINGS WORK: ABORIGINAL AND TORRES STRAIT ISLANDER INVOLVEMENT IN BIOREGIONAL PLANNING”. In: (1996).

[81] Sirion Vittayakorn et al. “Runway to realway: Visual analysis of fashion”. In: *2015 IEEE Winter Conference on Applications of Computer Vision*. IEEE. 2015, pp. 951–958.

[82] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. “Generating videos with scene dynamics”. In: *Advances in neural information processing systems* 29 (2016), pp. 613–621.

[83] Yaxing Wang, Lichao Zhang, and Joost Van De Weijer. “Ensembles of generative adversarial networks”. In: *arXiv preprint arXiv:1612.00991* (2016).

[84] Kai Xiao et al. “Noise or signal: The role of image backgrounds in object recognition”. In: *arXiv preprint arXiv:2006.09994* (2020).

[85] Wei Yang, Ping Luo, and Liang Lin. “Clothing co-parsing by joint image segmentation and labeling”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 3182–3189.

[86] James O Young. *Cultural appropriation and the arts*. John Wiley & Sons, 2010.

[87] James O Young and Susan Haley. “‘Nothing Comes from Nowhere’: Reflections on Cultural Appropriation as the Representation of Other Cultures”. In: *The ethics of cultural appropriation* (2009), pp. 268–289.

[88] Roshanak Zakizadeh et al. “Improving the Annotation of DeepFashion Images for Fine-grained Attribute Recognition”. In: *arXiv preprint arXiv:1807.11674* (2018).

[89] Han Zhang et al. “Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5907–5915.

[90] Shuai Zheng et al. “Modanet: A large-scale street fashion dataset with polygon annotations”. In: *Proceedings of the 26th ACM international conference on Multimedia*. 2018, pp. 1670–1678.

[91] Jun-Yan Zhu et al. “Unpaired image-to-image translation using cycle-consistent adversarial networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232.

[92] Shizhan Zhu et al. “Be your own prada: Fashion synthesis with structural coherence”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1680–1688.

[93] Zhen Zhu et al. “Progressive pose attention transfer for person image generation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2347–2356.