Neuroevolutionary Training of Deep Convolutional Generative Adversarial Networks

By

Kaitav Mehta

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2019

Neuroevolutionary Training of Deep Convolutional Generative Adversarial Networks

by

Kaitav Nayankumar Mehta

APPROVED BY:

K. Pfaff
Faculty of Nursing

A. Ngom
School of Computer Science

Z. Kobti, Advisor
School of Computer Science

September 24, 2019

DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION

I. Co-Authorship

I hereby declare that this thesis incorporates material that is a result of research conducted under the supervision of Dr. Ziad Kobti (Advisor). Dr. Kathryn Pfaff and Dr. Susan Fox contributed in revising the publication. In all cases, the key ideas, primary contributions, experimental designs, data analysis, interpretation, and writing were performed by the author, and the contribution of co-authors was primarily through providing feedback on the refinement of ideas and editing of the manuscripts.

I am aware of the University of Windsor Senate Policy on Authorship, and , certify that I have correctly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Previous Publication

This thesis includes one original paper that has been previously published for publication in peer-reviewed journals, as follows:

Section	Publication title/full citation	Publication status
4.2, 4.5, 5.8	Kaitav Mehta, Ziad Kobti, Kathryn Pfaff and Susan Fox, "*Culturally evolved GANs for generating fake Stroke Faces*", in ISCC Workshops- ICTS4eHealth 2019	Published

ABSTRACT

Recent developments in Deep Learning are noteworthy when it comes to learning the probability distribution of points through neural networks, and one of the crucial parts for such progress is because of Generative Adversarial Networks (GANs) [1]. In GANs, two neural networks, Generator and Discriminator, compete amongst each other to learn the probability distribution of points in visual pictures. A lot of research has been conducted to overcome the challenges of GANs which include training instability, mode collapse and vanishing gradient. However, there was no significant proof found on whether modern techniques consistently outperform vanilla GANs, and it turns out that different advanced techniques distinctively perform on different datasets. In this thesis, we propose two neuroevolutionary training techniques for deep convolutional GANs. We evolve the deep GANs architecture in low data regime. Using Fréchet Inception Distance (FID) score as the fitness function, we select the best deep convolutional topography generated by the evolutionary algorithm. The parameters of the best-selected individuals are maintained throughout the generations, and we continue to train the population until individuals demonstrate convergence. We compare our approach with the Vanilla GANs, Deep Convolutional GANs and COEGAN. Our experiments show that an evolutionary algorithm-based training technique gives a lower FID score than those of benchmark models. A lower FID score results in better image quality and diversity in the generated images.

DEDICATION

This Thesis is dedicated to my father, Nayan Mehta, my mother, Rita Mehta, and my brother, Kaivan Mehta.

ACKNOWLEDGEMENTS

I want to acknowledge my parents and family members Hiren Mehta and Nishita Mehta for providing me with enormous support to carry out this research.

I appreciate the time and vast knowledge of my supervisor Dr. Ziad Kobti. I want to thank Dr. Kobti for guiding me throughout my master's degree from course selection to building research skills. I want to thank the members of my thesis committee, Dr. Alioune Ngom and Dr. Kathryn Pfaff, for their helpful comments and suggestions for the completion of this thesis.

I would like to sincerely thank my friends, who have been vital support throughout my work. Finally, but not least, I want to thank God for giving me motivation and belief to carry out this research successfully.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Overview

Facebook Artificial Intelligence's research director Yann LeCun described generative adversarial training to be *"the most interesting idea in the last 10 years of Machine Learning."* [2]

Generative Adversarial Networks (GANs) were created by Ian Goodfellow in 2014 [1]. GANs are a machine learning approach qualified to generate novel synthetic outputs across a space of provided training examples.

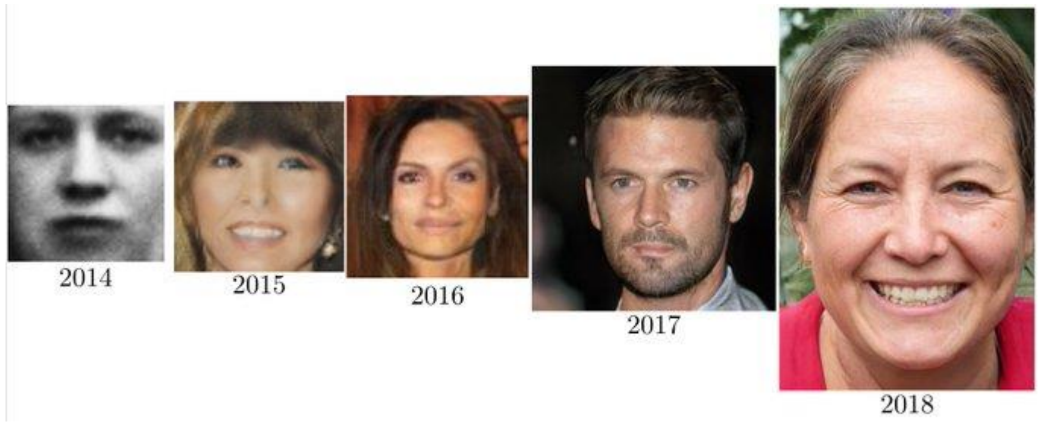


Figure 1 Progression of face generation [3]

Figure 1 shows the five years of progression of GANs for generating human faces [3]. Since 2014, the GAN progress has exploded and has led to generate realistic outputs. Today, GANs are able to output many different types of media, being in the form of images, videos, text and audio. These different synthetic outputs can be used to train different machine learning models. However, the training of GANs is a difficult task.

There are still fundamentally unresolved issues like vanishing gradient and mode collapse in GANs [4].

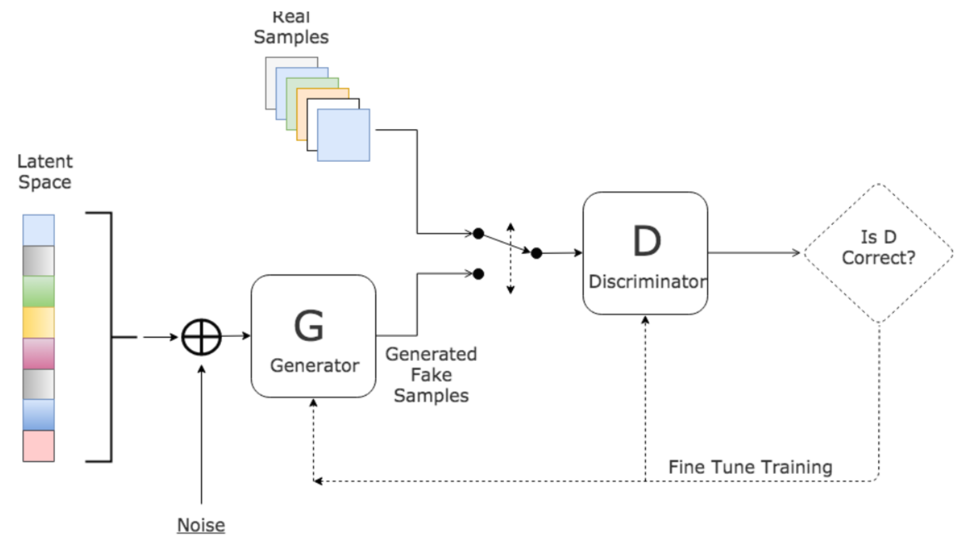


Figure 2 Schematic Representation of GANs [5]

Figure 2 shows a schematic representation of GANs. GANs combines two deep neural networks playing a minimax game with each other. The discriminator network tries to distinguish whether the sample is real or fake. While the other neural network, called a generator, tries to create fake samples that the discriminator thinks is real.

The generator never sees the original dataset, and it must learn to generate realistic samples by receiving criticism from the discriminator. This process is called adversarial loss, and when implemented correctly it works very well. The more the generator and the discriminator play this game, the more they advance each other's skills. The discriminator becomes very good at predicting synthetic data while the generator learns to create information that is identical from what is observed in the real world [1] [5].

Once the generator masters the distribution of the training samples, we can sample the generator n times for pragmatic outputs such as images, videos, text, numerical simulations, and just about anything else one can imagine. Further, the discriminator is also used for different tasks such as distinguishing outliers, abnormalities and anything which is not ordinary. This could be very beneficial in fields such as cybersecurity, radiology, astronomy, and manufacturing [5].

Adversarial training is also proven to be useful in many different applications like domain adaptation, data augmentation, and image-to-image translation [6]. Figure 3 shows one such example of image-to-image translation, wherein the eye vessels are translated to a fundus image [6].

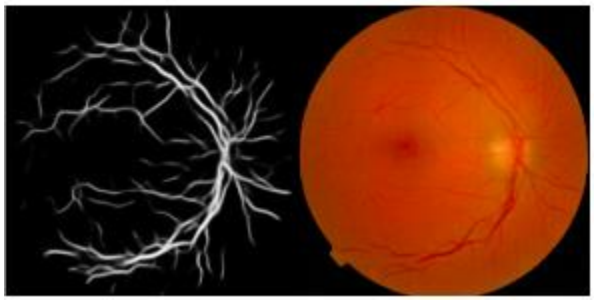


Figure 3 Vessels to fundus image [6]

However, training such GANs requires a large dataset. In many realistic settings, such as the medical domain, we need to achieve goals with a limited dataset. In such cases, deep neural networks fall short, overfitting on the training set and producing poor generalization on the test set [7]. To overcome such issues, it is possible to generate more data from existing data by applying data augmentation techniques to the original dataset. However, standard data augmentation produces only limited alternatives [7]. Thus, we want to generate images in such low scale data domain. GANs do not create new data; rather, they produce new data with different properties, which can capture many different aspects of the original data. All of these different aspects can be captured by a classifier to improve the accuracy of machine learning models.

Further, to generate such synthetic images, the GANs are required to be well trained. The training of the GANs can be improved by optimizing the hyperparameters and architecture of the deep neural networks. Generally, the topology and hyperparameters are chosen empirically, which takes a lot of human time. The method to search the accurate architecture can be automated. One such approach to request architecture is using neuroevolution [8].

Many approaches are proposed to find the architecture of neural networks, but decidedly less research has been conducted for evolving deep GANs. In previous works, there are not many components which are taken into consideration for evolving GANs. In E-GANs [9], for example, only weight parameters of neurons are considered for evolution.

Thus, in this thesis, we tried to generate images in the shallow dataset domain. With the image generation, we also stabilize the training of the GANs and search for the best GAN architecture.

1.2 Problem Definition

In large scale datasets, GANs were improved to generate high-quality images [10]. Despite the progress in GANs, there are open issues regarding the training of the GANs. Most common issues, like mode collapse and vanishing gradient, make the training of the GANs difficult. Various strategies have been proposed to minimize these issues, but fundamentally, the issue remains unresolved [11].

The Generator and Discriminator are deep neural networks in GANs. The architecture and hyperparameters of these networks are empirically determined by spending human time in the repetitive task such as fine-tuning the network. However, some techniques can automate the design of the networks. Neuroevolution is a technique that uses evolutionary algorithms to automate the design of neural network architecture.

Formally, we can define our problem as follows:
Let Generator(G) and Discriminator(D) hyperparameters be defined by following tuples:

$$G = \left(N, Wi, \sigma(x)\right) \;\; D = (N, Wi, \sigma(x)) \tag{1}$$

Where:

N: represents the number of layers of the network

Wi: is the weight initialization of each network

$\sigma(x)$: represents the activation function applied at each layer of both the networks

We want to search for the best trained GAN architecture. In both the generator and discriminator we are searching for the number of layers, activation function for each layer, output channels for each layer and weight initialization. Also, the search for the best loss function to train the weight parameters of the networks is automated.

An individual GAN equation is expressed as below:

$$GAN_i = \{G, D, Loss\} \tag{2}$$

In Equation 2, G represents Generator Neural Network and D represents Discriminator Neural Network, and Loss represents Loss function required to find the gradients to learn the weight parameters of the network using backpropagation.

Intuitively we want to test if the introduction of an evolutionary algorithm in deep GANs can design the topography of the network. Thus, we want evolutionary algorithms to automate deep convolutional GAN architecture search and increase the training stability in low data regime.

1.3 Motivation

Generative Adversarial Networks became remarkable, presenting impressive results mainly for image synthesis in the field of computer vision. Several works improving the GAN model have been published in the large-scale datasets. However, there are still fundamentally unresolved issues related to the training of the GANs. Vanishing gradient and mode collapse are the most common issues, making the training of GANs hard [12]. There are different strategies to minimize these issues, but radically, they remain unresolved. Another issue, not only related to GANs but also to neural networks, is the need to decide a network architecture previously. In this case, the topology and hyperparameters are chosen empirically or require careful design by experts. Hence, spending human time in a repetitive task such as fine-tuning the network.

Ideally, one would want to automate the method to generate the right neural architecture. One approach to generate these architectures is called Neuroevolution. Neuroevolution is the application of an evolutionary algorithm to automate the design of neural architectures [13]. Standard evolutionary algorithm has been successful in solving diverse and complex problems. Evolutionary computations mimic natural evolution, which is based on the principle of genetic inheritance. In natural systems, genetic evolution is a slow process. Cultural evolution enables the population to adapt to their changing environments at a rate that exceeds of biological evolution. Neural network design is inspired by the human brain neurons structure. The human brain has evolved over a long time, from very simple worm brains 500 million years ago to a diversity of modern structures today. We, humans, became the top predator when we started evolving culturally [14] [15] and hence, our motivation for using Cultural Algorithms.

1.4 Thesis Statement

Machine learning (ML) is used in order to make predictions or decisions without being explicitly programmed to perform the task. The larger the dataset, the greater the accuracy in the training of the ML models and better the performance. In real-world settings, the size of the datasets is small, which makes it challenging to perform well using these machine learning algorithms. As such the general goal is to explore methods to create synthetic dataset for such domains.
Recently, GANs was improved to generate high-resolution images in large-scale datasets [10]. However, there are still open problems regarding the training of the GANs. Our

hypothesis is neuroevolutionary training can resolve GAN training issues and can generate better images even if there is a small training dataset.

There are different approaches to automate the discovery of GAN architecture. One of them is to use a grid search, wherein all the possible combinations of the network architecture are tested, and the best one is selected. Such an approach can be very time-consuming. Second, a very well appreciated approach called AutoML involves usage of reinforcement learning, where the AI agents learn by trial-and-error in an environment without direct supervision. AutoML based techniques have made a significant impact in searching the different types of backbone architecture for deep neural networks [13]. One drawback of the AutoML approach is that it requires tremendous computing resources and data. However, a recent study has shown that evolutionary algorithms are a competitive alternative to such deep reinforcement approaches. Moreover, it is proven that evolutionary algorithms are substantially faster than deep reinforcement learning methods [16]. Hence, in our approach we have selected evolutionary strategies to automate the architecture search of deep convolutional GANs (DCGAN). Also, in our approach, we are using domain knowledge in cultural algorithm based neuroevolutionary training of DCGAN. By using domain knowledge, we propose that the hyperparameter search space will be dramatically reduced and eventually generate sharper and diverse realistic images.

We expect to see the improvement of the training stability, better generation of images and automatic discovery of efficient deep convolutional GANs topologies by the introduction of evolutionary algorithms.

1.5 Thesis Contribution

This thesis represents the problem of training deep convolutional GANs with a small dataset. With the training, GANs topography is also evolved using two evolutionary algorithms. Genetic algorithm and Cultural algorithm are used with the combination of neuroevolution of augmented topologies (NEAT) [8].

Moreover, the proposed approaches are tested on three different datasets. The three datasets are MNIST [17], F-MNIST [18] and Stroke Face. We also introduced the Stroke face dataset to test our approach in the low-scale dataset. To compare the quality and diversity of generated images, we have used the FID score [19], which is currently the state-of-the-art metric to evaluate GANs. In this thesis, we also demonstrate the transference of weight parameters of deep neural networks throughout the generations of the evolutionary algorithm.

Thus, this thesis contributes by implementing the following strategies:

- Genetic neuroevolutionary training of deep convolutional GANs (GAGAN)
- Cultural neuroevolutionary training of deep convolutional GANs (CAGAN)

1.6 Thesis Organization

The rest of the thesis is organized in the following way:

In chapter II, we do a background study of our thesis. We discuss the basic concepts of deep learning and evolutionary computation.

In Chapter III, we explain the literature review in the field of neuroevolution of Generative Adversarial Networks.

In chapter IV, we introduce our proposed approach in detail with algorithms to understand it better.

Chapter V describes the experimental setups and detailed results of the proposed methods.
In Chapter VI, we compare our work with other benchmark models and analyze the results.

Chapter VII concludes the research, discuss limitations and set up potential directions for future work.

CHAPTER 2

Introduction to Deep Learning and Evolutionary Computation

This chapter introduces the reader to deep learning. The following section introduces the fundamental concepts of Artificial Neural Networks. Section 2.2 Convolutional Neural Networks describes the detailed explanation of components of Convolutional Networks. Section 2.3 Generative Adversarial Networks provides an introduction to GANs and equips the reader with the necessary knowledge for the methods implemented in this thesis. An introduction to evolutionary algorithms is presented in section 2.4 , which is the fundamental technique used throughout this thesis.

2.1 Artificial Neural Networks

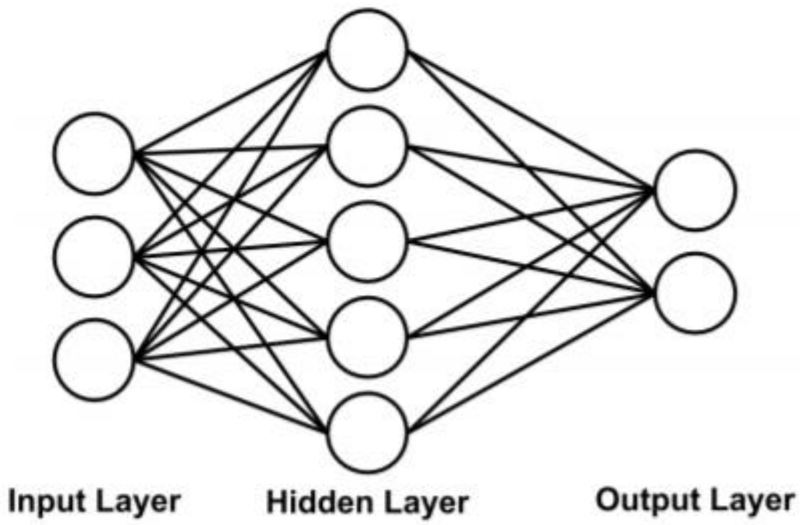


Figure 4 General Architecture of Neural Network

Artificial Neural Networks (ANN) are machine learning tools which are loosely based on human mind architecture [20]. ANN is one of the most active topics of research in machine learning, and it is because ANN has the capability to represent and learn highly complex and non-linear functions.

The most simple ANN contains three layers and is composed of an input layer, a hidden layer and an output layer, where each layer contains neurons [21]. Figure 4 shows a general ANN architecture which contains 3 input neurons, 4 hidden neurons and 2 output neurons.

2.1.1 Perceptron

Frank Rosenblatt developed the first neuron, which he named as perceptron in 1957 [22]. The basic unit of ANN is the perceptron (neuron). The perceptron works in the following way: All the inputs x are multiplied with their weights w. Let's call it k.

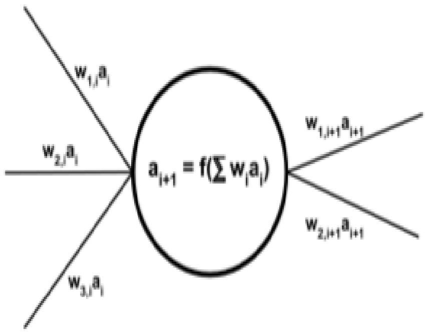


Figure 5 A sample perceptron [21]

Add all the multiplied values and call them a weighted sum. Apply the weighted sum to the Activation Function. The perceptron will return 1 only if the aggregated sum is more than some threshold else returns 0. A single perceptron can only be used to implement linearly separable functions.

2.1.2 Activation Functions

Activation functions are used to propagate the output of one-layer perceptron to another layer perceptron. Activation functions are scalar-to-scalar function, deciding the activation of perceptron in Neural Network. To introduce nonlinearity for the neural network, hidden layers uses activation function. Most of the important activation function belongs to a logistic class of transform when graphed resembles an S. For this section, we will state useful activation function in neural networks.

2.1.2.1 Linear Function

A linear function is the identity function represented by $f(x) = Wx$, where the dependent variable has a direct proportional relation relationship with the independent variable [23]. In practical definition, it means the function passes through signal unchanged.

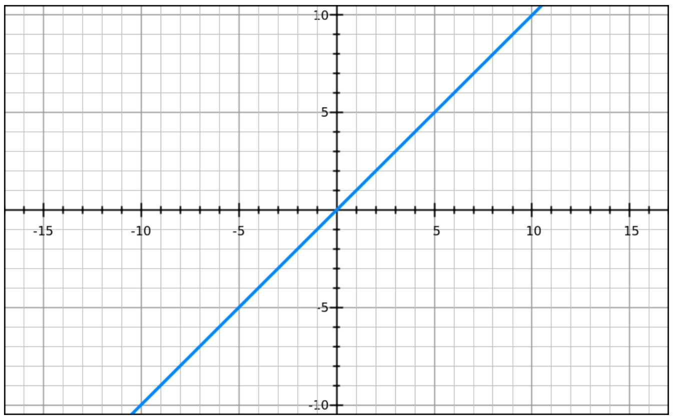


Figure 6 Linear activation function

Linear activation functions are commonly used in the input layer of the neural network.

2.1.2.2 Sigmoid Function

A sigmoid activation function [24] outputs an independent probability for each class. A sigmoid function will convert independent variables of infinite range into simple probabilities between 0 and 1, and most of its output will be very close to 0 and 1. The vertical line in Figure 7 is the decision boundary.

$$S(x) = \frac{e^x}{e^x+1} \tag{3}$$

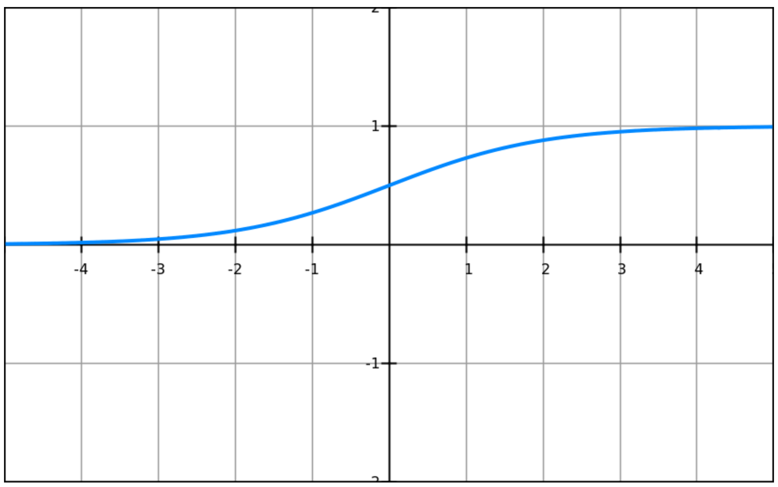


Figure 7 Sigmoid activation function

2.1.2.3 Tanh Function

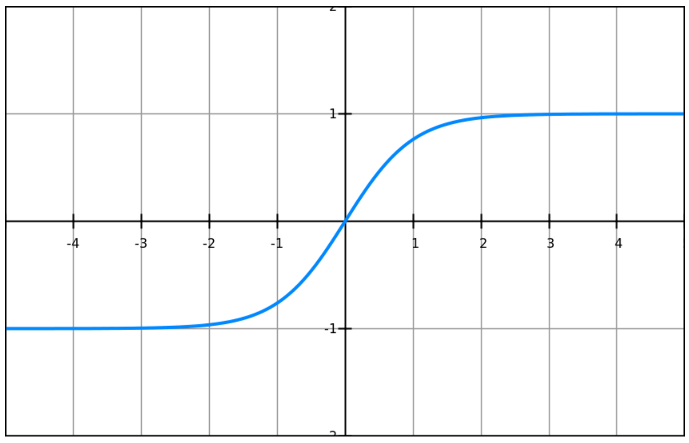


Figure 8 Tanh activation function

Tanh [25] represents the ratio of hyperbolic sine to the hyperbolic cosine. Tanh is normalized in the range of -1 to 1. The advantage of Tanh over the sigmoid function is that it can deal more efficiently with negative numbers.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} \tag{4}$$

2.1.2.4 Rectified Linear Function

Rectified Linear (ReLU) [26] activates a neuron if the input is above a certain value. When the input is above a certain quantity, it has a linear relationship with the dependent variable $f(x) = \max(0, x)$ as shown in Figure 9. ReLU is the current state of the art. Because the gradient of ReLU is either zero or a constant, it is possible to prevail in vanishing exploding gradient issue.

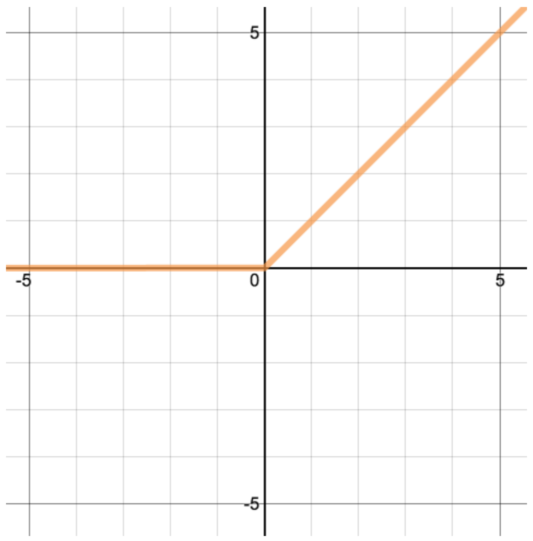


Figure 9 Rectified Linear activation function

2.1.2.5 Leaky ReLU Function

Leaky RELU [26] has a small slope of negative values instead of altogether zero. The downside for being zero for all the negative values is called a dying ReLU problem. Leaky ReLU is strategy to mitigate dying ReLU problem. Leaky ReLU results are not always consistent.

$$f(x) = \begin{cases} x & if\ x > 0 \\ 0.01x & otherwise \end{cases} \tag{5}$$

2.1.3 Loss Functions

Loss Functions calibrate how close the neural network is to the ideal towards which it is training. A metric is calculated based on the error observed in the network's predictions. We then aggregate these errors over the entire dataset and average them to find a single number representation of how close the network is to its ideal.

The weight and bias of the neural networks decide the output of the network and altering them alters the loss function. Looking for the ideal state is same as finding weight and bias of the network, which will minimize the loss function incurred from the errors. Finding these parameters cannot be solved analytically but can be found by iterative optimization algorithms like gradient descent.

The loss function notation can be described as follows:

$$h_{w,b}(X) = \hat{Y} \tag{6}$$

Where,
W, b: weight and bias of the network respectively
X: represents Input data
$\hat{Y}$: denote the output of the neural net
$h(X_i) = \hat{Y}_i$: denote the neural network transforming the given input to the output $\widehat{Y_\iota}$

2.1.3.1 Mean squared error loss

The error in a prediction is squared and is averaged over the number of data points. The MSE loss [27] can be described as follows:

$$L(W,b) = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y} - Y_i)^2 \tag{7}$$

The loss function boils-down the difference between desired and predicted, be that they are vectors, into a single number.

2.1.3.2 Hinge loss

When the network is to be optimized for hard classification like 0-1 classifier, hinge loss is most commonly used. For example, 0= no fraud and 1= fraud. The 0,1 choice is arbitrary and -1,1 is also seen in substitute of 0,1.
The hinge loss equation [23] can be described as follows:

$$L(W,b) = \frac{1}{N}\sum_{i=1}^{N} max(0{,}1 - Y_{i,j*}\hat{Y}_{i,j}) \tag{8}$$

2.1.3.3 Maximum likelihood loss

When probabilities are of great interest than hard classification, logistic loss is used. Example, probability of someone clicking on an advertisement. In maximum likelihood loss [23], we want to maximize the probability to predict the correct class, and we want to do so for each sample in the dataset.
We can describe the loss function for 0,1 classifiers as follows:

$$P(y_i|X_i;W,b) = (h_{W,b}(X_i))^{y_i} * \ (1 - h_{W,b}(X_i))^{1-y_i} \tag{9}$$

The above equation can be written as follows for each sample:

$$L(W,b) = \prod_{i=1}^{n} \hat{y_i}^{y_i} * (1-\hat{y_i})^{1-y_i} \tag{10}$$

2.1.3.4 Negative log-likelihood loss

For mathematical convenience, when dealing with the product of probabilities, it is accepted to convert them to the log of the probabilities. The product of the maximum likelihood transforms into the sum of the log of the probabilities. The logarithm is monotonically increasing function. Thus, minimizing the negative log-likelihood is equivalent to maximizing the probability.
The negative log-likelihood [20] can be written as follows:

$$L(W,b) = -\sum_{i=1}^{N} Y_i * \log \hat{y_i} + (1-Y_i) * \log(1-\hat{y_i}) \tag{11}$$

When the loss function is extended from two classes to M classes, it gives us the equation which is called as cross-entropy between two probability distributions.

2.1.4 Gradient Descent

Gradient descent is the first-order iterative optimization algorithm for finding the minimum of a function. It is represented as a vector on n partial derivatives of the function f. Gradient descent calculates the slope of the loss function by taking a derivative. On a two-dimensional loss function, the derivative would simply be the tangent of any point on the parabola, i.e. change in y over change in x.

The gradient points directly uphill, as shown in Figure 10, so a parameter is updated by taking a small step in the reversed direction of the gradient, this small step is known as **learning rate**. The size of the learning rate is difficult to set; it must be large enough to make progress but small enough to not miss the minimum.

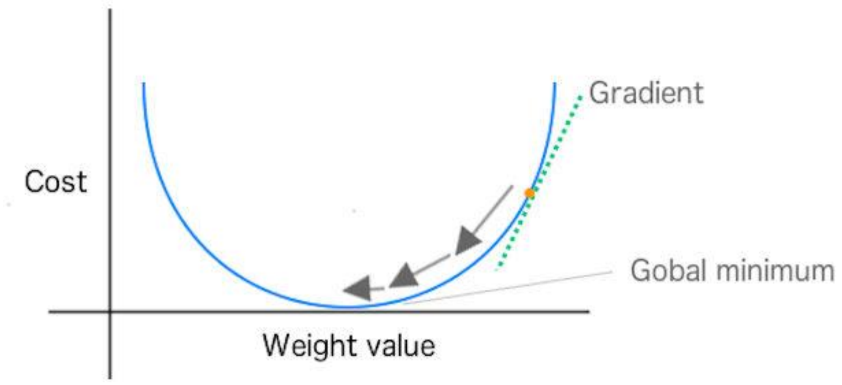


Figure 10 Showing Weight change using Gradient Descent

Gradient descent can be algebraically written as [28]:

$$\theta' = \theta - \eta\nabla f(\theta) \tag{12}$$

Where,

θ' : is the newly updated weight parameter

θ : is the old parameter

η : is the learning rate

$\nabla f(\theta)$: is the gradient of the loss function

The process of calculating the gradient of the loss function with respect to the network's parameter is usually made by the back-propagation algorithm.

2.2 Convolutional Neural Networks

Multi-Layer Neural Networks does not scale well with the image data. When the image data is parsed into the feed-forward simple neural networks, the learnable parameters increases to approximately 1 billion for 1024x1024 pixel size image. Even though we have computers to handle computation on this scale, it is very time-consuming. The structure of image data allows to change the architecture of a neural network in a way that we can take advantage of this structure, the goal of the CNN is to learn higher-order features in the data via convolutions. The efficacy of CNNs in image recognition is one of the main reasons why the world recognizes the power of deep learning. CNN architecture can be considered to be three-dimensional volume of neurons.

2.2.1 CNN Architecture Overview

CNN's [20] transforms the input data from the input layer through all connected layers into the set of class scores given by the output layer. High-level CNN architecture view is shown in Figure 11.

It consists of three parts:

1) Input layer
2) Feature-extraction layer
3) Classification layer

The input layer accepts three-dimensional input in the form of height*width*RGB colour channels. The feature extraction layer has a general repeating pattern of Convolutional layers and Pooling layers.

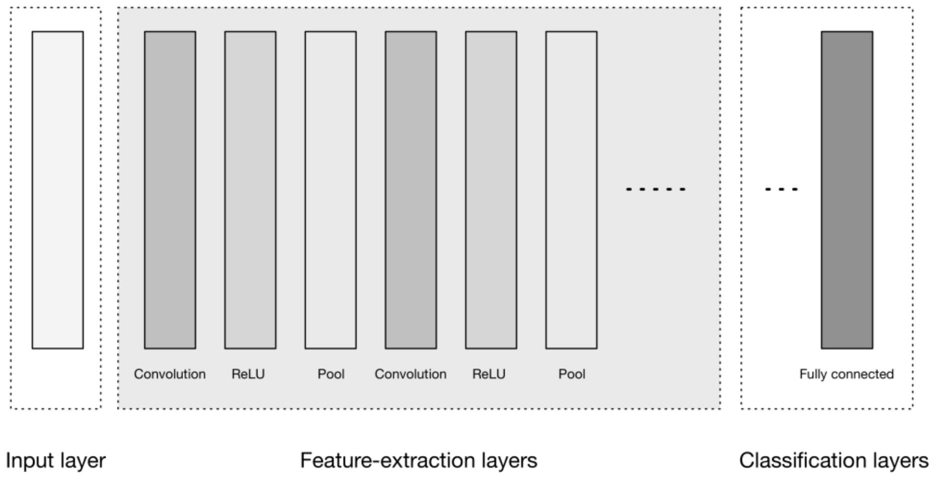


Figure 11 High-level general CNN architecture

2.2.2 Convolutional Layers

The Convolutional layers are the core building blocks of CNN architectures [23]. A convolution is how the input is modified by a filter. A filter is taken to slice through the image and map it to learn different portion of input image. As shown in Figure 12, Dot product is computed between the kernel and the layer of the image. The resulting output generally has the same spatial dimensions but increases the number of elements in the third dimension of the output.

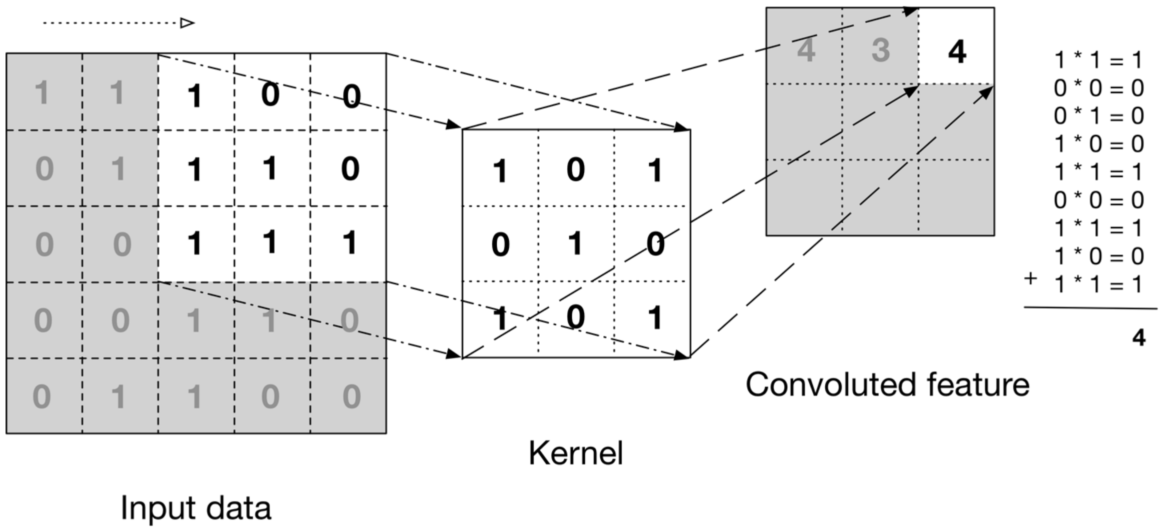


Figure 12 Sample Convolution operation [23]

2.2.3 Filters/Kernels

Filters are the function that has a width and height smaller than the input volume. Filters are applied across the width and height of the input volume in a sliding window manner, as shown in Figure 12. The output of the filter is computed doing the dot product of the filter and the input region. The filters also usually have a shared bias parameter for each convolutional layer. The output activation map is expressed as

$$a_{j,k}^{l} = \sigma(b^{l-1} + \sum_{m=0}^{p}\sum_{n=0}^{q} w_{m,n}^{l-1} a_{j+m,k+n}^{l-1}) \tag{13}$$

Where $a_{j,k}^{l}$ is the activation value of the kth neuron in the jth row of lth layer, b is the shared bias, w is the weight parameter of pxq kernel and $\sigma(z)$ is the activation function.

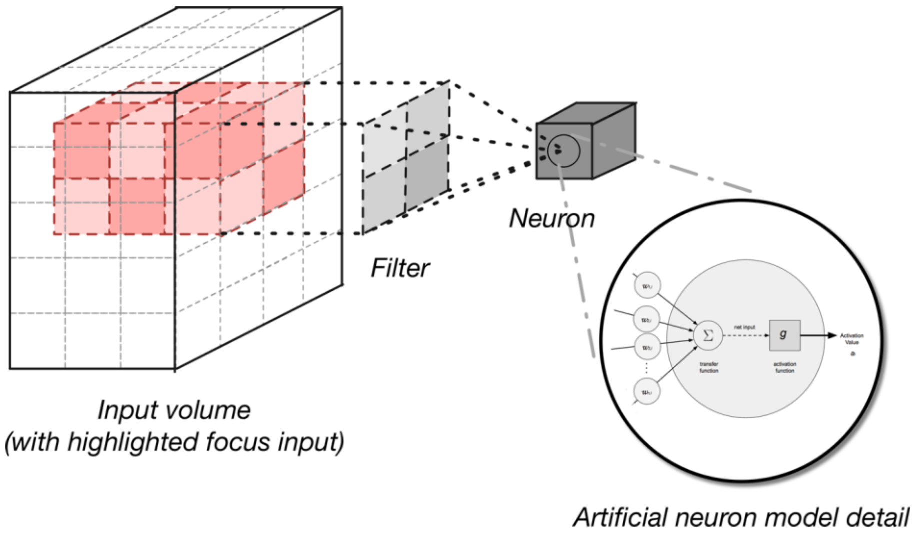

Neuron
Filter
Input volume
(with highlighted focus input)
Artificial neuron model detail

Figure 13 Generating an activation output volume

The above activation value calculation can be visualized, as shown in Figure 13.

2.2.4 Pooling Layers

Pooling layers are commonly followed by convolutional layers. Pooling layers are used to progressively reduce the spatial size of the data representation. They replace the output by taking the summary statistics of the nearby output values. Most common downsampling operation is the max operation, also called as max pooling. Max pooling reduces the image size by mapping the mxn window into a single result by taking the maximum value of the elements in the window, as shown in Figure 14.

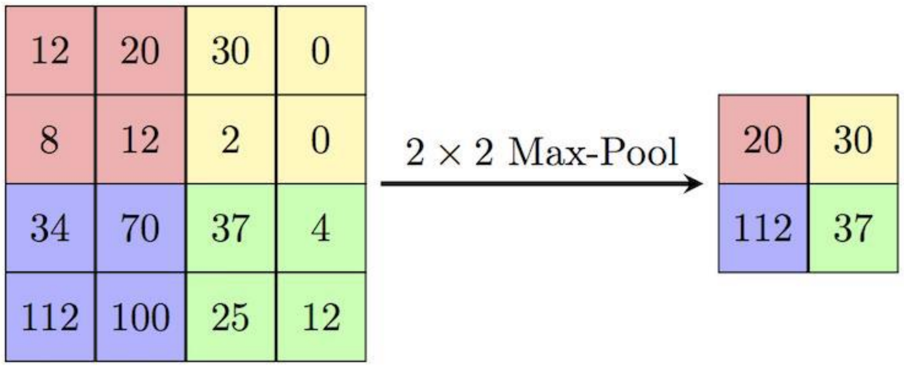


Figure 14 Max pooling

Convolution is powerful in detecting the same patters as horizontal edge or vertical edge across the image. Hence CNNs are well adapted to translation invariance of images.

2.3 Generative Adversarial Networks

One of the most impressive successes in deep learning has, so far, involved discriminative models, i.e. models that map the dependence of unobserved target variables y on observed variables x.

Discriminative models infer outputs based on inputs without caring about how the input was generated. Generative models, as opposed to discriminative models, maps how the input data was generated. They are a branch of unsupervised learning techniques [1].

Generative Adversarial Networks (GANs) [1] are a class of generative models. GANs are trained to generate fake data similar to some known input data. A GAN model consists of two types of neural networks, a generative model and a discriminative model. The two networks compete against each other, and they have an adversarial relationship. The generative model learns to generate data while the discriminative model learns to predict whether a data is from the model distribution p_{model} or the original data distribution p_{data}. During training, both models improve their methods until the artificially generated data are indistinguishable from real data [29]

2.3.1 Structure of a GAN

A GAN is made up of two different networks called as a Discriminator network and Generator network.

Generator Network- The generative network in GANs creates synthetic data with a special kind of layer called a transpose convolutional layer. The input to the generator z is sampled from some simple prior distribution. The generator can be seen as a kind of reverse CNN. It takes a vector of z-dimensional noise as input and upsamples it to images [1].

Discriminator Network- When modelling images, the discriminator network is generally a standard CNN. Using a secondary network as the discriminator neural network allows the GAN to train two networks in parallel in an unsupervised fashion. The input to the discriminator networks is images and outputs classification probabilities.
The gradient of the discriminator network output with respect to generated input data indicates how to make changes to the generated data to make it more realistic [23].

2.3.2 Training GANs

In the training process of GANs, both the generator and discriminator are trained simultaneously. Two mini-batches are sampled in the first step. One of the batches is z values from p_{model}, the model's prior over latent variables. In the next step for each network, a gradient step is made. For generator network, the gradients update the parameters $\theta^{(G)}$ to reduce its loss function $L^{(G)}\left(\theta^{(G)}, \theta^{(D)}\right)$ and one for updating discriminator's parameters $\theta^{(D)}$ to reduce discriminators loss function $L^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right)$. The tricky part here is that there are two optimizer function used one for each network. In other words, discriminator and generator play the two-player minimax game with value function $V(G, D)$ as shown in Equation 14 [1]. This training process is repeated for a number of training iterations.

$$minG\ maxD\ \mathrm{V(D,G)} = \mathbb{E}_{x\sim p_{data(x)}}[\log \mathrm{D(x)}] + \mathbb{E}_{z\sim p_{z(z)}}[\log (1 - D(G(z)))] \qquad (14)$$

GANs training is complicated; there must be a balance during training between the two networks. Otherwise, one can overpower the other network. It is therefore essential to have the correct hyperparameters, network topography and training procedure.

When the generator overpowers some weakness in a discriminator, then an issue called **mode collapse** occurs. In mode collapse, the generator produces very similar images regardless of a change in the input z. In another case, the discriminator can also overpower generator where the discriminator classifies fake generated data with absolute certainty. In this case, generator is left with no gradients and the network will not learn anything, this issue is called a **vanishing gradient**. These issues can be avoided by accurate hyperparameters of both the networks [29].

During the training process, the discriminator gets one of the two different inputs. The first is when x is real data. In this case, the discriminator $D(x)$ goal is to be near to 1. In the second scenario, both the generator and the discriminator participate. The discriminator receives generated data, i.e. $\mathrm{x} = \mathrm{G}(\mathrm{z})$, where the discriminator goal is to make $D(G(z))$ closer to 0 and the generator will try to make it near 1. This is how the generator learns to generate synthetic data. Finally, if the training is balanced enough then at the end of the training, they will achieve Nash equilibrium. When this is achieved the $D(x)$ for both the input will be equal to 0.5, and it will be valid for any x, i.e. $P_{model} = P_{data}$ [20] [29].

2.3.3 Deep Convolutional GAN

Deep Convolutional Generative Adversarial Networks (DCGAN) [30] is a special class of GANs which is heavily inspired by CNN. Most of the real-world image generation applications, starter approach is DCGAN [31]. Compared to regular GAN approaches DCGAN has more stable training architecture.

In DCGAN, the overall network architecture is composed of all convolutional layers. In discriminator, the pooling layers are replaced with transposed convolutions, and in a generator, it is replaced with strided convolutional layers. Such architecture design allowed the generator to learn its own upsampling. While in discriminator in the last layer, the transpose convolution was flattened and fed into sigmoid activation function. All the fully connected layers were also eliminated in DCGAN [30].

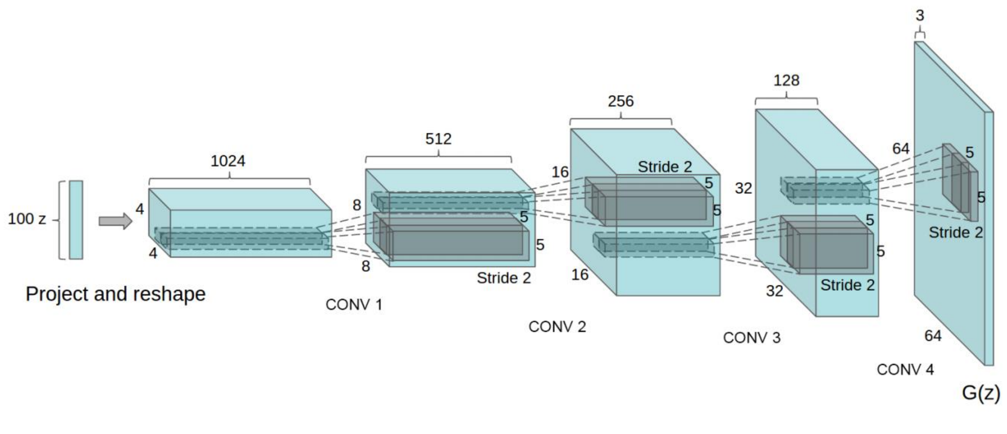


*Figure 15 An example of a generator network in DCGAN. A 100-dimensional noise z is passed into the transpose convolutional layers which are converted into 64*64 pixel image [30]*

Another significant change in DCGAN was to introduce the usage of batch normalization layer [31]. Because of batch normalization, each neuron had zero mean and unit variance, which helped to stabilize the training. This also allowed gradients to flow deeper and prevented the generator collapse. To avoid sample oscillation and instability, the generator output layer and discriminator input layer were not batch normalized [30].
The final change was the generator and discriminator activation functions were different. The generator had ReLU in all layers except for output, and the discriminator had LeakyReLU.

2.3.4 GAN Loss Functions

Like DCGAN, many other studies have tried to improve the training of the regular GANs. One of the ways is to improve the training using different variations of loss functions. Because the gradient used to train the network is calculated using the loss function, they play an essential role in training stability.

In GAN, the discriminator can be defined in term of the non-transformed layer $C(x)$, as $D(x) = sigmoid(C(x))$. Here $C(x)$ can also be called as the critic [32]. $C(x)$ can be interpreted as how real the data is, while a negative number means the data is synthetic. Also, let the real and fake data samples be represented as x_r and x_f respectively.

Using these representations, we will define five different GAN loss functions which are used in this thesis.

1. Vanilla GAN [1]

$$L_D^{VGAN} = -\mathbb{E}_{x_{r\sim\mathbb{P}}}[log\left(Sigmoid(C(x_r))\right) - \mathbb{E}_{x_{f\sim\mathbb{Q}}}[\log(1 - sigmoid\left(C(x_f)\right))] \tag{15}$$

$$L_G^{VGAN} = -\mathbb{E}_{x_{f\sim\mathbb{Q}}}[\log(sigmoid\left(C(x_f)\right))] \tag{16}$$

2. LSGAN [33]

$$L_D^{LSGAN} = \mathbb{E}_{x_r \sim \mathbb{p}}[(C(x_r) - 0)^2] + \mathbb{E}_{x_f \sim \mathbb{Q}}[(C(x_r) - 1)^2] \quad (17)$$

$$L_G^{LSGAN} = \mathbb{E}_{x_f \sim \mathbb{Q}}[(C(x_f) - 0)^2] \quad (18)$$

3. HINGE GAN [34]

$$L_D^{HingeGAN} = \mathbb{E}_{x_r \sim \mathbb{p}}[\max(0, 1 - C(x_r))] + \mathbb{E}_{x_f \sim \mathbb{Q}}[\max(0, 1 + C(x_f)] \quad (19)$$

$$L_D^{HingeGAN} = -\mathbb{E}_{x_f \sim \mathbb{Q}}[C(x_f)] \quad (20)$$

4. RSGAN [32]

$$L_D^{RSGAN} L_D^{RSGAN} = -\mathbb{E}_{(x_r, x_f) \sim (\mathbb{p}, \mathbb{Q})}[\log(sigmoid(C(x_r) - C(x_f)))] \quad (21)$$

$$L_G^{RSGAN} = -\mathbb{E}_{(x_r, x_f) \sim (\mathbb{p}, \mathbb{Q})}[\log(sigmoid(C(x_f) - C(x_r)))] \quad (22)$$

5. RaSGAN [32]

$$L_D^{RaSGAN} = -\mathbb{E}_{x_r \sim \mathbb{p}}[\log \widetilde{D}(x_r)] - \mathbb{E}_{x_f \sim \mathbb{Q}}[\log(1 - \widetilde{D}(x_f))] \quad (23)$$

$$L_G^{RaSGAN} = -\mathbb{E}_{x_f \sim \mathbb{Q}}[\log \widetilde{D}(x_f)] - \mathbb{E}_{x_r \sim \mathbb{P}}[\log(1 - \widetilde{D}(x_r))] \quad (24)$$

Where,

$\widetilde{D}(x_r) = sigmoid(C(x_r) - \mathbb{E}_{x_f \sim \mathbb{Q}} C(x_f))$

$\widetilde{D}(x_f) = sigmoid(C(x_f) - \mathbb{E}_{x_f \sim \mathbb{p}} C(x_r))$

2.4 Evolutionary Algorithms

An evolutionary algorithm (EA) is a technique inspired by biological evolution that aims to mimic the same evolutionary mechanism found in nature. In EAs, the population is composed of individuals that represent possible solutions for a given problem, using a high-order abstraction to encode their characteristics [27]. The algorithm uses various genetic operators like mutation, crossover and selection on the population in order to search for better solutions. Evolutionary algorithms have various advantages which have made them immensely popular.

- EAs are faster and more efficient when compared to traditional methods
- Has excellent parallel computing capabilities
- Optimizes both the continuous, discrete and also solve multi-objective optimization problems
- EAs are useful when the search space is substantially large, and there are a large number of parameters involved

Like any other technique, EAs also suffer from a few limitations.

- EAs are stochastic, because of which there might not be any guarantee on the optimality of the solution
- Fitness is calculated at every generation, which might be computationally expensive for a few problem domains like ours

2.4.1 Genetic Algorithms

Genetic Algorithms (GA) are the first EAs model developed to simulate natural genetic evolution. John Holland is considered the father of GAs [35]. GAs are search heuristics inspired by Charles Darwin's theory of natural evolution - "Survival of the fittest."

In GAs, we have a pool or population of possible solutions for the given problem. These pools or population undergo genetic operation like crossover and mutation, producing new offsprings. Genetic algorithms reflect the process of natural selection, where the fittest individuals are elected to reproduce the offspring of the upcoming generation. Each candidate individual is assigned a fitness value based on its objective function. The more fit the individuals in the population, the more chance they have to mate and produce more "fitter" individuals. We carry out this evolution procedure for multiple generations until we reach stopping criteria.

It is imperative to be familiar with some basic terminology which will be useful throughout the thesis.

- **Population -** It is a subset of all possible candidate solutions to the given problem.
- **Chromosomes -** A chromosome is one such candidate solution to the given problem
- **Gene -** It is one of the elements of a chromosome
- **Allele -** An allele is a value which a gene takes for a particular chromosome
- **Genotype -** A genotype is an individual of the population in the computation space. The individual's representation in computational space is in such a way that it can be easily understood and manipulated using a computation system
- **Phenotype -** Phenotype is the population space in the actual real-world solution domain. In phenotype, the solutions are represented in the way they exist in real-world situations.

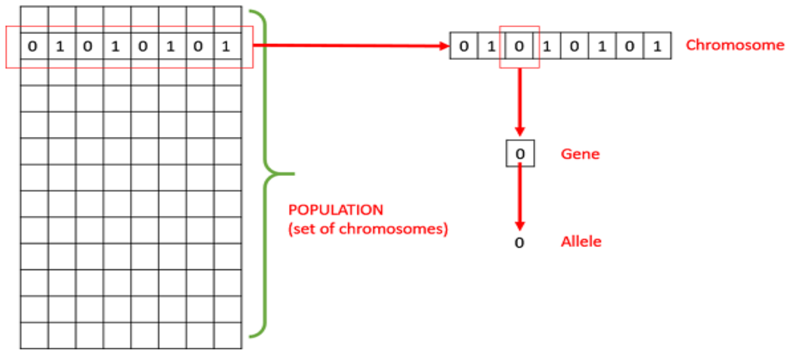


Figure 16 Representing Population space, Chromosome, Gene and Allele [36]

- **Encoding and Decoding –** In most of the problems, genotype and phenotype are different. As shown in Figure 17, Encoding is a process of transforming a solution from phenotype to genotype while decoding is the other way around, transforming a genotype to phenotype.

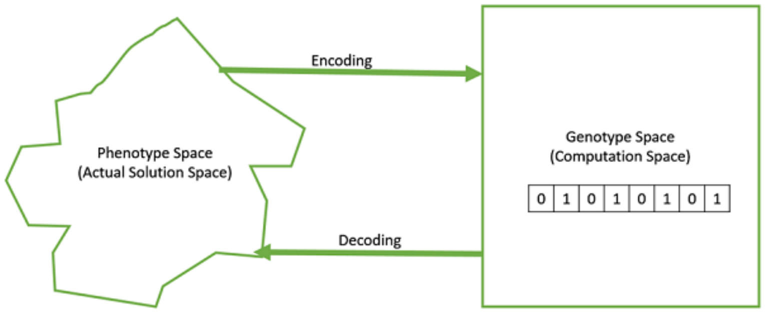


Figure 17 Encoding and Decoding [36]

- **Fitness Function** – A fitness function takes a genotype as input, and outputs suitability of the solution. In our case, the fitness function and the objective function are the same, while in some other cases it can be different based on the problem

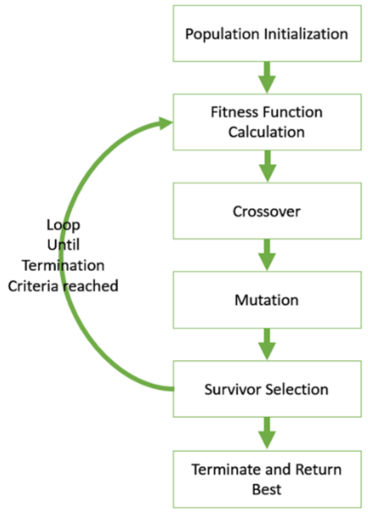

Population Initialization
Fitness Function Calculation
Crossover
Mutation
Survivor Selection
Terminate and Return Best
Loop Until Termination Criteria reached

Figure 18 Genetic Algorithm [36]

In GAs, we start with an initial population space which may be seeded by some heuristic or generated randomly. In the next step, we select the parents from this population space for yielding offsprings. Then with the elected parents, we do genetic operations like crossover and mutations to generate new offsprings. The better offsprings replace the individuals in the population and the evolutionary process repeats. A basic flow of GAs is shown in Figure 18.

2.4.1.1 Crossover Operation

The crossover genetic operator is similar to the reproduction and biological genes crossover. From the population, more than one parent is selected, and one or more new individuals are produced using the genes of the parent.

There are many different types of genetic crossovers like the one-point crossover, multi-point crossover and uniform crossover. For this thesis, we will only use the one-point crossover. It is also necessary to note that GA designer, might choose to implement a problem-oriented crossover.

One Point Crossover

Figure 19 shows a one-point crossover operation. A random or specific point is selected in the parent individuals, and the tails of its two parents are exchanged to generate a new offspring.

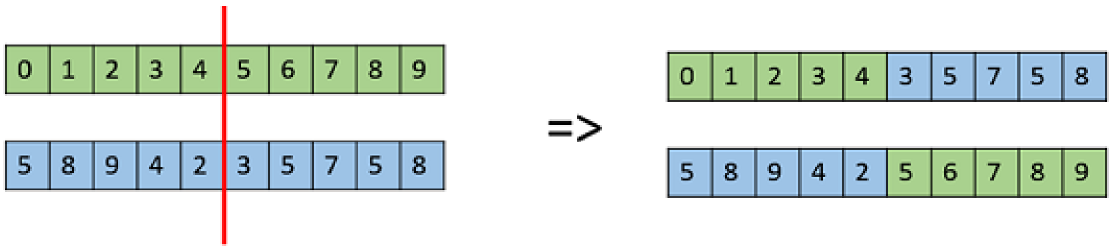


Figure 19 One Point Crossover

2.4.1.2 Mutation Operation

Mutation operation is a random tweak in the chromosome, to generate new offspring. The primary role of mutation is to introduce diversity in the population. For the converge of evolutionary algorithms, the mutation is an essential part. Many common types of mutation exist like bit flip, random resetting, swap mutation, scramble mutation etc. [36]. For this thesis, we have implemented the random resetting mutation.

Random Resetting

In random resetting mutation, a random value from the list of permissible allele is assigned to a gene which is uniformly randomly chosen.

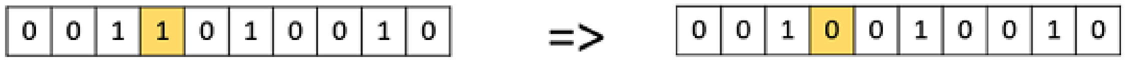


Figure 20 Random Mutation

2.4.1.3 Selection Operation

The selection operation decides which chromosome from the population and offspring set will proceed to the next generation, and which will be eliminated. It is a vital operation as it should be able to promote fitter chromosomes, while the diversity of the population space should also be maintained.

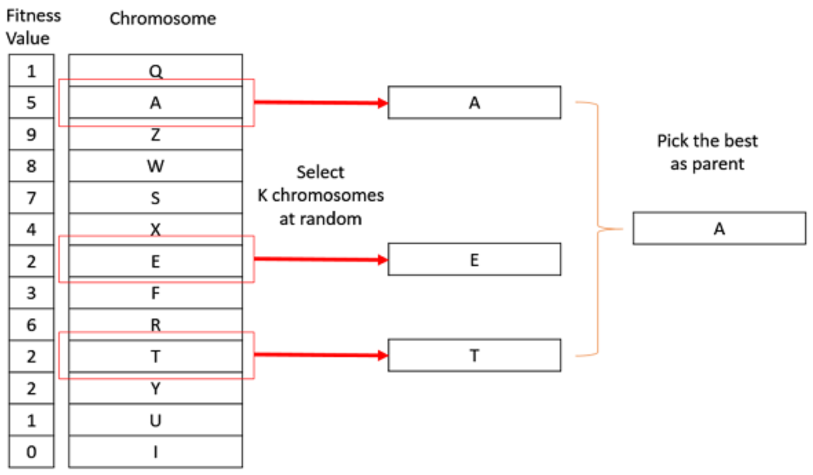


Figure 21 Tournament Selection [36]

For this thesis, we have used a tournament selection [37] operation. In a k-way tournament, k random chromosomes are selected, and we run a tournament among them. The fittest individual is elected among them and is passed on the next generation. In a similar way, many such tournaments can take place until we have our final selection of the candidates for the next generation. The k-way tournament selection is shown in Figure 21.

2.4.2 Cultural Algorithms

Cultural evolution(CE) was first introduced by Renyolds [38] in the early 1990s. CE bias the search process with prior knowledge about the domain as well as knowledge gained during the evolutionary process.

Standard EAs like genetic algorithms are unbiased using little or no domain knowledge to guide the search process. However, the performance of EAs can be improved considerably if the domain knowledge is used to bias the search process. Domain knowledge then serves as a mechanism to reduce the search space by pruning unwanted parts of the solution space by promoting desirable parts.

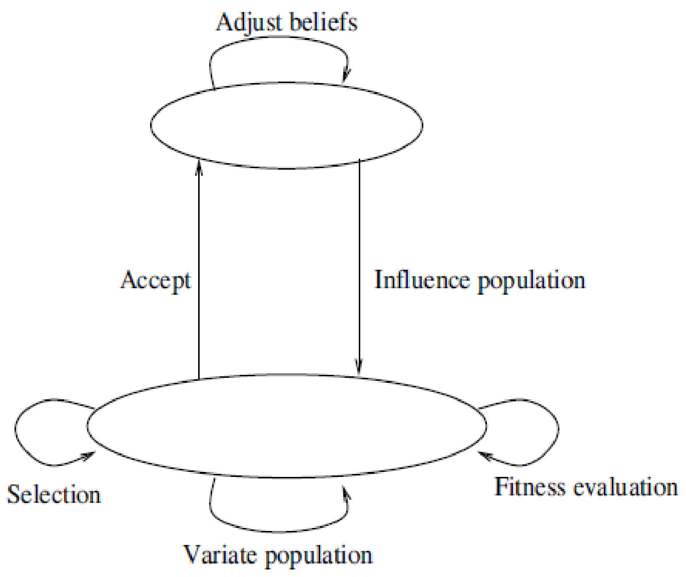


Figure 22 Population Space and Belief Space in CA [38]

A Cultural Algorithm (CA) maintains two search spaces: the population space and a belief space. The population space contains the individuals, and the belief space contains cultural knowledge. Both the population space and belief space evolve in parallel to optimize some function. Figure 22 visualizes the dual-inheritance system of CA. A two-way communication protocol is set up to exchange information between the search spaces. One protocol called as **acceptance** function used to select a group of individuals

from the current population, these selected individuals will be used to adjust the current belief space and second **variation** operator uses the beliefs to control the changes in the individuals [27].
The next section discusses the belief space in more detail.

2.4.2.1 Belief Space

The belief space serves as a knowledge repository, where the collective behaviour of the individual in the population space are stored [38]. The belief space can effectively be used to prune the population search space: the knowledge within the belief space is used to route individuals away from undesirable areas in the population space towards more promising areas. It has been proven that the use of a belief space reduces computational cost dramatically [39].

To represent the behavioural pattern of individuals from the population space, belief space contains a number of knowledge components. The type of knowledge component and data structures used to represent the knowledge depends on the problem being solved.
In general, the belief space contains five different knowledge components [38]:

1. A **situational** knowledge component keeps track of the elite solutions found at each generation.
2. A **normative** knowledge component provides specifications for individuals behaviour, used as guidelines for mutational adjustment to individuals.
3. A **domain** knowledge component contains an archive of best solutions since evolution started. Domain knowledge is not re-initialized at each generation.
4. A **history** knowledge component maintains information about the sequence of environmental changes. For each environmental change, the following information is stored: the best solution, the change in direction for each dimension and the current change distance.
5. A **topographical** knowledge component which maintains a multi-dimensional grid representation of the search space.

CHAPTER 3
Related Work

In this chapter, we have reviewed the literature of evolutionary Generative Adversarial Networks.

Modern machine learning techniques focus heavily on the usage of deep learning approaches. In deep learning, the neural network weights are trained through a variation of stochastic gradient descent. An alternative way comes from the field of Neuroevolution. Neuroevolution is the application of evolutionary algorithms in the evolution of neural networks. Neuroevolutionary approach can be applied to weights, topography and hyperparameters of the neural networks. Neuroevolution can be used for the generation of network architecture, and a substantial benefit is the automation of topography design and the parameters of the network [40].

However, evolutionary algorithms are not the only approach for Neuroevolution. Neuroevolution can be combined with a Reinforcement learning approach, also called as AutoML [41]. But the problem with such AutoML approach is that it requires vast computational resources in spite of their success, in many real-world applications it makes unfeasible to apply [16]. Therefore, we have focused our thesis on classic evolutionary approaches.

GANs were first introduced in 2014 [1] by Ian Goodfellow. Several works improving the GAN model were recently published, leveraging the quality of the results to impressive levels [10] [11] [29]. However, there are some open problems related to training of GANs like mode collapse and vanishing gradient [42]. Neuroevolution of GANs can be helpful to resolve the existing problems in GANs [43]. Therefore, this literature focuses on a niche area of the evolution of generative adversarial networks. Moreover, the first evolutionary GANs were introduced in 2018 by Wang et al., so this research area is very recent. We have tried to cover all the evolutionary GAN approaches until August 2019. Related work is divided into three sections, where the first section contains methods of neuroevolution of weight parameter of the generator network. In the second section, only the discriminator and generator architecture is evolved. And in the last section study of a combination of weight parameter, topography and hyperparameter of the GANs are presented.

3.1 Neuroevolution of Weights

Various evolutionary algorithms like genetic algorithm [35] and coevolutionary algorithm [27] are used to evolve the weight parameters of the generator and discriminator networks.

In March 2018, Wang et al. proposed the first evolutionary generative adversarial networks (E-GAN) [9] approach. E-GAN was proposed with the intention to improve the training stability and better generative performance of GANs. Figure 23 shows the E-GAN architecture where a population of generator G_θ evolves in a dynamic environment, the discriminator D.

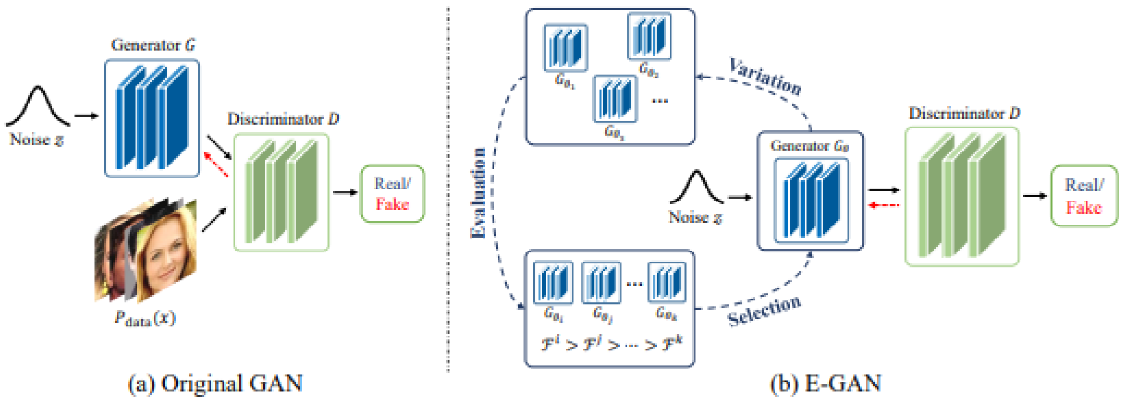


Figure 23 Conventional GAN versus E-GAN [9]

Unlike in conventional GAN, E-GAN uses pre-defined objective functions, alternatively training generator's weight parameters. In each evolutionary step, there are three sub-stages: variation, evaluation and selection. One of the main contributions of E-GAN is the variation step, in which asexual reproduction with different mutations is used to produce the next generation's individuals. E-GAN was tested on datasets like LSUN bedroom [44] and CelebA [45]. In E-GAN the generator quality was compared with conventional GANs using FID score and demonstrated that E-GAN achieves convincing generative performance and minimizes training problems in conventional GANs.

Thereafter, in August 2018, Abdullah et al. proposed a spatial coevolutionary approach called Towards Distributed Coevolutionary GANs (Lipizzaner) [46]. In which, the researchers investigate the usage of coevolutionary algorithms with conventional GAN training. Their aim was to bridge the gap between works of deep learning and evolutionary computing communities towards a better understanding of gradient-based and gradient-free GAN dynamics.

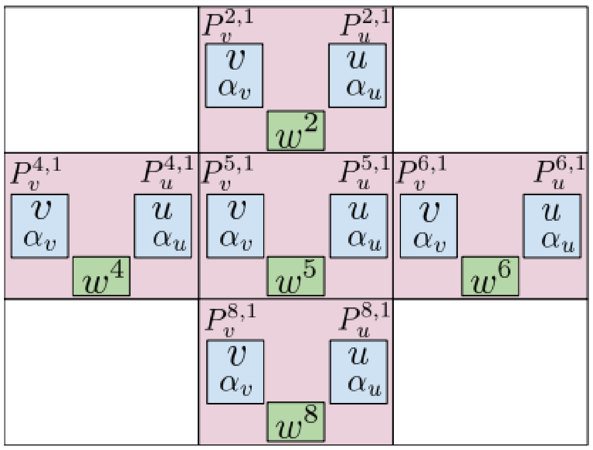


Figure 24 Spatial coevolution of generator and discriminator population [46]

Figure 24 represents a spatial GAN training framework that allows scaling over a distributed spatial grid topology. In spatial coevolution, GAN individuals are distributed on a grid, as shown in Figure 24, where the local interaction of individuals governs the fitness evaluation, selection and mutation. Lipizzaner framework was tested on MNIST

[17] and CelebA datasets. Researcher in their experiments shows that coevolution is a promising framework for escaping degenerate GAN training behaviour.

An improved version of Lipizzaner was introduced by Jamal et al. in July 2019. This approach is called as spatial evolutionary generative adversarial networks (Mustangs) [47].

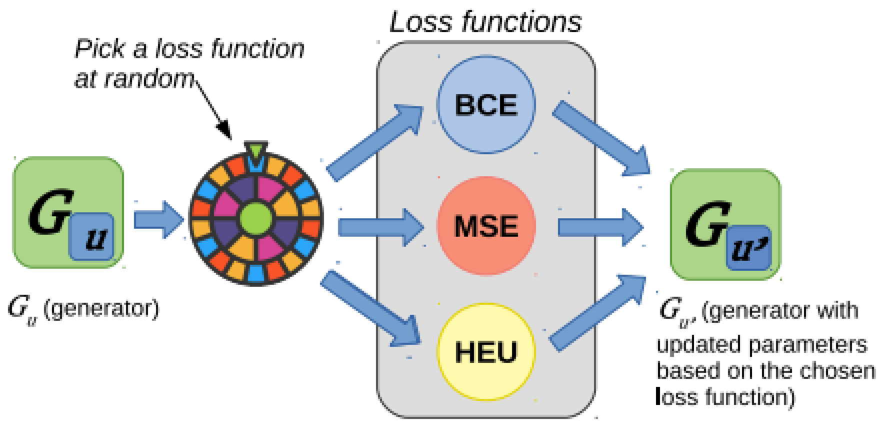

Pick a loss function at random
Loss functions
BCE
MSE
HEU
G_u
G_u (generator)
$G_{u'}$
$G_{u'}$ (generator with updated parameters based on the chosen loss function)

Figure 25 Graphical representation of the mutation used in Mustangs [47]

In mustangs, the main idea focuses on combining mutation from E-GAN and population diversity from Lipizzaner. Figure 25 shows the selection of random loss function to create a new generator $G_{u'}$. Mustangs were tested on MNIST and CelebA datasets, and they demonstrated statistically faster training compared to Lipizzaner.

Recently in July 2019, Cho et al. tried genetic algorithms to stabilize the training of GANs [48]. In this approach, authours attempted to improve the discrimination ability of the D and accordingly improve the performance of the generator, as shown in Figure 26. The chromosome in this approach is fake generated images by G. The synthetic images of high fitness were selected, i.e. the samples that were discriminated real by D and the population of fake images are evolved using a genetic algorithm. This approach was tested on MNIST dataset and authors claims to improve the convergence speed and GAN stability during training.

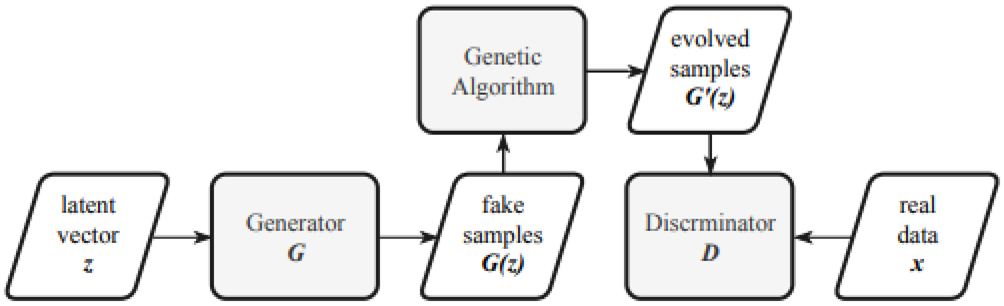


Figure 26 GAN combined with GA [48]

The weight-based neuroevolutionary training approaches were able to minimize the training instability. However, the GANs network topography and hyperparameter were empirically selected, wasting individual time on the monotonous experiments such as fine-tuning the network.

3.2 Neuroevolution of Topography

In this section, we will present the work where only the architecture of GANs is evolved.

Progressive GANs [10] uses a simple strategy to evolve GANs during the training procedure. The idea is to increases the number of layers progressively in both the generator and discriminator.

This progressive growing will make the model complex as the training proceeds. Also, the resolutions of training images are increased at each progression, as shown in Figure 27. However, the layers are preconfigured in this approach, i.e. they are hand-designed architecture. The progressive layers are not evolved using any stochastic method. Hence, the network model is evolved in a pre-configured way but does not use any evolutionary algorithm. Therefore, we consider this pre-defined progressive growing of GANs as the first approach to evolutionary Generative Adversarial Networks. Progressive Growing of

GANs used CelebA and CIFAR 10 dataset for their experimentation. Progressive Growing of GANs is state-of-the-art in image generation.

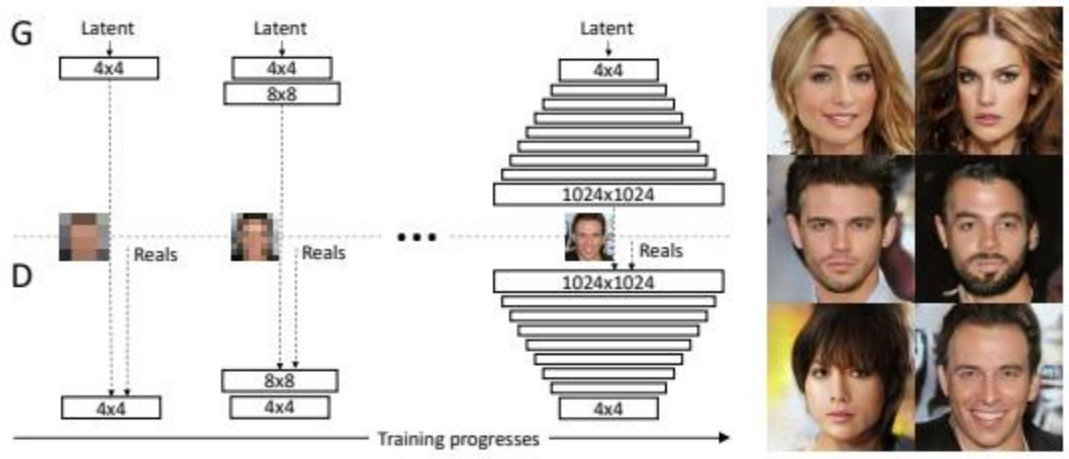


Figure 27 Progressive Growing of GANs [10]

In July 2018, Unai et al. presented evolved GANs for generating Pareto Set approximations [43]. In this paper, a neuroevolutionary approach in combination of a genetic algorithm is used to evolve the deep GANs architecture. The deep GANs architecture presented in the paper uses conventional GANs, i.e. Multi-Level Perceptron's (MLP) are used as the hidden layers in the GANs architecture. Our approach GAGAN is different in a way that we are using convolutional layers instead of MLP. However, our GAN evolvability components are inspired by this approach. Moreover, after every generation, the weight parameters of the trained networks are deleted. The evolved GANs are not used to generate images. Instead, they were used to generate Pareto set points.

The major drawback of evolving just the architecture of the network is that it does not take full advantage of the weights learned in the evaluation of the previous solutions.

3.3 Neuroevolution of Weights and Topography

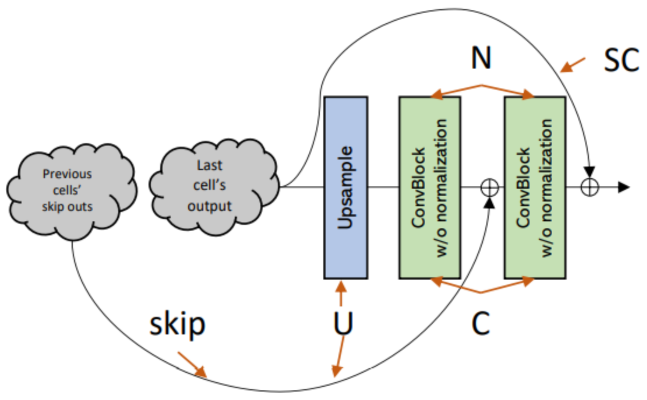
In this section, we will present the literature review of GANs whose weight parameter, as well as network architecture, are evolved.

Neuroevolution of Augmented Topology (NEAT) [8] is a famous approach to evolve weight and topography of neural networks. Deep Neat [40] was recently proposed to extend the NEAT approach to larger search space such as deep neural networks. Most of the GANs evolutionary approaches in this section are inspired from NEAT or Deep Neat based methods.

In March 2019, Costa et al. proposed coevolution of Generative Adversarial Networks (COEGAN) [12]. In COEGAN authours combines neuroevolution and coevolution in the coordination of the GAN training algorithm. In COEGAN the activation functions, number of hidden layers of the network, output channels and weight parameter are evolved in coevolutionary fashion. However, COEGAN approach does not take advantage of evolving different loss functions to train the GANs. Our approach is different in a way, we are using convolution layer while the COEGAN uses a combination of linear and convolutional layers. Moreover, in our approach, the GANs are evolved using genetic and cultural algorithms, and in COEGAN coevolutionary approach is used. COEGAN was evaluated with conventional DCGAN on MNSIT dataset using FID score.

In July 2019, an evaluation of COEGAN [49] was presented by Costa et al., wherein COEGAN was evaluated on Fashion-MNIST dataset. The evaluation suggests that COEGAN can be used as a training algorithm for GANs to avoid common issues, such as mode collapse.
Recently, in August 2019, a reinforcement learning-based approach was presented by Gong et al. for Neural Architecture Search (NAS) of GANs (AutoGAN) [50]. The search space of generator architecture was defined in AutoGAN, as shown in Figure 28.

Recurrent Neural Network (RNN) is used to guide the search, with the parameter sharing and dynamic resetting to accelerate the process. Inception score is adopted as a reward. Also, a multi-level architectural search is introduced to perform the neural architectural search. AutoGAN experiments were performed on CIFAR-10 and STL-10 datasets and evaluated with state-of-the-art GANs using FID score.



N
SC
Previous cells' skip outs
Last cell's output
Upsample
ConvBlock w/o normalization
ConvBlock w/o normalization
skip
U
C

Figure 28 Search space in AutoGAN [50]

However, there is a high computational cost associated with AutoGAN; it takes approximately 43 hours for training on CIFAR-10. Thus, only the generator's architecture was evolved.

Method	**Author**	**Type**	**Approach**
E-GAN [9]	Wang et al.	Weight parameter	Genetic
Lipizzaner [46]	Abdullah et al.	Weight parameter	Coevolution
Mustang [47]	Jamal et al.	Weight parameter	Coevolution
Stabilized GAN training [48]	Cho et al.	Weight parameter	Genetic
Progressive GAN [10]	Karras et al.	Topography	Hand-designed
Evolved GANs [43]	Unai et al.	Topography	Genetic

COEGAN [12]	Costa et al.	Weight and Topography	Coevolution
Evaluating COEGAN [49]	Costa et al.	Weight and Topography	Coevolution
AutoGAN [50]	Gong et al.	Weight and Topography	Reinforcement learning

Table 1 Comparision of various methods to evolve GANs

Table 1 summarizes the literature review for this thesis, in which it compares all different proposed approach for evolving Generative Adversarial Networks.

CHAPTER 4
Proposed Approach

In this chapter, we have discussed the proposed neuroevolutionary training algorithms for Deep Convolutional Generative Adversarial Networks (DCGAN). The chapter begins with the introduction of gene representation and fitness evaluation of the individuals. Thereafter this is followed by an explanation of pseudocodes of our proposed technique.

4.1 Proposed Training to Neuroevolve Deep Convolutional GANs

There are two proposed training strategies for evolving deep convolutional GANs. Both strategies are applied to automate the architecture search and stabilize the training of DCGAN. The names of the strategies are as stated below:

- Genetic neuroevolutionary training of deep convolutional GANs (GAGAN)
- Cultural neuroevolutionary training of deep convolutional GANs (CAGAN)

4.2 Individual Representation

In GAGAN and CAGAN, the genome is represented as an array of genes which are directly mapped into a phenotype consisting of a sequence of layers in a deep neural network. Each gene represents a convolutional or transpose convolutional layers. Moreover, each gene also has an activation function, chosen from the following set: ReLU, LeakyReLU, ELU, Sigmoid and Tanh. There is also a loss function gene associated with each genotype. The loss function is chosen from the following set: BCE, MSE, RSGAN, RAGAN and Hinge loss. From the specific parameter of each type of gene, convolutional and transpose convolutional layers only have the number of output channel as a random parameter. The strides and kernel size are fixed. The number of input channel are calculated dynamically based on the previous layer. Therefore, only the number of layers, activation function with each layer, output channels and loss function are subject to mutation operations.

Each individual's genotype is composed of two separate arrays of gene: one array represents Generator network G_i and the second array represents a Discriminator network D_i and a universal loss function gene. The individual genotype is represented by the following equation:

$$I_i = \{G_i, D_i, l\} \tag{25}$$

Figure 29 shows a sample of a discriminator phenotype. The discriminator is composed of three layers wherein each convolutional layer is followed by batchnorm2d layer and activation function. The output channel of the previous layer will be the input channel of the current layer.

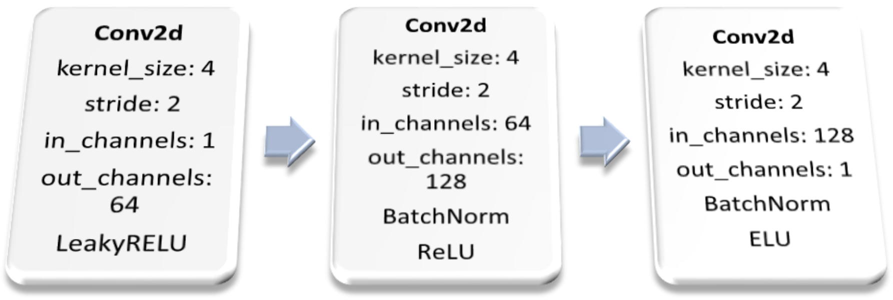


Figure 29 A phenotype of the discriminator

Figure 30 shows a sample of a generator phenotype. The generator network is also composed of three layers. Each convolutional transpose layer is followed by batchnorm2d layer and activation function. Similar to discriminator in the generator output channel of the previous layer will be the input channel of the current layer.

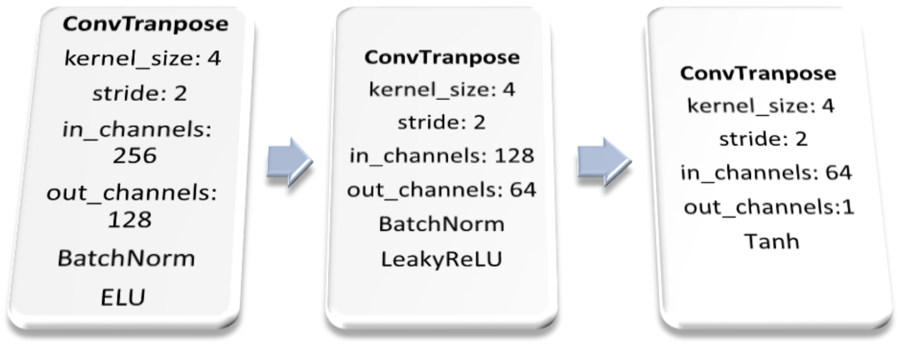


Figure 30 A phenotype of the generator

In the population space, there is a list of individual GAN which represents genetic components. Thus, GAGAN and CAGAN use a list-based encoding and genetic operators that operate on these lists. Each network parameter list includes convolutional or transpose convolutional layer, output channel in each layer, activation function for each layer. In generator genotype, all hidden layers are composed of transpose convolutional section followed by batch normalization and activation function. While in discriminator genotype, all hidden layers are convolutional, followed by batch normalization and activation function. These hidden layers design is inspired by DCGAN [28]. The specification of the hidden layers (e.g. weights and bias) will be trained by a variation of the gradient descent method and will not be part of the evolution. However, the weights of the networks are preserved over the generations for each individual. During the evaluation step, fake generated images are used to assess the quality of the individual by a predefined fitness function.

4.3 Individual Generation

We will identify the GAN components which will be used to generate the individual. As described in Equation 25, each GAN individual consists of a Generator Network, a Discriminator Network and a loss function.

The following components are defined to initialize an Individual:

1. Architecture of Generator. This includes:
    a. Number of layers of the architecture
    b. Activation function for each layer
    c. Output channel for each layer
2. Architecture of Discriminator (with similar elements to be considered to those for the generator)
3. Loss function used to train both the architecture
4. Weight initialization technique for each network

We will encode the value of each gene in a categorical way.
Activation Functions and Loss Functions encoding is described as follows:

Activation Function	**Encoding**
LeakyReLU	0
ReLU	1
ELU	2
Sigmoid	3
Tanh	4

Table 2 Activation Functions Encoding

Loss Function	**Encoding**
Binary Cross-Entropy (Vanilla GAN)	0
Mean Squared Error (LSGAN)	1
Relativistic Standard GAN (RSGAN)	2
Relativistic Average GAN (RAGAN)	3
Hinge GAN	4

Table 3 Loss Functions Encoding

The weight initialization is encoded 0 for Normal and 1 for Xavier initialization. The output channels are randomly chosen from a range of 64 to 512, and the outputs channels are in multiples of 64. Selection of output channels is inspired by DCGAN architecture [31].

Example of a genotype encoding can be shown in the following table:

Gene	Generator	Discriminator
Number of layers	5	6
Activation functions	[1,1,0,2,4]	[0,0,0,1,0,3]
Output channels	[512,256,128,64,64]	[64,64,128,256,512,512]
Weight initialization	0	1
Loss Function	2	

Table 4 Individual GAN genotype

During the population initialization N, such GAN genotypes are generated.

4.4 Fitness Evaluation

The fitness evaluation is the process of measuring the fitness of an individual. We have tried to use the loss function of the individual as a fitness evaluation metric. However, preliminary experiments evidenced that the loss function does not represent a good measure for quality. Since the loss functions are unstable during the training of the GANs, it is not suitable to be used as a fitness function in evolutionary algorithms.

Fréchet Inception Distance (FID) [51] is the state-of-the-art metric to compare the generative component of the GANs and outperforms other evaluation metrics, such as the Inception score [11] with respect to diversity and quality. Inception Net [19] is used in FID, and it is trained on ImageNet [51]. This Inception Net is used to transform the images to feature space. This feature space is interpreted as a continuous multivariate Gaussian [12]. So, the mean and covariance of two Gaussians are estimated using real and fake samples. The FID score between two Gaussians is given by the following equation:

$$\mathrm{FID}(x, g) = ||\mu x - \mu g||\mathbf{22} + \mathrm{Tr}(\Sigma x + \Sigma g - \mathbf{2}(\Sigma x \Sigma g)^{1/2}) \tag{26}$$

In Equation 26, μx, Σx, μg and Σg represent the mean and covariance estimated for the real dataset and fake samples, respectively.

4.5 Genetic Neuroevolutionary Training of Deep Convolutional GANs

One of the simplest way to evolve any neural network can be done through the use of Genetic Algorithms. Figure 31 shows a visual representation of the usage of genetic algorithm in neuroevolution. Where the weights of the neural networks are evolved using a genetic algorithm, then the network is asked to perform some action on the environment and based on the action, a fitness score is calculated. However, there is a problem when this approach is applied to deep convolutional neural networks, the parameters required to optimize deep networks increases to hundreds of thousands in number. Neuroevolution of Augmented topologies (NEAT) [8] deals with this problem, where authours instead of evolving weights of the network evolves the topology of the network. The weights are trained through traditional stochastic gradient descent algorithm [52].

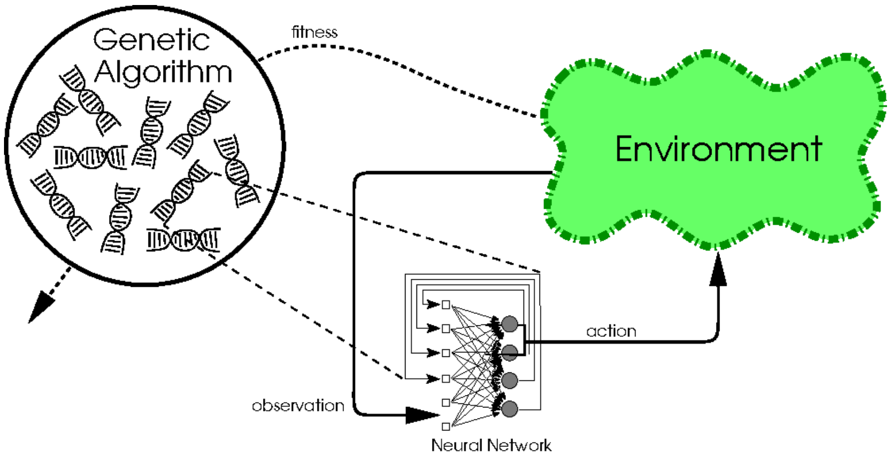


Figure 31 Genetic Neuroevolution of Neural Network

GAGAN model combines neuroevolution and genetic algorithms in the coordination of the deep Convolutional GAN training algorithm. Our approach is based on NEAT [8], that was adapted to the context of GANs. Algorithm 1 presents the GAGAN, a genetic neuroevolutionary training of deep convolutional GANs.

In this section, we will give a detailed explanation of genetic operations like crossover and mutations performed in the algorithm. At the end of this section, each step of Algorithm 1 is analyzed.

4.5.1 Crossover

The crossover operation is used to combine genetic information of two parents to generate new offsprings. Initially, we have tried to apply a k-point crossover. We tried to pick randomly k different genes and swap between the parent individuals. However, preliminary test evidenced that k-point crossover decreases the performance of the system. Such a decrease in performance happens because when the new gene is introduced in the individual genotype, the trained weights of the network are not compatible.

So for our crossover operation, we are using the single-point crossover. Let two selected parent individuals from the population be represented as follows:

$$P_1 = (G_{1,} D_1)\ ,\ P_2 = (G_{2,} D_2)$$

The crossover operation creates two offsprings as follows:

$$O_1 = (G_2, D_1)\ ,\ O_2 = (G_{1,} D_2)$$

Crossover preserves the integrity of each network with trained weights and bias.

4.5.2 Mutation

The mutation operation is composed of five primary operations, as stated in

Table 5. One of the generator network or discriminator network is selected to mutate, from the elected individual of the population space. For discriminators, the available layer is convolution layer, and for generator, the available layer is transpose convolution. The maximum and minimum number of layers' range is the input parameter for the GAGAN algorithm.

Mutation Operations
Add Layer
Delete Layer
Activation function change
Output channel change
Loss function change

Table 5 Mutation operations

In Add layer mutation, according to the network type, a new layer is added. Add layer mutation operation will never exceed the maximum range of layers. With the new layer, batchnorm layer and randomly selected activation function are also added. Also, the input channel of the next layer is reinitialized according to the output channel of the new layer.
Similarly, in Delete layer mutation, a convolutional or transpose convolution layer is deleted. Input channel of the next layer is appropriately changed according to the previous layer output channel. The weights of the layers are not altered during this mutation operation. The Delete layer mutation will never go beyond the minimum range.
If the mutation operation chooses activation function change, then a random layer is selected of the previously elected GAN network. A new activation function from Table 2 is selected, and it is used to replace the elected networks activation function.
A layer is arbitrarily chosen for which output channel change is to be performed. One new channel number is selected from the range of 64-512 and is replaced with the chosen layer output channel. Because of the output channel mutation, the input channel of the next layer is also changed dynamically.
The loss function is selected from Table 3 and is replaced with the existing loss function of the individual. The mutation of these attributes follows a uniform distribution, with a predefined range limiting the possible values.

4.5.3 GAGAN Algorithm

The purpose of the GAGAN algorithm is to search for best network topography and train the network to improve the generation of the images.

The algorithm starts with the creation of population space according to the input parameter population_size. An individual in the population is initialized as described in 4.3 Individual Generation. All the individuals of the population are trained for one epoch. Each epoch has a predefined batch size and training iteration as the input of the algorithm. The parameters of the epoch should be sufficient enough for the deep neural networks to learn some underlying task. Then a fitness score for each individual chromosome is calculated, as shown in Equation 26. Lower the FID score, better the individual solution. Hence, the best individual is saved at this step of the algorithm, as shown in Algorithm 1 step (6-8).

New solutions are generated using genetic operations. In which population individuals become the parent, and Mutation(asexual) or Crossover(bisexual) reproduction is used to generate the offsprings. The size of the offspring is equal to the population size, and according to the predefined probability, Crossover or Mutation operation is chosen. Two random individuals are selected from the population, and as described in section 4.5.1 Crossover operation is performed. For mutation operation, any random individual is elected from the population space, and as explained in section 4.5.2 Mutation operation is done. It is important to note that the choice of the mutation is made uniformly at random. The mutation and crossover are presented in step 10 of Algorithm 1. In step (11-17), the offsprings are trained, and the FID score is calculated for each new individual. If the offspring is better than the current best individual, then the best individual is updated with that offspring.

```
Algorithm 1 Genetic Algorithm(GA) for evolving deep convolutional GANs
  Input: training_images, population_size, generation_max, iteration, layers_range
  Output: best trained GAN individual
 1: Set g ⇐ 0. Initialize population $P_0$ by generating population_size of GANs.
 2: while g < generation_max do
 3:    for ind in range (population_size)  do
 4:       Train ind for one epoch
 5:       Calculate fitness F using FID score
 6:       if fitness(best_ind) > F then
 7:          Update best_ind = ind
 8:       end if
 9:    end for
10:    Create offspring set $O_g$ by applying crossover with probability $P_x$=0.1
       and mutation with probability $P_m = 1 - P_x$=0.9.
11:    for offspring in range ($O_g$)  do
12:       Train offspring for one epoch
13:       Calculate fitness of offspring using FID score
14:       if fitness(best_ind) > F then
15:          Update best_ind = ind
16:       end if
17:    end for
18:    Create $P_{g+1}$ by using tournament selection from $\{P_g, O_g\}$
19:    g ⇐ g + 1
20: end while
21: return  best_ind
```

Algorithm 1 Genetic Algorithm(GA) for evolving deep convolutional GANs

In step 18, the selection operation is performed. Selection gives preference to the better chromosomes to pass their genome to the next generation of GAGAN algorithm. Different types of selection method exist, empirically we have chosen tournament selection as it gives the best results in our problem space. Tournament selection involves running multiple tournaments among the randomly chosen k individuals. Using tournament selection gives the diversity and best offspring for the upcoming generation. The whole process is repeated until the specified number of generation, generation_max is also one of the input parameters of the algorithm. The number of generation should be sufficient for individuals to show the convergence. Finally, the bests individual having the lowest fitness score is returned as the output of the algorithm.

4.6 Cultural Neuroevolutionary Training of Deep Convolutional GANs

Unlike Genetic Algorithms, Cultural Algorithm enables the population to adapt to their changing environment at a rate that exceeds that of biological evolution. CAGAN model combines neuroevolution and cultural algorithm in the coordination of the GAN training algorithm. In this section, we will introduce the cultural components and how to adjust those components, followed by usage of the culture to exceed the natural evolution. Finally, we will analyze all the steps of CAGAN.

4.6.1 Adjusting Cultures

In CAGAN, we maintain two search spaces: the population space, and a belief space (to represent cultural component). The belief space models cultural information about the population, while population space represents individuals. Both the population space and belief space evolve in parallel, with both influencing one another.

In our method, we have used Situational, Domain and Normative knowledge to adjust the belief space and influence the GAN population. Mathematically, the belief space is represented as follows

$$B(t) = [S(t), N(t), D] \tag{27}$$

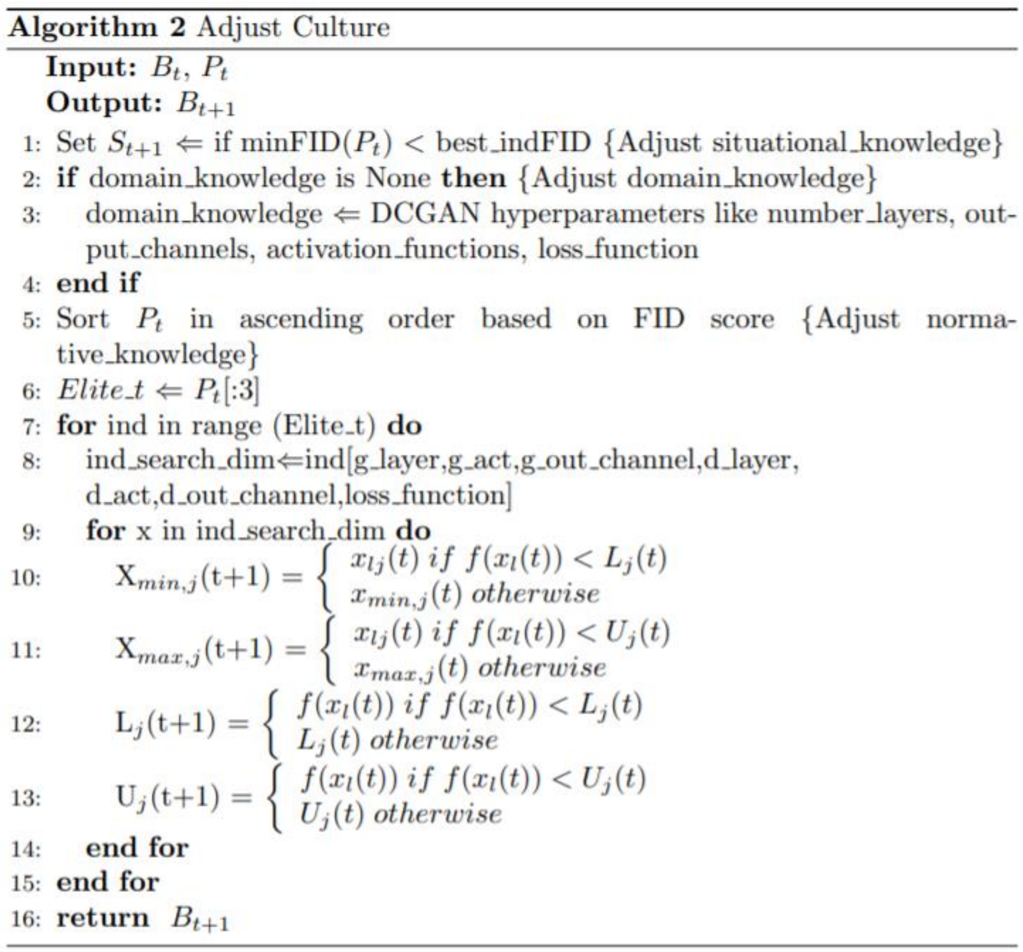
Where $B(t)$ represents belief space at generation t, $S(t)$, $N(t)$ and D represents the Situational, Normative and Domain knowledge components respectively. Situational and Normative component are updated simultaneously in every generation.

Algorithm 2 shows how these knowledge components are updated in each generation. We will explain the adjustment of each knowledge components in the following sections.

4.6.1.1 Situational Knowledge

Let $S(t)$ have an individual who has the fitness value represented as $best\_indFID$. If during the adjustment of situational component there is an individual in population Pt who has FID score lesser than $best\_indFID$; then we update the situational component as shown in step 1 of Algorithm 2. The best individual from the Pt is set as a situational component. At all-time the $S(t)$ will have only one individual.

This property of storing the best individual is known as Elitism. Elitism guarantees that the evolutionary process converges. However, the chances of converging to a local optimum also increase due to elitism.



```
Algorithm 2 Adjust Culture
  Input: $B_t$, $P_t$
  Output: $B_{t+1}$
1: Set $S_{t+1} \Leftarrow$ if minFID($P_t$) $<$ best_indFID {Adjust situational_knowledge}
2: if domain_knowledge is None then {Adjust domain_knowledge}
3:     domain_knowledge $\Leftarrow$ DCGAN hyperparameters like number_layers, output_channels, activation_functions, loss_function
4: end if
5: Sort $P_t$ in ascending order based on FID score {Adjust normative_knowledge}
6: $Elite\_t \Leftarrow P_t[:3]$
7: for ind in range (Elite_t) do
8:     ind_search_dim$\Leftarrow$ind[g_layer,g_act,g_out_channel,d_layer,d_act,d_out_channel,loss_function]
9:     for x in ind_search_dim do
10:        $X_{min,j}(t+1) = \begin{cases} x_{lj}(t)\ if\ f(x_l(t)) < L_j(t) \\ x_{min,j}(t)\ otherwise \end{cases}$
11:        $X_{max,j}(t+1) = \begin{cases} x_{lj}(t)\ if\ f(x_l(t)) < U_j(t) \\ x_{max,j}(t)\ otherwise \end{cases}$
12:        $L_j(t+1) = \begin{cases} f(x_l(t))\ if\ f(x_l(t)) < L_j(t) \\ L_j(t)\ otherwise \end{cases}$
13:        $U_j(t+1) = \begin{cases} f(x_l(t))\ if\ f(x_l(t)) < U_j(t) \\ U_j(t)\ otherwise \end{cases}$
14:    end for
15: end for
16: return $B_{t+1}$
```

Algorithm 2 Adjust Culture

4.6.1.2 Domain Knowledge

The domain knowledge component differs from the situational knowledge component in that knowledge is not re-initialized at each generation, but contains an archive of the best solutions since evolution started – very similar to the hall-of-fame used in coevolution.
In our domain knowledge, we are storing the hyperparameters of a benchmark DCGAN [31] model. Hyperparameters like generators and discriminators activation function, number of layers, output channels for each layer and loss functions are stored in a dictionary in belief space. The domain knowledge is initialized as described in step 2-4 of Algorithm 2.

4.6.1.3 Normative Knowledge

The normative knowledge component maintains a set of intervals, one for each dimension of the problem is solved. These intervals characterize the range of what is believed to be good areas to search in each dimension [38].
The normative component is represented as follows

$$N(t) = (g_{layer}(t), g_{act}(t), g_{out}(t), d_{layer}(t), d_{act}(t), d_{out}(t), loss(t)) \tag{28}$$

Where,
$g_{layer}(t)$: Generator layer count component
$g_{act}(t)$: Generator activation component
$g_{out}(t)$: Generator output channel array component for each layer
$d_{layer}(t)$: Discriminator layer count component
$d_{act}(t)$: Discriminator activation component
$d_{out}(t)$: Discriminator output channel array component for each layer

Where for each dimension following information is stored:

$$X_j(t) = (I_j(t), L_j(t), U_j(t)) \tag{29}$$

$I_j(t)$ denotes a closed interval, $I_j(t) = [\, x_{min,j}\,(t), x_{max,j}\,(t)]$, $L_j(t)$ is the score for the lower bound, and $U_j(t)$ is the score for the upper bound.

In adjusting the normative knowledge component, a conservative approach is followed when narrowing the intervals, thereby delaying a too early exploration. To update the normative component, top three elites are elected from the population P_t. For all the dimension of the search space, these three elites are used to update the normative knowledge, as shown in the steps (9-14) of Algorithm 2.

Where,

x_l(t): l^{th} elite at generation t

$f(x_l(t))$: fitness value of the l^{th} elite individual at generation t

$x_{lj}(t)$: is the value of j^{th} gene of the l^{th} elite at generation t

$x_{min,j}\,(t)$: is the value of the jth gene of the l^{th} elite whose fitness value is less than that of the individual with the smallest j^{th} gene at generation t

$L_j(t)$: represents the fitness value of the elite that is less than the fitness value of the individual having the smallest j^{th} gene at generation t

$x_{max,j}\,(t)$: would signify the value of the j^{th} gene of the l^{th} elite whose fitness value is less than that of the individual with the highest j^{th} gene at generation t

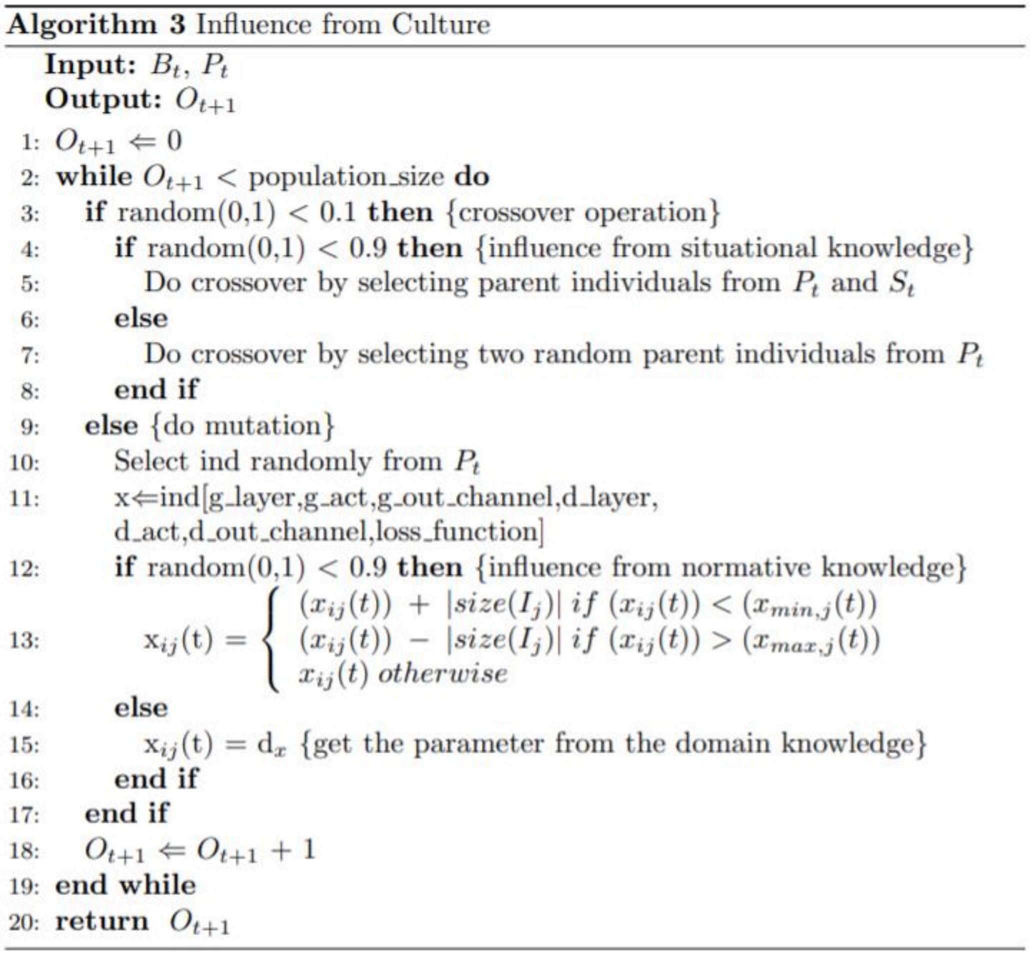
$U_j(t)$: would represent the fitness value of the individual that is less than the fitness value of the individual having the largest j^{th} gene at generation t

4.6.2 Influence Functions

Beliefs are used to adjust individuals in the population space to conform closer to the global belief space. The adjustments are realized via influence functions.

In our model, the belief space is used to generate new offsprings by using genetic operations like mutations and crossover. The belief knowledge will be used to determine the search direction and step sizes.

The input to Algorithm 3 is the belief space B_t and population P_t. At the end of the algorithm offspring O_{t+1} of the population size is created. It is important to note that during the offspring creation, the weights of the networks are preserved showcasing **transfer learning** between the generations.



```
Algorithm 3 Influence from Culture
    Input: $B_t$, $P_t$
    Output: $O_{t+1}$
 1: $O_{t+1} \Leftarrow 0$
 2: while $O_{t+1} <$ population_size do
 3:   if random(0,1) $< 0.1$ then {crossover operation}
 4:     if random(0,1) $< 0.9$ then {influence from situational knowledge}
 5:       Do crossover by selecting parent individuals from $P_t$ and $S_t$
 6:     else
 7:       Do crossover by selecting two random parent individuals from $P_t$
 8:     end if
 9:   else {do mutation}
10:     Select ind randomly from $P_t$
11:     x⇐ind[g_layer,g_act,g_out_channel,d_layer,
        d_act,d_out_channel,loss_function]
12:     if random(0,1) $< 0.9$ then {influence from normative knowledge}
13:       $\mathrm{x}_{ij}(\mathrm{t}) = \begin{cases} (x_{ij}(t)) + |size(I_j)| \; if \; (x_{ij}(t)) < (x_{min,j}(t)) \\ (x_{ij}(t)) - |size(I_j)| \; if \; (x_{ij}(t)) > (x_{max,j}(t)) \\ x_{ij}(t) \; otherwise \end{cases}$
14:     else
15:       $\mathrm{x}_{ij}(\mathrm{t}) = \mathrm{d}_x$ {get the parameter from the domain knowledge}
16:     end if
17:   end if
18:   $O_{t+1} \Leftarrow O_{t+1} + 1$
19: end while
20: return $O_{t+1}$
```

Algorithm 3 Influence from Culture

The probability of performing the crossover operation is decided to be 0.1 empirically to prevent the algorithm from local minima. In the crossover operation, the offsprings are created using situational knowledge and domain knowledge. Crossover operation is identical to the crossover in GAGAN as defined in section 4.5.1 Crossover.

Unlike in GAGAN, the mutation operation uses Normative knowledge and Domain knowledge. For any x mutation dimension, the search direction and step sizes are determined using the Normative knowledge as demonstrated in step (10-13) of Algorithm 3 Influence from Culture

Where,

$x_{ij}(t)$: represents the value of the j^{th} gene of the i^{th} individual

$size(I_j)$: $x_{max,j}(t) - x_{min,j}(t)$

If $x_{ij}(t)$ is less than $x_{min,j}(t)$, then it is incremented by the normative knowledge size for that gene, and if it is higher than the knowledge range, then it is decremented or else it is kept unaltered.

If domain knowledge is elected for the mutational adjustment, then the gene j of i^{th} individual is replaced from the domain knowledge.

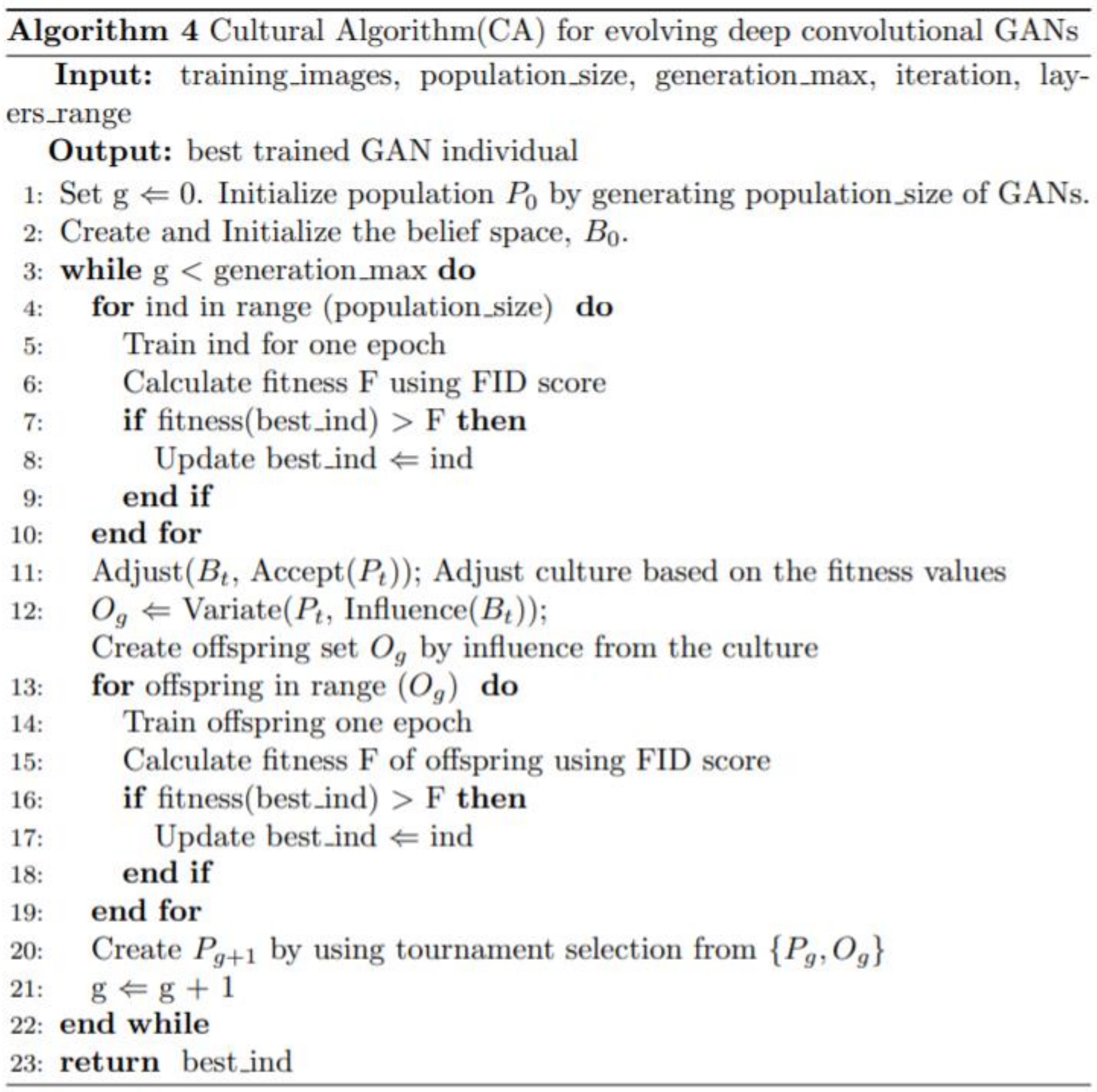
4.6.3 CAGAN Algorithm

CAGAN algorithm starts with the initialization of population space, similarly to GAGAN as presented in 4.3 Individual Generation. In the next step, Belief space B_0 is initialized, where domain knowledge dictionary is created according to the hyperparameters of DCGAN, Situational knowledge and Normative knowledge are kept empty. At the start of evolution, all the individuals are trained for one epoch, and their fitness scores are calculated as explained in 4.4 Fitness Evaluation.

A reference of best_ind is kept throughout the generations, if the fitness value of any individual is less than the fitness value of best_ind then the best individual is updated as stated in step (4-9) of Algorithm 4.

Training of individual is followed by adjustment of belief space. The detailed explanation of the belief space modification is mentioned in 4.6.1 Adjusting Cultures. The belief space will be used in the creation of the new offspring; the individuals in the population are variated using the cultural influence. This variation of the population space is shown in step 12, and the detailed description is in 4.6.2 Influence Functions.

The new offsprings are then trained for one epoch, and their fitness scores are calculated. This process is shown in step (13-19) of Algorithm 4. After that, using the tournament selection operation, the next population set is created. This evolution process is repeated over the number of generations, which is the input parameter of the algorithm.



Algorithm 4 Cultural Algorithm(CA) for evolving deep convolutional GANs

Input: training_images, population_size, generation_max, iteration, layers_range

Output: best trained GAN individual

```
1:  Set g ⇐ 0. Initialize population $P_0$ by generating population_size of GANs.
2:  Create and Initialize the belief space, $B_0$.
3:  while g < generation_max do
4:    for ind in range (population_size) do
5:      Train ind for one epoch
6:      Calculate fitness F using FID score
7:      if fitness(best_ind) > F then
8:        Update best_ind ⇐ ind
9:      end if
10:   end for
11:   Adjust($B_t$, Accept($P_t$)); Adjust culture based on the fitness values
12:   $O_g$ ⇐ Variate($P_t$, Influence($B_t$));
      Create offspring set $O_g$ by influence from the culture
13:   for offspring in range ($O_g$) do
14:     Train offspring one epoch
15:     Calculate fitness F of offspring using FID score
16:     if fitness(best_ind) > F then
17:       Update best_ind ⇐ ind
18:     end if
19:   end for
20:   Create $P_{g+1}$ by using tournament selection from $\{P_g, O_g\}$
21:   g ⇐ g + 1
22: end while
23: return best_ind
```

Algorithm 4 Cultural Algorithm (CA) for evolving deep convolutional GANs

Once the last generation is completed, the GAN individual having the lowest FID score is then returned. Best GAN individual is saved with the weights and can be used as an API to generate new images.

CHAPTER 5

Experiments and Results

In this chapter, we present different types of datasets used for experimentation, which is followed by the experimental setup for the proposed algorithms and the result obtained from those experiments.

5.1 Datasets

We will evaluate the performance of our method on three different datasets. Two of them are standard benchmark dataset for GANs, and the third one (Stroke Faces) is created by us. Following are the names and sample training images of these datasets.

- MNIST handwritten digit dataset [17]- consists of 60000 training images and 10000 testing images. Each image is grayscale of size 28*28, but for our experimentation, we scale the images to 64*64.

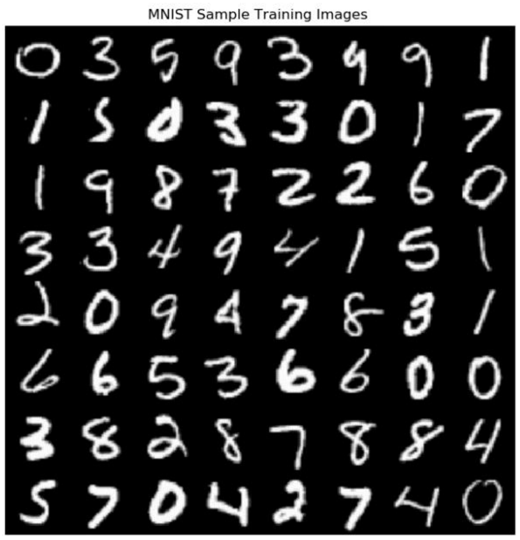


Figure 32 MNIST Dataset

- Fashion MNSIT [18] also consists of 60000 training examples and 10000 test images. Each example is a 28*28 grayscale image, and dataset has 10 different labelled classes. Similar to MNIST, we scale the dataset to 64*64 grayscale images.

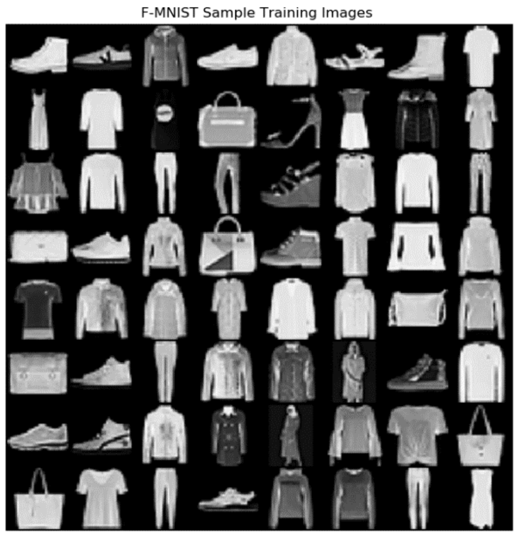


Figure 33 Fashion MNIST Dataset

- Stroke Faces- Stroke faces dataset is one of our contribution. It consists of 3 labelled classes of child, men and women having a stroke or facial paralyzes. This dataset has total 1280 samples scraped from google images and is pre-processed to 64*64 size grayscale images.

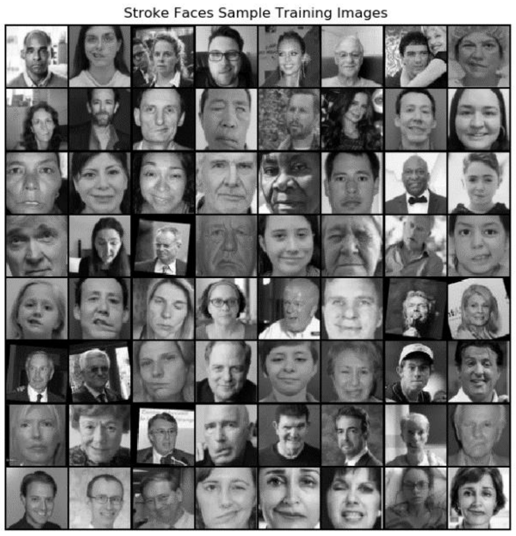


Figure 34 Stroke Face Dataset

Usually, the network would be training for several epochs using the whole dataset in the procedure. But there are several domains where there is a lack of dataset, so we are testing our approach in low data regime. We will only use a small subset of the dataset per generation. Combining the small dataset, with the transfer of parameters between the generations, was sufficient to produce an evolutionary pressure towards efficient solutions to promote the GAN convergence.

5.2 Experimental Setup

Evolutionary Parameters	Value
Number of Generations	50
Population size	7
Crossover rate	0.1
Mutation rate	0.9
Layers range	[5:8]
Output channel range	[64:512]
Tournament size	3
FID samples	1000
Batch size	64
Batches per generation	20
Optimizer	Adam
Learning rate	0.001

Table 6 Experimental parameters

Table 6 describes the parameters used in all experiments reported in this thesis.

For evolutionary parameters, we chose to execute our experiments for 50 generations. After this number of generations, the fitness stagnates, and we expect no improvement in the results. We have used 7 individuals for the population, i.e. there will be 7 different generators and discriminators in the population. We choose 7 because that was the maximum computational power we had for carrying out our experiments. A larger population will probably achieve better results, but the computational cost is too high. The maximum layer range of 8, is also restricted for the same reason. We empirically define the probability of crossover to be 0.1 and mutation to be 0.9; this is because we want the evolutionary algorithms to avoid the local optimum.

For the GAN parameters, we choose 64 as the batch size, running 20 batches per generation. This amounts to **1280 samples** per generation to train the individual. The optimizer used in this method is Adam [53].

For all the three dataset MNIST, F-MNIST and Stroke face, we executed both the proposed model three times.

5.3 Using GAGAN to Generate MNIST Images

FID score is calculated by comparing 1000 generated images and original dataset images. Figure 35 shows the progression of the FID fitness score for the best individual represented by GAGAN. Moreover, the second line *AVG GAGAN*, shows the average of the best individuals FID score at each generation achieved in three runs. We can see the fitness of the generator reducing through generation with reduced noise.

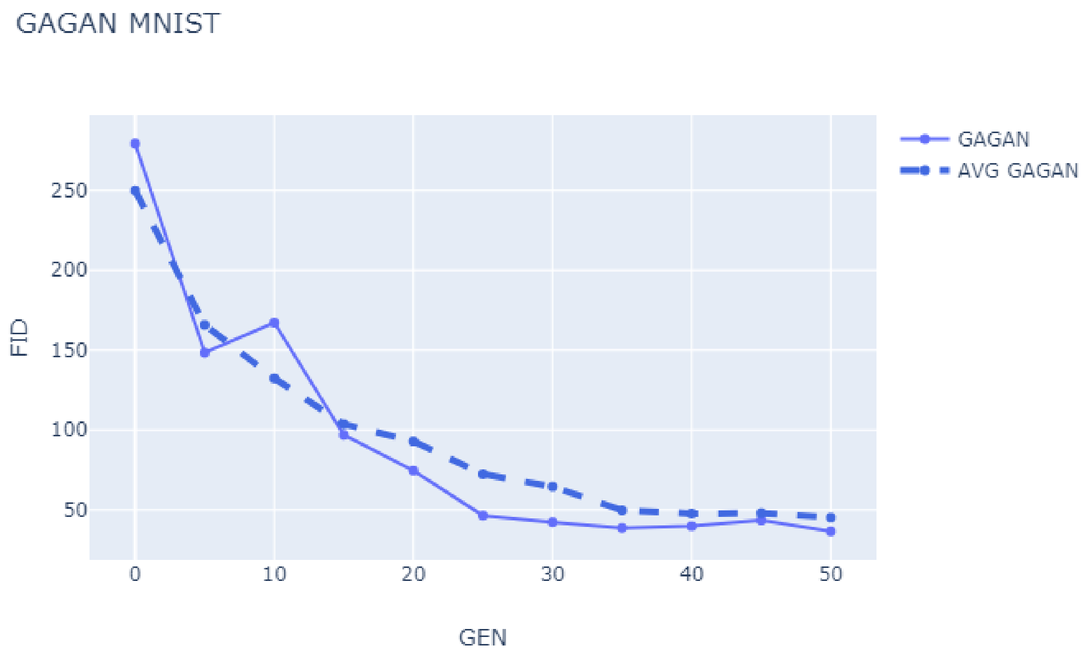


Figure 35 Graph showing the best FID score achieved at each generation versus the mean of three runs for GAGAN MNIST

In the final generation, the lowest FID was reported to be 36.61, with a standard deviation of $\sigma = 7.15$.

Figure 36 contains generated samples selected to represent the progression of the generator during the evolutionary algorithm. We can see in the first generation only noisy samples, without any structure resembling a digit. From generation 25 we can start distinguishing between the digits, with a progressive improvement of the quality.

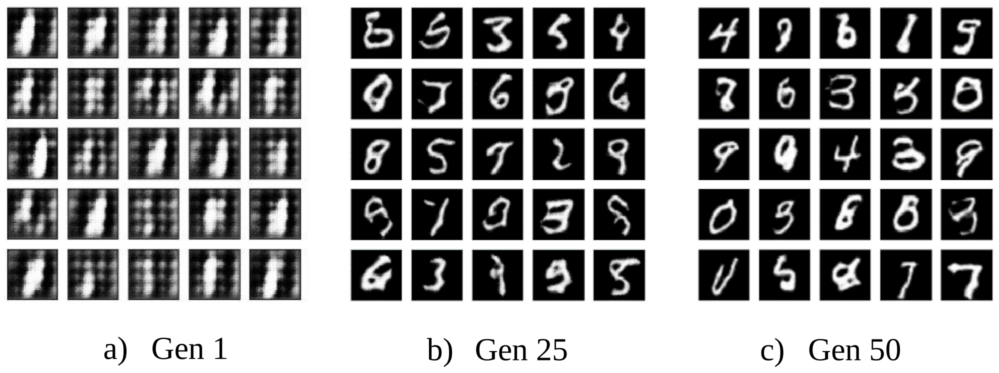


Figure 36 The progression of MNIST samples created by best GAGAN generator in generations a) 1, b) 25 and c) 50

5.4 Using GAGAN to Generate F-MNIST Images

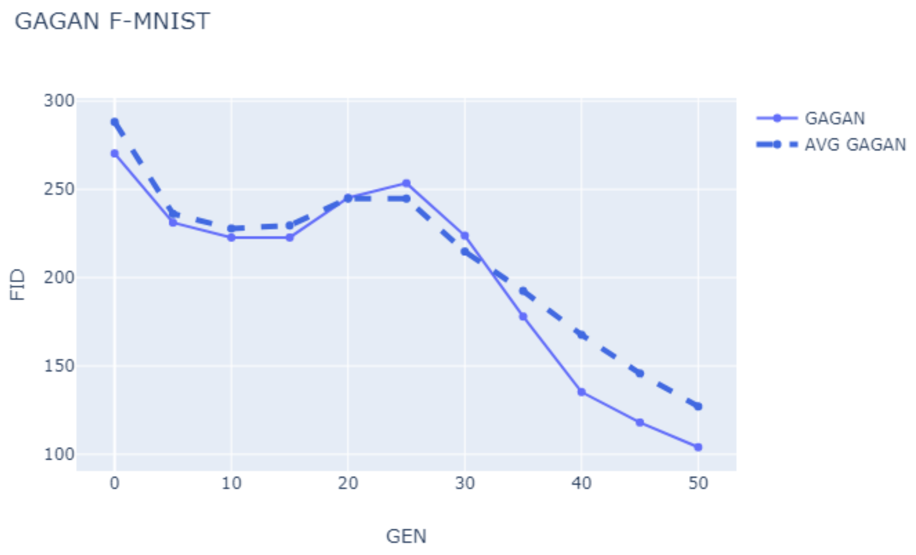


Figure 37 Graph showing the best FID score achieved at each generation versus the mean of three runs for GAGAN F-MNIST

Figure 37 shows the graph of the best individual generators FID scores over the 50 generations, versus the mean of the best individuals in the three runs of the experiments for Fashion MNSIT dataset. In the last generation for GAGAN F-MNIST, the lowest FID was reported to be 104.02, with a standard deviation of $\sigma = 20.68$.

Figure 38 shows a similar progression of the best individual GAGAN for the F-MNIST.

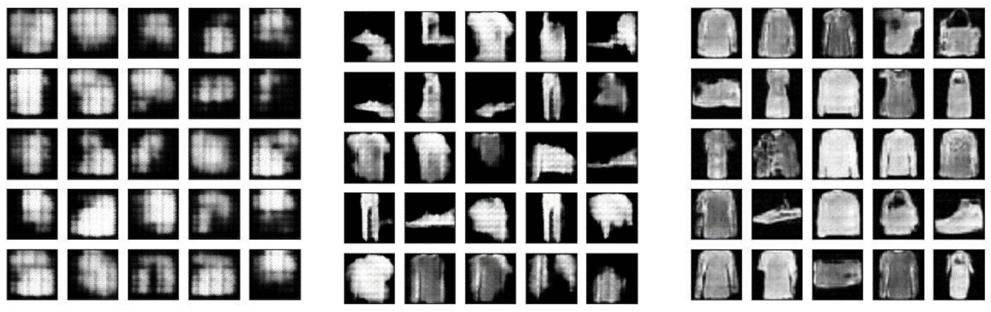


Figure 38 The progression of F-MNIST samples created by best GAGAN generator in generations a) 1, b) 25 and c) 50

5.5 Using GAGAN to Generate Stroke Faces

Similar to another dataset we have compared GAGAN generated stroke faces using FID score as shown in Figure 39. The best individuals FID score is plotted with the average FID score of top individuals from the three runs of the experiments over 50 generations. The best FID score for GAGAN stroke faces is recorded to be 93.16. From the three runs of this GAGAN model for stroke faces the standard deviation in the last generation is $\sigma = 26.99$.

Figure 40 shows the progression of the stroke faces over the generation. In the first generation, only noise is generated. At generation 25, we can distinguish some faces having the droopy stroke effect. In generation 50, we can see some improvement in the quality of generated stroke faces. However, there is some noise also seen in the generated faces. This is because the dataset we are training on is minimal and when compared to benchmark models using FID score as shown in section 6.3 Comparison between Vanilla GAN, DCGAN, GAGAN and CAGAN to generate Stroke faces our proposed models generates better quality images.

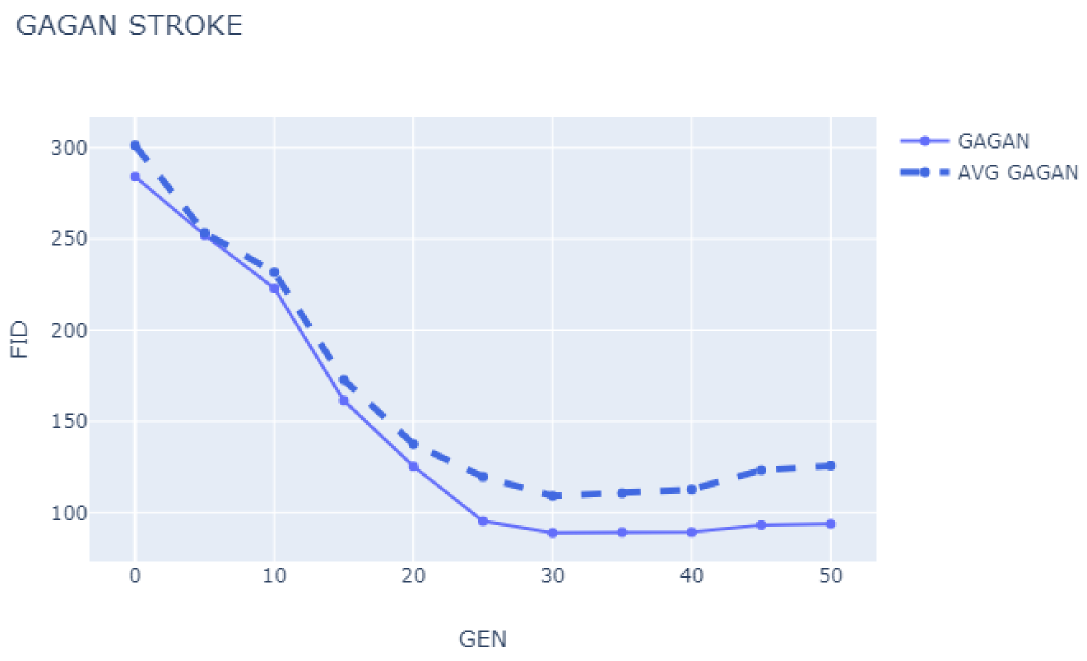
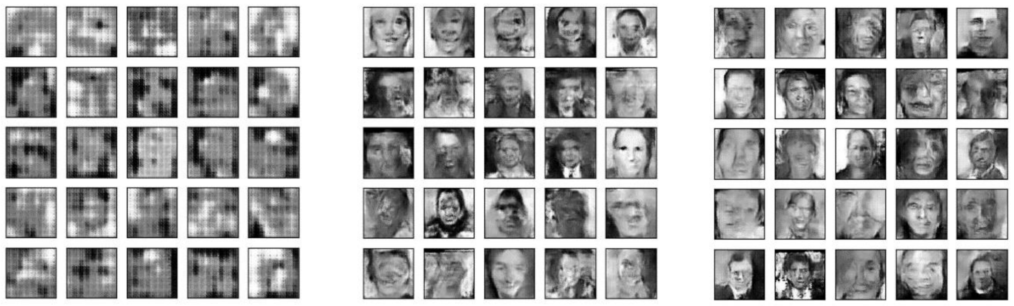


Figure 39 Graph showing the best FID score achieved at each generation versus the mean of three runs for GAGAN Stroke Faces



a) Gen 1 b) Gen 25 c) Gen 50

Figure 40 The progression of Stroke face samples created by best GAGAN generator in generations a) 1, b) 25 and c) 50

5.6 Using CAGAN to Generate MNIST Images

In this section, we will compare our second model CAGAN to generate MNIST images. As mentioned in section 5.2 Experimental Setup, only 1280 sample training images are used to train this model. In the following chapter, we will compare both the proposed technique with other benchmark models.

Figure 41 shows the progression of the FID fitness score for the best individual represented by CAGAN. Also, the second line *AVG CAGAN*, shows the mean FID score of the best individuals at each generation achieved in three runs. We can see the fitness of the generator reducing through generations.

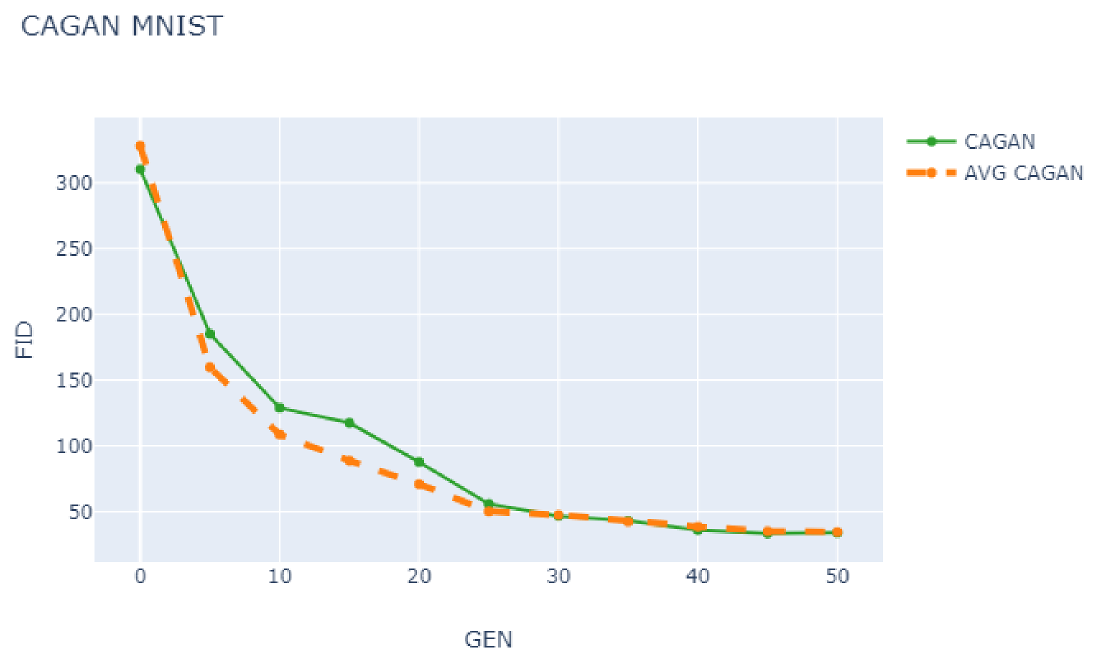


Figure 41 Graph showing the best FID score achieved at each generation versus the mean of three runs for CAGAN MNIST

Figure 42 shows the generated images of the best individual of CAGAN population. Visually it is hard to compare the images generated by different models, that is why we rely on FID score to validate the performance of the different models. The best FID score for MNIST dataset by CAGAN is reported to be 33.37, with the standard deviation for the last generation in total three-run is $\sigma = 0.80$.

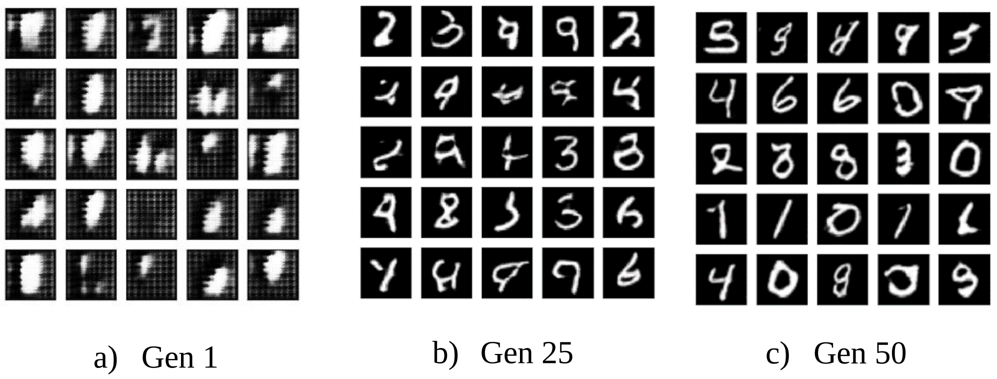


Figure 42 The progression of MNIST samples created by best CAGAN generator in generations a) 1, b) 25 and c) 50

5.7 Using CAGAN to Generate F-MNIST Images

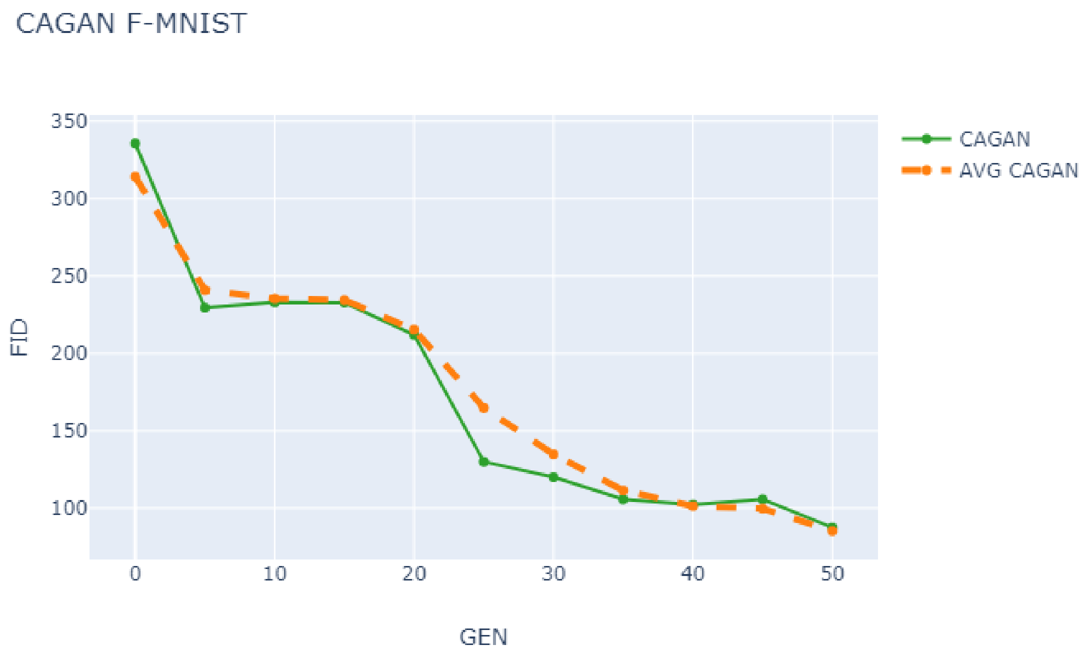
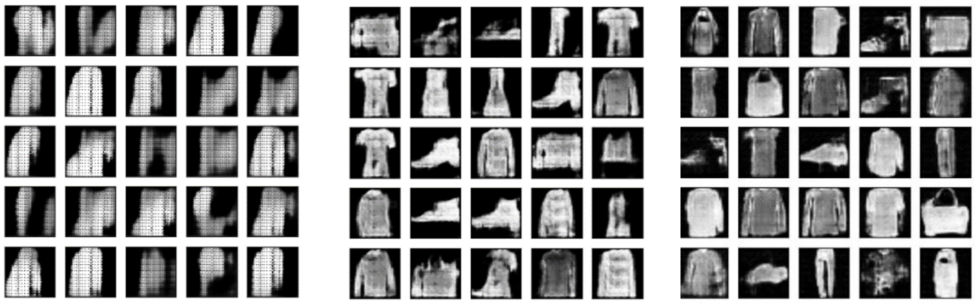


Figure 43 Graph showing the best FID score achieved at each generation versus the mean of three runs for CAGAN F-MNIST

Figure 43 shows the graph of the best CAGAN individual generator FID scores over the 50 generations, versus the mean of the best individuals in the three runs of the experiments for Fashion MNSIT dataset. In the last generation for CAGAN F-MNIST, the lowest FID was reported to be 79.97, with a standard deviation of $\sigma = 3.79$. When compared to the standard deviation of GAGAN, the CAGAN standard deviation is dramatically lower.

Figure 38 shows a progression of the best individual CAGAN generated images for the F-MNIST.



b) Gen 1 c) Gen 25 d) Gen 50

Figure 44 The progression of F-MNIST samples created by best CAGAN generator in generations a) 1, b) 25 and c) 50

5.8 Using CAGAN to Generate Stroke Faces

Figure 45 shows the FID score over the generations of the best CAGAN individual found in three runs for Stroke Faces. The best FID score at the last generation of CAGAN was reported to be 74.36. The mean of the top individuals of three runs is also shown in Figure 45. The standard deviation of the last generation for all the runs of CAGAN Stroke Faces is proclaimed to be $\sigma = 21.25$.

In Figure 46, CAGAN stroke face generation is shown. In first-generation, the best CAGAN individual is generating the noise, but as the training progress, the generator learns the complicated stroke face dataset and some of the generated images shows the facial stroke signs. However, there is some noise present in the final training generation of CAGAN, but the generated images are outperforming the existing benchmark models. The comparison of benchmark models is shown in the next chapter.

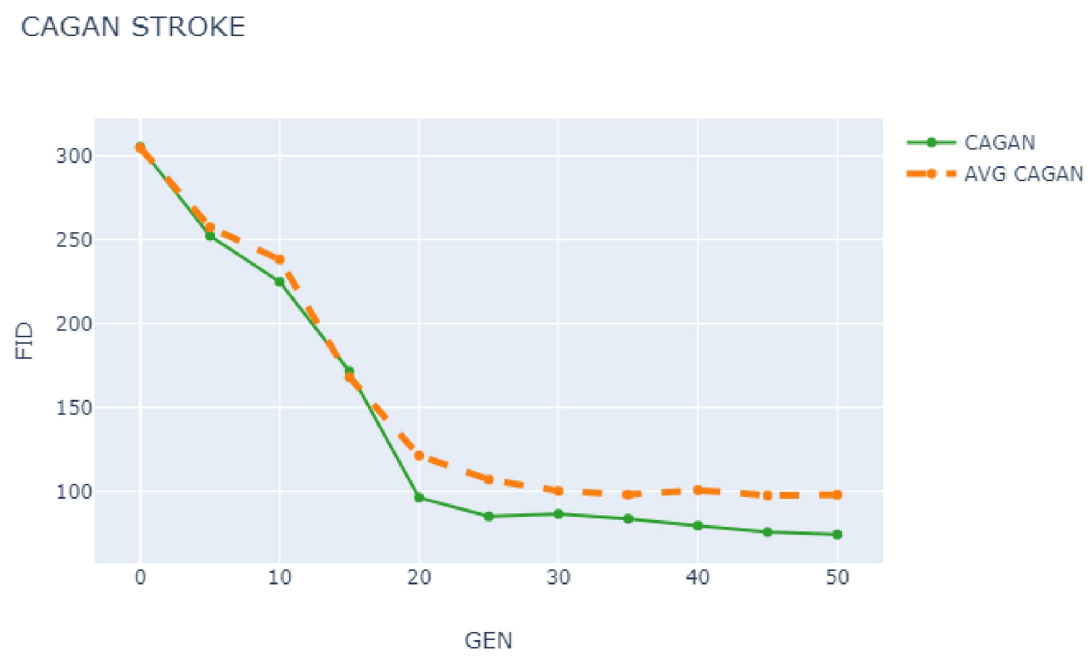


Figure 45 Graph showing the best FID score achieved at each generation versus the mean of three runs for CAGAN Stroke Faces

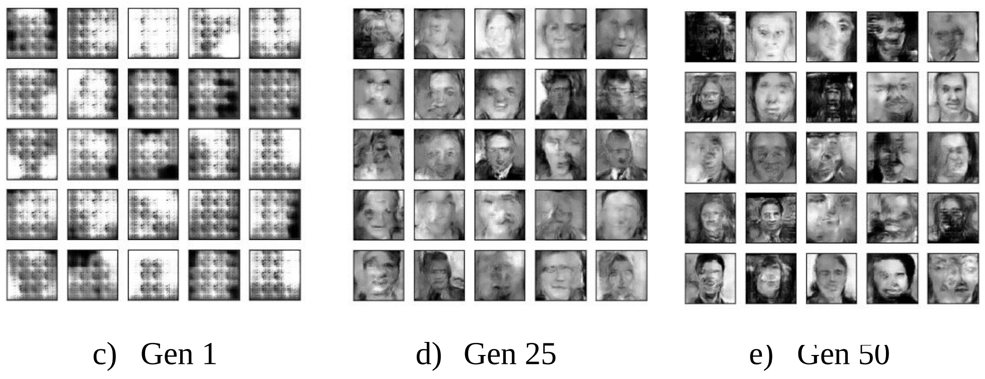


Figure 46 The progression of Stroke Face samples created by best CAGAN generator in generations a) 1, b) 25 and c) 50

5.9 Best Evolved Architecture

Table 7 represents the best architecture found by using CAGAN for MNIST after 50 generations. Both the architectures are composed of convolutional layers. The loss function for this best-evolved architecture is RAGAN loss function.

It is necessary to note that not only the final architecture is essential but also the process to construct the final models because of the mechanism of transference of the learned weights throughout generations. Therefore, we are only showing one evolved architecture.

Generator
$z \epsilon \mathbb{R}^{100}$ ~N(0,1)
ConvTranspose2d 4x4, Stride 1, Pad 0, no bias, 100→512
BN and LeakyReLU
ConvTranspose2d 3x3, Stride 1, Pad 1, no bias, 512→512
BN and ELU
ConvTranspose2d 4x4, Stride 2, Pad 1, no bias, 512→256
BN and ELU
ConvTranspose2d 3x3, Stride 1, Pad 1, no bias, 256→256
BN and ReLU
ConvTranspose2d 4x4, Stride 2, Pad 1, no bias, 256→128
BN and ELU
ConvTranspose2d 4x4, Stride 2, Pad 1, no bias, 128→64
BN and LeakyReLU
ConvTranspose2d 4x4, Stride 2, Pad 1, no bias, 64→1
Tanh

Discriminator
$x \epsilon \mathbb{R}^{1*64*64}$
Conv2d 4x4, Stride 2, Pad 1, no bias, 1→64
ReLU
Conv2d 4x4, Stride 1, Pad (1,2,21), no bias, 64→64
BN and LeakyReLU
Conv2d 4x4, Stride 2, Pad 1, no bias, 64→128
BN and LeakyReLU
Conv2d 4x4, Stride 2, Pad 1, no bias, 128→256
BN and ReLU
Conv2d 4x4, Stride 2, Pad 1, no bias, 256→512
BN and ELU
Conv2d 4x4, Stride 1, Pad 1, no bias, 512→1

Table 7 Best CAGAN evolved Generator and Discriminator Architecture for MNIST

CHAPTER 6

Comparison and Discussion

In this chapter, we have compared our proposed approach with three other different methods and analyze our results.

The study made by Lucic et al. [42] found that metric to represent better diversity and quality of generated samples when compared with real data is FID score metric. Thus, based on this study, the results are compared using FID scores.

6.1 Comparison between Vanilla GAN, DCGAN, COEGAN, GAGAN and CAGAN to generate MNIST images

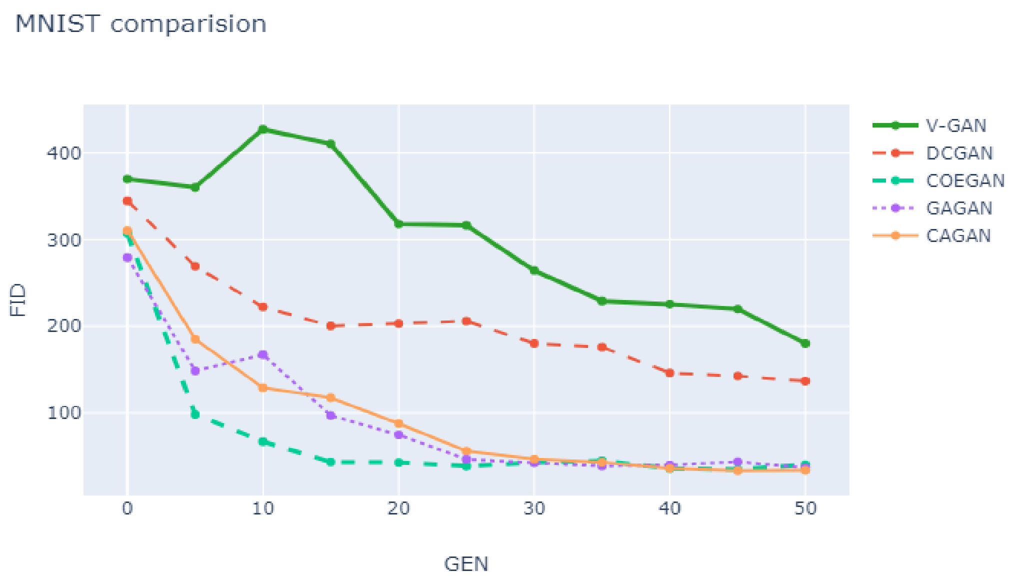


Figure 47 Comparison of Vanilla GAN, DCGAN, COEGAN, GAGAN and CAGAN for MNIST dataset

Figure 47 shows the comparison of the best FID score of Vanilla GAN, DCGAN, COEGAN, GAGAN and CAGAN. The best FID score is obtained from the three runs of each experiment. The mean FID is not shown here, because we are only interested in the best individual for generating images. The graph shows that in the low data regime, non-evolutionary methods like V-GAN and DCGAN have very high FID score compared to evolutionary methods, which means that the generated images for MNIST are not of better quality and diversity.

For COEGAN, it takes 20 generations to reach an FID score of 42 whereas for GAGAN and CAGAN it takes around 30 generations. However, at the end of 50^{th} generation CAGAN catches up with the COEGAN having the best FID score of 33.

6.2 Comparison between Vanilla GAN, DCGAN, COEGAN, GAGAN and CAGAN to generate F-MNIST images

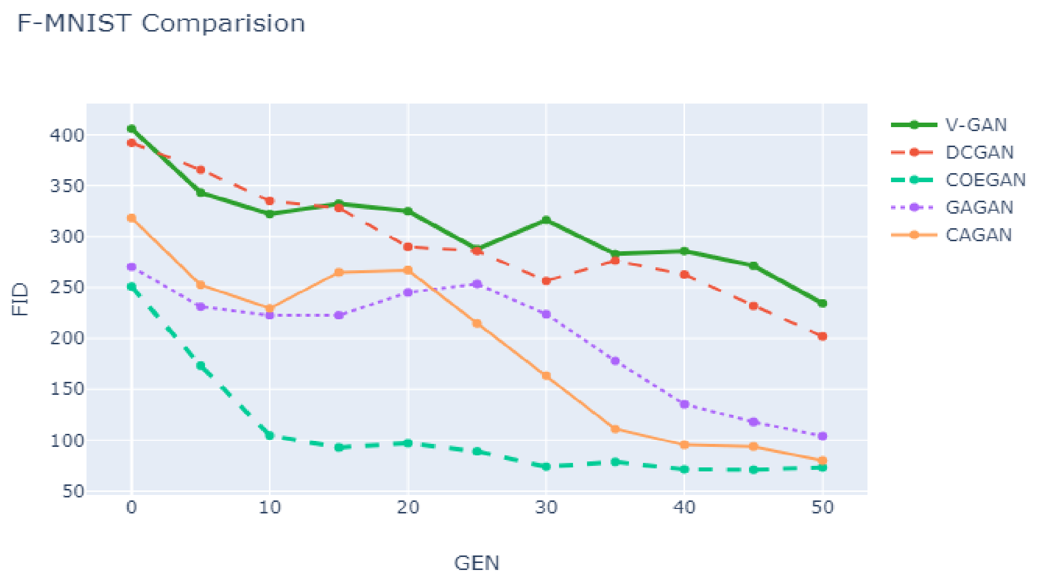


Figure 48 Comparison of Vanilla GAN, DCGAN, COEGAN, GAGAN and CAGAN for F-MNIST dataset

Best FID of the generators in Vanilla GAN, DCGAN, COEGAN, GAGAN and CAGAN are shown in Figure 48. We can see that the result of the COEGAN is better than the results of all other methods. Nonetheless, our approach CAGAN gives comparable FID score.

Because of computational limitation, we were only able to use seven individuals in our approach, whereas the COEGAN uses ten individuals in their population. With the increase in the higher GPU computational power, we can increase the population size and probably CAGAN can achieve equivalent FID number.

6.3 Comparison between Vanilla GAN, DCGAN, GAGAN and CAGAN to generate Stroke faces

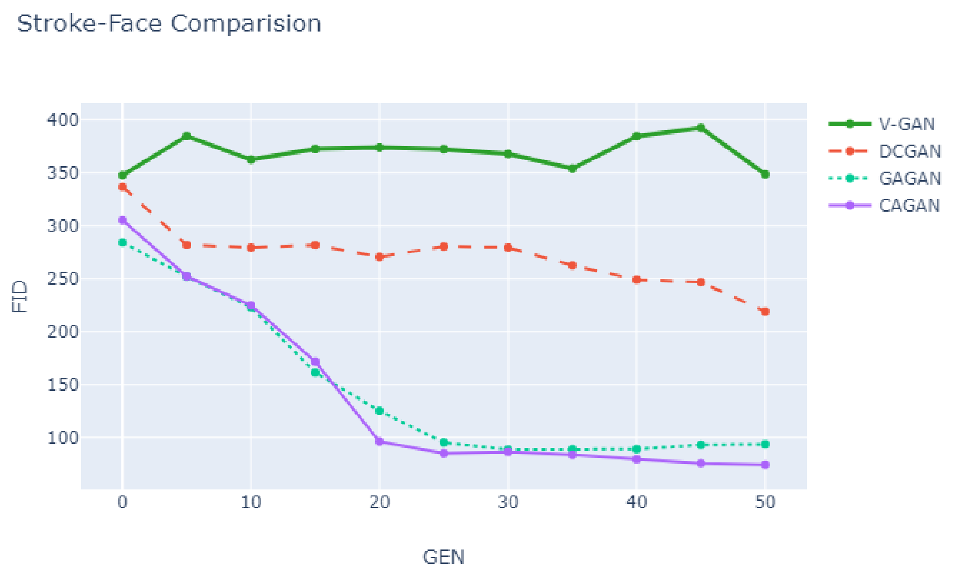


Figure 49 Comparison of Vanilla GAN, DCGAN, GAGAN and CAGAN for Stroke Face dataset

In this section, we have compared the stroke face generation with vanilla GAN, DCGAN, GAGAN and CAGAN. The COEGAN is not used here because their code repository has not given the compatibility to test it with the custom dataset. Figure 49 shows the best FID score plotting of Vanilla GAN, DCGAN, GAGAN and CAGAN. From the graph, we can see that after 25^{th} generation the FID score stagnates for GAGAN and CAGAN. The best architecture found in CAGAN outperforms all other referenced methods.

Table 8 summaries all the results of five different models for three referenced datasets used in this thesis. We can see that the CAGAN achieves best FID score of 33.37 for MNIST dataset, whereas COEGAN ranks at second best with 34.66 FID metric.
For F-MNIST dataset COEGAN exceeds all other approaches and has best FID score of 71.04, while the position of CAGAN is second with comparable FID score of 79.97.

Model	**MNIST**	**F-MNSIT**	**Stroke Face**
Vanilla GAN	180	234	348
DCGAN	113	202	218
COEGAN	34.66	**71.04**	-
GAGAN	36.61	104.02	93.16
CAGAN	**33.37**	79.97	**74.36**

Table 8 FID score comparison of five different models with three datasets

For the generation of stroke face dataset, the CAGAN is elite with FID score of 74.36, whereas the GAGAN has the second-best position with FID score of 93.16.

CHAPTER 7
Conclusion & Future Work

Generative Adversarial Networks (GANs) gained relevance for generating a synthetic dataset similar to the original images. However, training stability issues like vanishing gradient and mode collapse make the training of GAN, a hard work.

We propose two different training technique called as GAGAN and CAGAN, which uses neuroevolution with genetic algorithm and cultural algorithm respectively. The proposed methods were designed by inspiration on NEAT [8] and COEGAN [12].

In this thesis, we presented the experiments made with MNIST, F-MNIST and Stroke Face datasets to assess the efficiency of GAGAN and CAGAN in low data regime. In our experimentation, we found no evidence of mode collapse or vanishing gradient for all the three datasets. The natural evolution of the genetic and cultural algorithm contributed to preventing these issues. Moreover, the proposed method was able to generate better images when compared to referenced benchmark models. Thus, GAGAN and CAGAN presented more stable training solutions than regular GANs. We compared our results with Vanilla GAN, DCGAN and COEGAN; the result displayed that CAGAN achieved the best FID score for most of the datasets. However, COEGAN outperformed CAGAN for F-MNIST dataset.

A significant limitation of our approach is training many deep neural networks throughout the generations. Because of which proposed neuroevolutionary training approach have a high computational complexity which may turn their application unfeasible. Besides that, our proposed approach did not outperform state-of-the-art techniques, as presented in [10].

In the future work, we can extend this approach to test transferability of neuroevolved GANs from grayscale to RGB and test for higher resolution of images. We would also like to have a sensitivity analysis of the effect of different type of crossover and mutation with varying probability rate.

The proposed stroke face dataset can be pre-processed to have a higher image resolution and can be used to generate RGB stroke faces. Which further can be used to create novel approaches that more quickly and accurately recognize a stroke, particularly in counties and other settings where access to CT scans and specialized health care services is limited.

We can also expand the parameters used in the experiments in this thesis to enable the generation of a larger network. Thus, a larger population of GANs can be used with a bigger limit in the number of genes in the chromosome.

REFERENCES

[1] I. J. Goodfellow, J. . Pouget-Abadie, M. . Mirza, B. . Xu, D. . Warde-Farley, S. . Ozair, A. C. Courville and Y. . Bengio, "Generative Adversarial Nets," , 2014. [Online]. Available: https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf. [Accessed 14 4 2019].

[2] Y. LeCun, "Quora," July 2016. [Online]. Available: https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning.

[3] I. Goodfellow, "Twitter," 2018. [Online]. Available: https://twitter.com/goodfellow_ian/status/1084973596236144640?s=20.

[4] K. Shmelkov, C. Schmid and K. Alahari, "How good is my GAN?," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[5] M. Hergott, *A Leap into the Future: Generative Adversarial Networks,* Medium, 2019.

[6] X. Yi, E. Walia and P. Babyn, "Generative adversarial network in medical imaging: A review," *Medical Image Analysis,* p. 101552, 2019.

[7] A. Antoniou, A. Storkey and H. Edwards, "Augmenting image classifiers using data augmentation generative adversarial networks," in *International Conference on Artificial Neural Networks*, 2018.

[8] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation,* vol. 10, pp. 99-127, 2002.

[9] C. Wang, C. Xu, X. Yao and D. Tao, "Evolutionary generative adversarial networks," *IEEE Transactions on Evolutionary Computation,* 2019.

[10] T. Karras, T. Aila, S. Laine and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, "Improved techniques for training gans," in *Advances in neural information processing systems*, 2016.

[12] V. Costa, N. Lourenço and P. Machado, "Coevolution of Generative Adversarial Networks," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar),* 2019.

[13] Estenben, *Using Evolutionary AutoML to Discover Neural Network Architectures,* Google Brain, 2018.

[14] H. Y. Noah, "Sapiens: A brief history of humankind," *NY: HarperCollins*, 2015.

[15] G. M. Feinman and L. R. Manzanilla, Cultural evolution: Contemporary viewpoints, Springer Science & Business Media, 2000.

[16] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567,* 2017.

[17] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[18] H. Xiao, K. Rasul and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," 28 8 2017. [Online].

[19] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[20] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[21] B. Shabash and K. C. Wiese, "EvoNN: a customizable evolutionary neural network with heterogenous activation functions," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2018.

[22] F. Rosenblatt, The perceptron, a perceiving and recognizing automaton Project Para, Cornell Aeronautical Laboratory, 1957.

[23] J. Patterson and A. Gibson, Deep Learning A Practitioner's Approach, O'Reilly, 2007.

[24] D. H. Von Seggern, CRC standard curves and surfaces with mathematica, Chapman and Hall/CRC, 2016.

[25] E. Fan, "Extended tanh-function method and its applications to nonlinear equations," *Physics Letters A,* vol. 277, pp. 212-218, 2000.

[26] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10),* 2010.

[27] A. P. Engelbrecht, Computational Intelligence : an Introduction, Wiley., 2007 .

[28] M. Nielsen, Neural Networks and Deep Learning, 2015.

[29] I. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160,* 2016.

[30] A. Radford, L. Metz and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434,* 2015.

[31] N. Inkawhich, *DCGAN Tutorial,* 2018.

[32] A. Jolicoeur-Martineau, "The relativistic discriminator: a key element missing from standard GAN," *arXiv preprint arXiv:1807.00734,* 2018.

[33] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang and S. Paul Smolley, "Least squares generative adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[34] H. Zhang, I. Goodfellow, D. Metaxas and A. Odena, "Self-attention generative adversarial networks," *arXiv preprint arXiv:1805.08318,* 2018.

[35] J. H. Holland, "Genetic algorithms," *Scientific american,* vol. 267, pp. 66-73, 1992.

[36] "Tutorials Point," [Online]. Available: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm.

[37] "GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/tournament-selection-ga/.

[38] R. G. Reynolds, "An introduction to cultural algorithms," in *Proceedings of the third annual conference on evolutionary programming*, 1994.

[39] R. G. Reynolds and S. Zhu, "Knowledge-based function optimization using fuzzy cultural algorithms with evolutionary programming," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 31, pp. 1-18, 2001.

[40] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy and others, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, Elsevier, 2019, pp. 293-312.

[41] K. O. Stanley, J. Clune, J. Lehman and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence,* vol. 1, pp. 24-35, 2019.

[42] M. Lucic, K. Kurach, M. Michalski, S. Gelly and O. Bousquet, "Are gans created equal? a large-scale study," in *Advances in neural information processing systems*, 2018.

[43] U. Garciarena, R. Santana and A. Mendiburu, "Evolved GANs for generating Pareto set approximations," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018.

[44] F. a. S. A. a. Z. Y. a. S. S. a. F. T. a. X. J. Yu, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," in *arXiv preprint arXiv:1506.03365*,

2015.

[45] Z. a. L. P. a. W. X. a. T. X. Liu, "Deep Learning Face Attributes in the Wild," in *Proceedings of International Conference on Computer Vision*, 2015.

[46] T. Schmiedlechner, A. Al-Dujaili, E. Hemberg and U.-M. O'Reilly, "Towards distributed coevolutionary gans," *arXiv preprint arXiv:1807.08194*, 2018.

[47] J. a. H. E. a. O. U.-M. Toutouh, "Spatial evolutionary generative adversarial networks," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019.

[48] H.-Y. Cho and Y.-H. Kim, "Stabilized training of generative adversarial networks by a genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019.

[49] V. a. L. N. a. C. J. a. M. P. Costa, "COEGAN: evaluating the coevolution effect in generative adversarial networks," in *Proceedings of the Genetic and Evolutionary Computation Conference,* 2019.

[50] X. Gong, S. Chang, Y. Jiang and Z. Wang, "AutoGAN: Neural Architecture Search for Generative Adversarial Networks," *arXiv preprint arXiv:1908.03835,* 2019.

[51] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems*, 2017.

[52] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 421-436.

[53] D. P. a. B. J. Kingma, "Adam: A method for stochastic optimization," in *arXiv preprint arXiv:1412.6980*, 2014.

[54] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS-W*, 2017.

[55] R. Allain, Plotly Technologies Inc., [Online]. Available: https://plot.ly/~RhettAllain/412/mass-of-8-x-115-sheets-of-paper/ .

[56] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018.

[57] M. Suganuma, S. Shirakawa and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary*

Computation Conference, 2017.

[58] E. Real, A. Aggarwal, Y. Huang and Q. V. Le, "Regularized evolution for image classifier architecture search," *arXiv preprint arXiv:1802.01548,* 2018.

[59] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.

[60] P. H. Mcquesten, "Cultural enhancement of neuroevolution," 2002.

[61] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau and C. Gagné, " DEAP: Evolutionary Algorithms Made Easy," *Journal of Machine Learning Research* , vol. 13 , pp. 2171-2175, 7 2012 .

[62] P. Bontrager, W. Lin, J. Togelius and S. Risi, "Deep interactive evolution," in *International Conference on Computational Intelligence in Music, Sound, Art and Design*, 2018.

[63] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein and others, "Imagenet large scale visual recognition challenge," *International journal of computer vision,* vol. 115, pp. 211-252, 2015.

[64] D.-A. Clevert, T. Unterthiner and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289,* 2015.

[65] U. Garciarena, A. Mendiburu and R. Santana, "Towards automatic construction of multi-network models for heterogeneous multi-task learning," *arXiv preprint arXiv:1903.09171,* 2019.

[66] X. Cui, W. Zhang, Z. Tüske and M. Picheny, "Evolutionary stochastic gradient descent for optimization of deep neural networks," in *Advances in neural information processing systems,* 2018.

APPENDICES

Appendix A

All the experiments were performed using Pytorch [54] library. To develop the deep learning models for research purposes I strongly suggest using Pytorch. For plotting the graphs, I have used Plotly [55].

All the source code for this thesis is available at https://github.com/KaitavMehta95/GAN-Evolution. One may need to change the hyperparameters, activation functions, loss functions, output channels range or even the code used in above reference to better suit their dataset and experiments.

The code was executed on NVIDIA GeForceGTX 1070 GPU with dedicated GPU memory of 8 GB. Each run of the experiment took around 16 hours, the runtime of the algorithm can be improved by increasing the GPU memory.

VITA AUCTORIS

NAME:	Kaitav Mehta
PLACE OF BIRTH:	Gujarat, India
YEAR OF BIRTH:	1995
EDUCATION:	M.K secondary and higher secondary School, Ahmedabad, Gujarat, 2012
	Government Engineering College Gandhinagar, B.E., Gandhinagar, Gujarat, 2016
	University of Windsor, M.Sc., Windsor, ON, 2019