2www`

Image generation utilising a Generative Adversarial Network.

****

Honours Degree in

**Computing with Software Development**

Student:

Gavin Everett (T-00189398)

Project Supervisors:

Paul Collins

Helen Fitzgerald

# Abstract

A Generative Adversarial Network is a subset of the artificial intelligence family. The network uses unsupervised learning algorithms on two networks called the generator and the discriminator. Both the generator and discriminator networks work against each until and equilibrium is met at which point the best results have been achieved. The network can be used to generate synthetic images based off random noise inputted to the generator network.

This thesis investigated a subset of the Generative Adversarial Network called the Conditional Generative Adversarial Network to generate authentic colored images using sketches and image synthesis. The Conditional Generative Adversarial Network takes in an extra input named ‘y’ which is used to pass in a specific condition. The network was built using Python and used frameworks such as Keras and Tensorflow. Tensorflow and Keras worked together to create the model. The CUHK dataset was used to train the model which contained sketches and images. The images were pre-processed using OpenCV and Pillow. The sketch images were passed into the generator model as the condition followed by a vector of random pixels to generate an image of color. The image generated was then passed to the discriminator model where it was examined on its authenticity with that of the real training data. Backpropagation was used to improve the results of both networks.

The overall concept of generating images from a Conditional Generative Adversarial Network can be viewed as a success. The networked was trained rigorously with the generator network returning an accuracy value of 85% which can be seen as a respectable figure due to the fact that Generative Adversarial Networks are a relatively new subset to the artificial intelligence family.

# Acknowledgements

I would like to thank my supervisor Paul Collins for his support and advice throughout the year and Helen Fitzgerald who guided me along the way.

Contents

[Abstract 1](#_Toc7780625)

[Acknowledgements 2](#_Toc7780626)

[Introduction 8](#_Toc7780627)

[Chapter 2: Artificial Neural Networks 9](#_Toc7780628)

[2.1 Introduction to Artificial Neural Networks 9](#_Toc7780629)

[2.2 Biological and Artificial Comparison 10](#_Toc7780630)

[2.2.1 Biological Neuron 10](#_Toc7780631)

[1.2.2 Artificial Neuron 11](#_Toc7780632)

[2.2.3 Bias Node/Neuron 11](#_Toc7780633)

[2.2.4 Input Node 11](#_Toc7780634)

[2.2.5Hidden Node 12](#_Toc7780635)

[2.2.6 Output Node 12](#_Toc7780636)

[2.3 Back Propagation Algorithm 12](#_Toc7780637)

[2.4 Artificial Neural Network Layers – Single Layer Perceptron 12](#_Toc7780638)

[2.5 Artificial Neural Network Layers – Multilayer Perceptron 13](#_Toc7780639)

[2.5.1 Input Layer 14](#_Toc7780640)

[2.5.2 Hidden Layers 14](#_Toc7780641)

[2.5.3 Output Layer 14](#_Toc7780642)

[2.6 Training ANN 14](#_Toc7780643)

[2.6.1 Supervised Training 14](#_Toc7780644)

[2.6.2 Unsupervised Training or Adaptive Training 14](#_Toc7780645)

[2.7 Activation Functions 15](#_Toc7780646)

[2.7.1 Linear Activation function 15](#_Toc7780647)

[2.7.2 Sigmoid Activation function 15](#_Toc7780648)

[2.7.3 SoftMax Activation function 16](#_Toc7780649)

[2.7.4 Hyperbolic tangent Activation Function 16](#_Toc7780650)

[Chapter 3: Convolutional Neural Networks 18](#_Toc7780651)

[3.1. Introduction to Convolutional Neural Networks 18](#_Toc7780652)

[3.1 Convolutional Neural Network Models 18](#_Toc7780653)

[3.2 Layers 19](#_Toc7780654)

[3.2.1 Convolutional Layer 19](#_Toc7780655)

[3.2.2 Pooling Layer 19](#_Toc7780656)

[3.2.3 Fully Connected Layer 19](#_Toc7780657)

[3.3 Detecting a feature 20](#_Toc7780658)

[3.4 Overfitting 20](#_Toc7780659)

[Chapter 4: Generative Adversarial Networks 22](#_Toc7780660)

[4.1 Overview of Generative Adversarial Networks 22](#_Toc7780661)

[4.2 Networks 23](#_Toc7780662)

[4.2.1 Generative Network 23](#_Toc7780663)

[*4.2.2 Discriminative Network* 25](#_Toc7780672)

[4.3 Architecture 25](#_Toc7780673)

[4.3.1 Deep Convolutional Generative Adversarial Network 25](#_Toc7780674)

[4.4 Conditional GAN 26](#_Toc7780675)

[4.5 Image Models 27](#_Toc7780676)

[4.5.1 Parametric 27](#_Toc7780677)

[4.5.2 Non-Parametric 27](#_Toc7780678)

[4.6 Functions 27](#_Toc7780679)

[4.6.1 Loss Function 27](#_Toc7780680)

[4.7 Techniques 28](#_Toc7780681)

[4.7.1 Normalizing Inputs 28](#_Toc7780682)

[4.7.2 Optimizers 28](#_Toc7780683)

[4.7.3 Batch Normalization 29](#_Toc7780684)

[4.7.4 Train with Labels 29](#_Toc7780685)

[4.8 Image Synthesis 29](#_Toc7780686)

[4.9 Gradient Descent 29](#_Toc7780687)

[4.9.1 Steps 30](#_Toc7780688)

[4.9.2 Calculate the gradient 31](#_Toc7780689)

[4.9.3 Convergence 31](#_Toc7780690)

[4.10 Problems 31](#_Toc7780691)

[4.10.1 Mode Collapse 31](#_Toc7780692)

[4.10.2 Convergence Metric 31](#_Toc7780693)

[4.11 Wasserstein GAN 32](#_Toc7780694)

[4.11.1 Lipschitz Continuity 33](#_Toc7780695)

[4.11.2 Problems 33](#_Toc7780696)

[4.11.3 Gradient Penalty 33](#_Toc7780697)

[Chapter 5: Methodology and Design 34](#_Toc7780698)

[5.1 Key Findings 34](#_Toc7780699)

[5.2 Research Question 34](#_Toc7780700)

[5.3 Proposal 35](#_Toc7780701)

[5.4 Vision Document 35](#_Toc7780702)

[5.4.1 Problem Statement 35](#_Toc7780703)

[5.4.2 Product Position Statement 35](#_Toc7780704)

[5.4.3 Stakeholder and User Descriptions 35](#_Toc7780705)

[5.4.4 Product Overview 36](#_Toc7780706)

[5.4.5 Assumptions and Dependencies 37](#_Toc7780707)

[5.4.6 Product Features 37](#_Toc7780708)

[5.4.11 Moscow Method 37](#_Toc7780709)

[5.5 Functional Specification 39](#_Toc7780710)

[5.5.1 User Stories 39](#_Toc7780711)

[5.5.2 Risk Analysis 40](#_Toc7780712)

[5.5.3 Project Plan 41](#_Toc7780713)

[Chapter 6: Development Environment 42](#_Toc7780714)

[6.1 TensorFlow 42](#_Toc7780715)

[6.2 Keras 42](#_Toc7780716)

[7 Implementation 43](#_Toc7780717)

[7.1 Sprint 1 43](#_Toc7780718)

[7.2 Sprint 2 Data and Environment Preparation 47](#_Toc7780719)

[Task 1 47](#_Toc7780720)

[Task 2 48](#_Toc7780721)

[7.3 Sprint 3 50](#_Toc7780722)

[Task 1 50](#_Toc7780723)

[Task 2 50](#_Toc7780724)

[Task 3 53](#_Toc7780725)

[7.4 Sprint 4 55](#_Toc7780726)

[Task 1, 2, 3 55](#_Toc7780727)

[Task 4 57](#_Toc7780728)

[Task 5 60](#_Toc7780729)

[7.5 Sprint 5 62](#_Toc7780730)

[Task 1 62](#_Toc7780731)

[Task 2 63](#_Toc7780732)

[Task 3 63](#_Toc7780733)

[7.6 Sprint 6 64](#_Toc7780734)

[Task 1 64](#_Toc7780735)

[7.7 Sprint 7 68](#_Toc7780736)

[Task 1 68](#_Toc7780737)

[7.8 Sprint 8 70](#_Toc7780738)

[Task 1 70](#_Toc7780739)

[7.9 Sprint 9 71](#_Toc7780740)

[Task 1 71](#_Toc7780741)

[Task 2 72](#_Toc7780742)

[Task 3 72](#_Toc7780743)

[Task 4 73](#_Toc7780744)

[Task 5 73](#_Toc7780745)

[Task 6 78](#_Toc7780746)

[7.10 Sprint 10 79](#_Toc7780747)

[Task 1 79](#_Toc7780748)

[8 Findings and Conclusions 80](#_Toc7780749)

[8.1 Computational Performance 80](#_Toc7780750)

[8.2 Training Observations 80](#_Toc7780751)

[8.3 Research Question Answered 81](#_Toc7780752)

[Bibliography 82](#_Toc7780753)

[Appendix 85](#_Toc7780754)

Table of Figures

[Figure i Biological Neuron versus Artificial Neural Network 6](#_Toc6740002)

[Figure ii Single layer perceptron 9](#_Toc6740003)

[Figure iii Multilayer perceptron 9](#_Toc6740004)

[Figure iv Linear activation 11](#_Toc6740005)

[Figure v Sigmoid activation 12](#_Toc6740006)

[Figure vi Softmax activation function 12](#_Toc6740007)

[Figure vii Hyperbolic tangent 13](#_Toc6740008)

[Figure viii Fully connected layer 16](#_Toc6740009)

[Figure ix Overfitting model 17](#_Toc6740010)

[Figure x Generative adversarial process 18](#_Toc6740011)

[Figure xi Generative adversarial network process 19](#_Toc6740012)

[Figure xii Generative and Discriminator loss graphed 19](#_Toc6740013)

[Figure xiv Generative adversarial network training 20](#_Toc6740015)

[Figure xv Deep convolutional generative adversarial network 21](#_Toc6740016)

[Figure xvi DCGAN demonstrated 22](#_Toc6740017)

[Figure xvii Conditional GAN formula 22](#_Toc6740018)

[Figure xix Conditional generative adversarial network process 23](#_Toc6740020)

[Figure xx Generative adversarial network loss function 24](#_Toc6740021)

[Figure xxi Generative adversarial network optimizer Adam 24](#_Toc6740022)

[Figure xxii Generative adversarial network optimizer Adagrad 25](#_Toc6740023)

[Figure xxiii Gradient Descent 26](#_Toc6740024)

[Figure xxiv Calculating steps 27](#_Toc6740025)

[Figure xxv Wasserstein generative adversarial network results 28](#_Toc6740026)

[Figure xxvi Wasserstein generative adversarial network training visualisation 29](#_Toc6740027)

[Figure xxvii Computational Performance 75](#_Toc6740028)

[Figure xxviii Generator Loss 76](#_Toc6740029)

# Introduction

To implement a Conditional Generative Adversarial Network, the area of artificial neural networks must be investigated. The area of artificial neural networks has been vastly explored over the last number of years and has a great foundation to create neural networks easily. A sub category of this is the generative adversarial network. This type of network involves two networks running simultaneously one which is called the ‘generator’ and the other the ‘discriminator’. The generator network takes in a vector of latent space and converts this vector to a randomly generated image. The discriminator is used to take input from the real training dataset and the generators generated data then outputting a result of true or false. This thesis will discuss algorithms associated with artificial neural networks and generative adversarial networks to see if a sketch can be converted to a colored image.

Machine Learning is an area of artificial intelligence that involves a computer to learn from a set of data without being coded. Machine Learning is widely used in a variety of daily items which include for example when you access your phone through voice commands or facial recognition or a car that will prevent you from crashing. Some tasks that are related to machine learning would be recognition (voice, facial), predictions and also robotics.

Machine Learning is used when a program cannot be used to solve a given problem (Alpaudyn, 2010). When trying to solve a problem on a computer an algorithm is needed. Sometimes a human will not have the capabilities to develop an algorithm. An example would be identifying spam emails from a normal email. Each item is an email document with a specified amount of character. The input would be either yes if it is spam or no if it is not spam. But trying to Program an algorithm would be impossible to try and detect if the email is spam as it changes every so often. Machine learning solves this by intaking the data which in this case is the email document and creates an algorithm to differentiate between spam email and normal emails.

Since vast advancements in technology it is now easy to process large amounts of data. A lot of data devices these days are now digital and store reliable data. The application of machine learning methods to databases is called data mining (Alpaudyn, 2010).

# Chapter 2: Artificial Neural Networks

## 2.1 Introduction to Artificial Neural Networks

An Artificial Neural Network is a computing system that is inspired by animal brains. The model of an ANN is based off a biological neural network. The first implementation of and Artificial Neural Network was in 1943 by Warren S. McCulloch and Walter Pitts. Both Warren and Walter investigated first mathematical models of neurons but due a lack of technology were not able to move forward with investigation. This was the beginning of artificial neural networks. In the late 1950s/ early 1960s this is when artificial intelligence began to take off. Mark Rosenblatt an American phycologist in the area of artificial intelligence developed the first perceptron which was an algorithmically developed neural network. This was the first successful neural network developed and inspired many mathematicians and engineers to delve into the area of neural nets.

In the 1960 an American Phycologist named Bernard Widrow invented MADALINE I with students in his class. MADALINE I was the first neuro learning unit for neural networks with multiple adaptive elements (Lehr, 1993). In the 1980s Widrow and his students devised a series of uses for ADALINE and MADALINE. Applications included speech and pattern recognition, weather forecasting and adaptive controls (Lehr, 1993). ADALINE was the first commercial neural network ever created.

Another machine that was created in the mind 1980s was the Boltzmann machine. The Boltzmann machine was invented by Geoffrey Hinton. The Boltzmann was one of the first neural networks and had a simple learning algorithm that allows them to discover features that represent complex regularities in data (Hinton, 2014).

The first neural networks conference was held in San Diego, California in 1987 and speakers such as Bernard Widrow and John Hopfield who were known for their developments in the area. Some of the sessions included Optical Neurocomputers, Electrical Neurocomputers and Network Architectures. There were over 30 companies and organisations who attended the event (Anon., 1987).

# 2.2 Biological and Artificial Comparison

An artificial neuron is based off the biological neuron model. The diagram includes the difference in components between a biological neuron network and an artificial neural network,

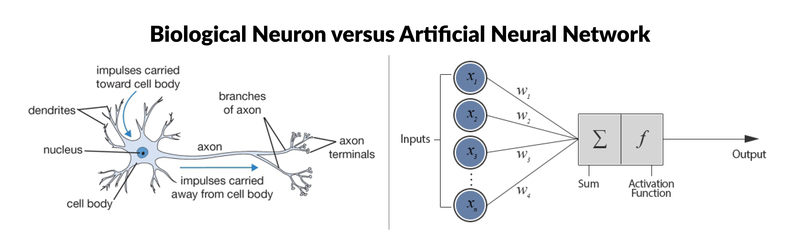


Figure i Biological Neuron versus Artificial Neural Network

Fig (Roell, 2017)

## 2.2.1 Biological Neuron

Information is received at the synapse of a neuron through its dendrites. The input is then sent through the dendrites and once it has reached the cell body it will then perform a summation of electrical impulses reaching the body and some function of this sum is performed. Once the result of the of the function is acquired if the result is greater than a particular threshold the neuron wont fire. This means it will send a signal down through the axon to communicate with other neurons (Fyfe, 2000).

The biological neural network contains of an Axon, Synapse, Soma and Dendrites which make up the process of a neuron. The process starts with the Dendrites that look like wires which accept the inputs , then The Soma will then process the input and send the output through the axon where it will be sent out the correct synapse channel , The Axon turns the processed input into output and the synapses is for contact between the neurons as visible in 2.1 one there is different stems for the synapse. The Axon can be seen and a connection wire. An electro-chemical transmission signal occurs at the synapse so that information can be shared between one neuron to another. Every neuron is connected to larger neurons through synapses and work in parallel with each other (Dave Anderson, 1992).

## 1.2.2 Artificial Neuron

The artificial neuron will recreate the same theoretical function of the biological neuron network. As shown in the Fig 2.1 above the artificial neural network neuron will take in an input or inputs through the x(n) mathematical symbol. Every input that comes in will then be multiplied by a weighted connection. The weighted connections are represented by w(n). The inputs are then fed into a specified function and create a result (Dave Anderson, 1992).There are different functions that can be used to produce different results.

**Terminology between Biological and Artificial Neural Networks.**

|  |  |
| --- | --- |
| Biological Terminology | Artificial Neural Network Terminology |
| Neuron | Node/Unit/Cell/Neurode |
| Synapse | Connection/Edge/Link |
| Synaptic Efficiency | Connection Strength/Weight |
| Firing Frequency | Node Output |
|  |  |
|  |  |

(Kishan Mehrotra, 2000)

## 2.2.3 Bias Node/Neuron

The bias node can be added to a neural network to increase learning. Bias nodes produce a value of 1 and are not attached to the previous layer. This is called bias activation.” Bias neurons allow the output of an activation function to be shifted” (Yliopisto, 2017). An example would be when result is received from an activation function (it can be any e.g. sigmoid). To change the steepness of the function just modify it. To shift the curve use bias.

## 2.2.4 Input Node

The input neuron is used as a placeholder it does not use any functions. In the input the value is weighted and summed (Yliopisto, 2017).

## 2.2.5Hidden Node

The hidden node receives inputs from other nodes. The inputs are received from nodes that come before the hidden layer. The hidden node is not connected to the output or the incoming data. The hidden node sends outputs to nodes as output or other (Yliopisto, 2017).

## 2.2.6 Output Node

The output stores the result of the node from the hidden layer. It is used to represent the final output value. The final value comes from the hidden layer were it is weighted summed and used in a function.

## 2.3 Back Propagation Algorithm

The back-propagation algorithm formulated a solution to the multi-layer perceptron that back in the 1960s was experiencing problems due to technology. Training methods were introduced for the multi-layer perceptron but they were very slow (Graupe, 2007).

In 1986 Williams, Runmelhart and Hinton provided a solution to the problems of the multi-layer perceptron. They provided a solution that would allow you to attach summed weights to inputs. This solution opened up the development of using multi-layers artificial neural networks (Graupe, 2007).

The back propagation was used widely in areas such as pattern recognition, classification and sometimes in some medical cases (Gong, 2009). The back propagation is usually used with some sort of algorithm for learning. The most popular being gradient descent algorithm (GDA). According to (Gong, 2009) when using the back-propagation algorithm with the gradient descent algorithm it would cause long training time in learning.

# 2.4 Artificial Neural Network Layers – Single Layer Perceptron

The single layer perceptron consists of a line of neurons each which produce an output. All neurons in the single layer perceptron are parallel to each other. Each neuron takes in a set of inputs. See fig below,



Figure ii Single layer perceptron

(Olivier Michel, 2018)

The single layer perceptron is very limited in the sense that if it were to be fed with large amounts of input it would not be able to classify or solve the problems which may occur. Specifically, A neuron can have binary inputs. Binary inputs can contain 2 to the power of n input patterns. Each input pattern can have at most 2 binary outputs (Graupe, 2007).

# 2.5 Artificial Neural Network Layers – Multilayer Perceptron

In the artificial neural network, the multi-layer perceptron has 3 layers which are the input layer, hidden layer and the output layer. Below is an example of what the layers connected looks like,

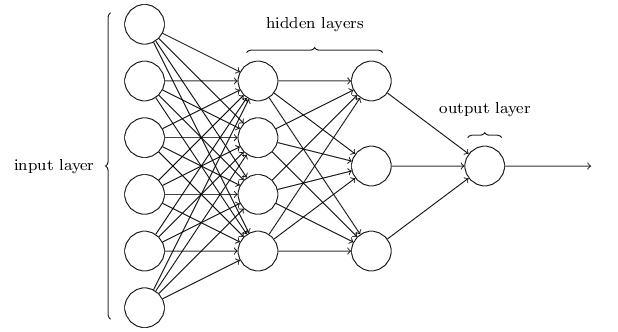


Figure iii Multilayer perceptron

(Karim, 2016)

The multi-layer perceptron is predominantly used in the area of classification of objects and images. A layer in a neural network is made from a row of neurons.

## 2.5.1 Input Layer

The function of the input layer is to just take in a piece of input from a dataset. Once the input layer receives the data, the data will be transferred to the next layer of the multi-layer perceptron.

## 2.5.2 Hidden Layers

In the hidden layers of the multi-layer perceptron this is where the inputs received from the input layer are assigned weights. After the weights are assigned the inputs are then put into a specific function and then are transferred to the output layer.

## 2.5.3 Output Layer

The output layer is the last layer in the multi-layer perceptron and it is used to produce and output or result to the program.

# 2.6 Training ANN

There are two types of training for an artificial neural network called supervised, unsupervised or adaptive training.

## 2.6.1 Supervised Training

In the supervised training method, a teacher is available for the neural network. This is where the network is supplied with inputs and has desired outputs. The teachers acts as a monitor to check for system performances and error checking. Its there to validate that output results are acceptable (Kishan Mehrotra, 2000).

## 2.6.2 Unsupervised Training or Adaptive Training

Unsupervised training is where there is no teacher involved. Here training the neural network will work independently and work off the guidance that it learns through sample data and patterns. Unsupervised training is the exact opposite of Supervised it will be provided with the inputs but there won’t be a desired output due to not having a teacher (R.Sathya, 2013).

# 2.7 Activation Functions

Activation functions are functions that deal with the output nodes. There are over several different activations functions a neural network can use. When creating a neural network, it is important specify which activations function will be used as this will affect how input is formatted. The activation functions produce an output of yes or no but in this case, they use numeric between -1 and 1.

## 2.7.1 Linear Activation function

The Linear activation function is the most basic activation function.



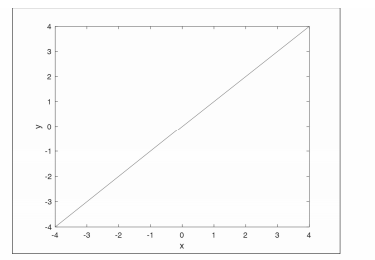
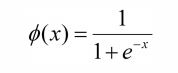


Figure iv Linear activation

## 2.7.2 Sigmoid Activation function

The sigmoid activation function is a commonly used function in neural networks. The sigmoid only outputs positive values only. The sigmoid equation is shown below (Yliopisto, 2017) ,



The values for the sigmoid function range between 0 and 1. Below is a diagram of what the sigmoid looks like,

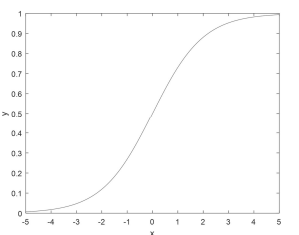


Figure v Sigmoid activation

## 2.7.3 SoftMax Activation function

Another function that is often used is the SoftMax activation function. Like all of the other activations functions the SoftMax is found at the output layer of the multi-layer perceptron. In the SoftMax function it states that the node with the highest value will get the input as a member of its class. It will then force the output of the neural network to represent the probability that the input falls into each class. “Without the SoftMax, the node’s outputs are simply numeric values, with the greatest indicating the winning class.” (Yliopisto, 2017)



Figure vi Softmax activation function

## 2.7.4 Hyperbolic tangent Activation Function

According to(Yliopisto, 2017)the hyperbolic tangent function is one of the most important activation functions. Its range is between -1 and 1 in comparison to the sigmoid function which is between 0 and 1.



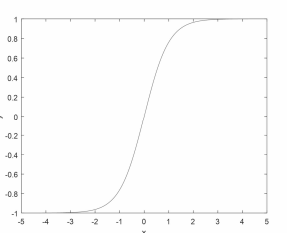


Figure vii Hyperbolic tangent

# Chapter 3: Convolutional Neural Networks

## 3.1. Introduction to Convolutional Neural Networks

A Convolutional Neural network is a Deep Learning algorithm which takes in an image as an input and applies weights and biases to it to be able to differentiate one from another (Saha, 2018).

A study conducted by (Honglak Lee, 2011) where they selected 15 to 30 images from different classes and datasets. The results report that was produced was averaged over 10 trials. When combining the first and second layers they achieved 57.7% using 15 images per class and 65.4% test accuracy using 30 training images per class on the Caltech-101 object classification task. This shows that the more that the more the neural net is trained the more likely the test accuracy is to increase (Honglak Lee, 2011). The group also did a test error average over a range of 10 random training sets and got a return value of only 0.8% which is miniscule. In another example of the Caltech-101 object classification task a Norwegian group used better a better kernel matrix for their neural net and ended with a test accuracy of 77.7% when using 30 images per class. Another study conducted by (R. Vaillant, 1994) where they used 28 volunteers of both genders to take a photo of their face. People with glasses were requested to take them off due to highlights that may appear in images and affect the machine learning process. The size of the window chosen for the neural net was a 20x20 pixel. It was this size as this is the closest at which you can distinguish between a human and non-human face. The database contained 1792 of patches with different faces and these were passed through. 7 different neural networks were tested. After testing the images through 2 neural nets the results were equivalent. On the test set, the quadratic error decreased quickly and the rate of recognition increased to an outstanding 96%.

## 3.1 Convolutional Neural Network Models

LeNet-5 was a convolutional neural network designed for hand written and machine printed character recognition (Cun, 2010). A study conducted by (Larissa Ferreira Rodrigues, 2017) on HEp-2 cells cited that (Anon., 2017)proposed a method for automatic classification of HEp-2 cells. The Convolutional Neural net that was used shared the same architecture of LeNEt-5. It was said that during experimentation that during an instance of processing one image the net achieved an accuracy of 88.58%, which increased to 96.76% with the increase of dataset with multiple copies of the image and also when the images were rotated. The best results for this method were achieve with the GoogleNET architecture resulting in a 98.17% accuracy which is the best performance ever recorded in literature (Larissa Ferreira Rodrigues, 2017). From a dataset of 13,596 images of HeP-2 cells each cell was centred. The best accuracy value returned was GoogLeNet which returned 95.03%.

## 3.2 Layers

The 3 layers that make up the Convolutional Neural Network Architecture are The Convolutional Layer, Pooling Layer and Fully Connected Layer.

3.2.1 Convolutional Layer

The convolutional layer is usually composed of 7+ feature maps with different weight vectors so that multiple features can be extracted at each location (Yan Le Cun, n.d.). Each feature map is made from units that are organized into planes. Every unit in the feature map takes inputs from a small region of the image at state and the units in the featured map are constrained to have the same weight values (Bishop, 2006). An example of a feature map is if it had 100 units that are arranged in a 10x10 grid and each unit taking inputs for a 5x5 region of the image. The feature map would then have 25 adjustable parameters plus a bias parameter.

3.2.2 Pooling Layer

The pooling layer takes in the results the come from the Convolutional Layer. The Pooling layer will perform down sampling on the given input to reduce the spatial size of the input and reduce the amount of parameters associated with it. The Pooling layer has 3 types which include Max, Average and Sum (Stutz, 2014). The Map function is most commonly used to reduce the overall size of a feature map. A window of size 2x2 is placed on top of a feature map. The max number from each 2x2 window will be taken and this in return will reduce the spatial size by approximately 25% and keeps the depth of the map (O’Shea, 2015).

3.2.3 Fully Connected Layer

The Fully Connected Layer is usually at the end of the architecture. In the Fully connected layer each unit/neuron in the layer is connected to that units of the previous layer. The goal of the fully connected layer is to classify the input based on features from the previous convolutional and pooling layers (Aghdam, 2017). This can be then passed to the SoftMax activation function for probability.

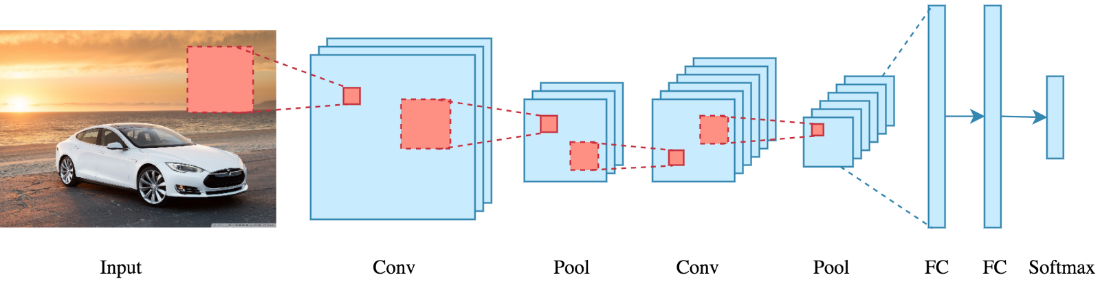


Figure viii Fully connected layer

(Dertat, 2017)

## 3.3 Detecting a feature

When a feature is detected on an image its location is not recorded but its approximate location relative to the other features is saved. After the convolutional layer the next layer performs averaging and subsampling reducing the overall size of the feature map. (Yan Le Cun, n.d.)

3.4 Overfitting

Overfitting is where a model tries to adapt itself too much to the data that is specified. For example, Feeding the network with different images of same breed of dog, if using a complex model with lots of neurons that wants to distinguish if it's a dog in the picture. Once it learns that breed of dog If the model sees a different breed of dog it may run into difficulties identifying it.

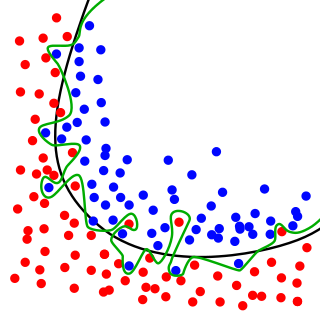


Figure ix Overfitting model

(elitedatascience, 2017)

In the diagram above input A is represented by the red dots for simplicity lets says they are pictures of cats. Then the blue dots on the other side are dogs. The green line and black line can be represented as machine learning models. The green line is following the data and always knows if it is a cat or a dog and the black line is sometimes wrong. The black line is more general if a new input of a dog or cat was inputted and wasn't a part of the training data the black line will have a better chance of getting the right answer because the green line follows the training data. The black line is an example of a line that is more general and doesn't suffer from overfitting.

# Chapter 4: Generative Adversarial Networks

## 4.1 Overview of Generative Adversarial Networks

Generative Adversarial Networks also known as (GANS) is a class of artificial intelligence using unsupervised learning algorithms and incorporates two neural networks called discriminator and generator that battle against each until an agreement is met. GANS were founded in 2014 by Ian Goodfellow (Antonia Creswell, 2018). The network is used to create a photorealistic image from inputs to the network through image synthesis. The generative network is known as the “forger” network in the process. The network generates images based off feedback it receives from the discriminator and must learn. The discriminator network has access to both sets of data. It has access to the real images that are inputs and the images created by the generator. Both models keep going back and forth until a point is reached where the images created by the generator are no longer distinguishable to the genuine input images (Antonia Creswell, 2018). The networks play a min-max game where one of the networks wants the number to be high and the other want it to be low. The number is coming from the error rate of the discriminator. The generator is satisfied with a high error rate whereas the discriminator would like a low error rate.

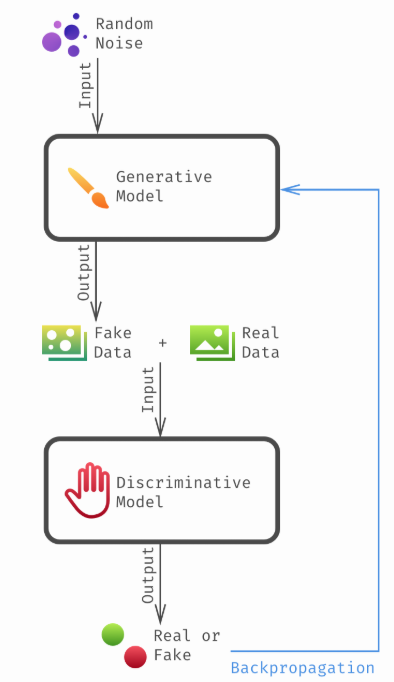


Figure x Generative adversarial process

(Ricard, 2017)

## 4.2 Networks

### 4.2.1 Generative Network

The generative model is known as the forger in a generative adversarial network. The Generative Network takes in random noise and uses to try and create sample data which can be sent to the discriminator.

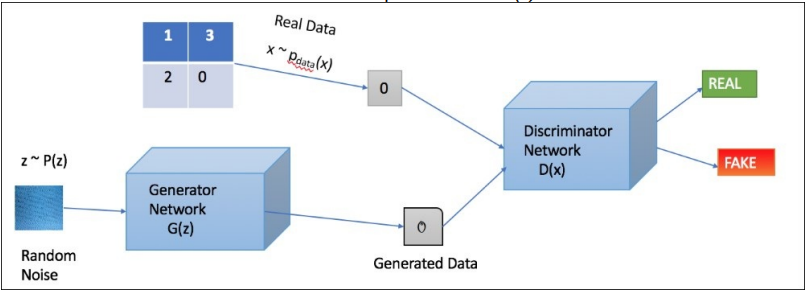


Figure xi Generative adversarial network process

(Ganguly, 2017)

In the above it is the visible to see that the generator noted as **G(z)** takes in input from **Noise P(z).** This generates the data and then is fed to the discriminator network noted **D(x).**

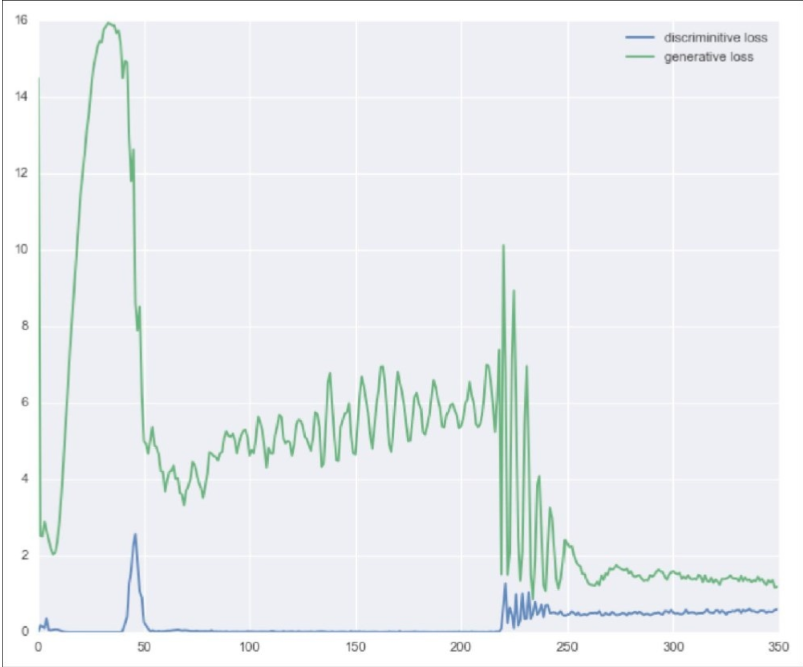


Figure xii Generative and Discriminator loss graphed

In above Figure xiii is a graph to plot the loss between both network’s discriminator and generator. It is said that eventually both networks will reach a balanced state, but this can’t be guaranteed and both networks can end up learning for a long period of time.

In simple terms the generator network will need some sort of input in order to generate an image sample. G can be seen as the generator network and Z can be seen as the input that generates images G(z).

A probability distribution will need to be assigned at the start of the neural network and then used as a condition on another probability distribution in order to generate images. The distribution used will be a training dataset. The probability distribution will never change once it is set at the beginning and once you feed an input to the generator a vector is sampled, and that vector will be fed a random number of pixels.

The most common choice of for probability distribution is to use the zero centred unit variance Gaussian distribution.

The zero cantered Gaussian means that it will be easy to pick samples.

The unit variance is in reference to each element in a specific vector that might represent a feature in an image. Examples may include (fingers, nails). (Goodfellow, 2014)

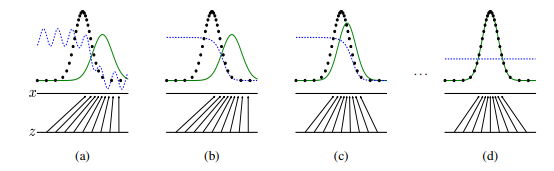


Figure xiv Generative adversarial network training

In figure xivthe process of the of training the networks is occurring. The ***green*** line can be identified as the generative distribution, the ***blue*** dashed line can be seen as the discriminator distribution so that it can discriminate between samples from the data generating distribution i.e. a dataset (the **black** dotted line). and those of the generator distribution. The first horizontal line is where **Z** is sampled. From the above over time the generative distribution (***green*** line) becomes more intelligent and when it gets to (d) it reaches equilibrium where the generated data is identical to the data distribution. (Goodfellow, 2014)

4.2.1.1 Deconvolution

The deconvolution layer is applied in GANS where images are created. The layer is applied in the generator network and is used to up sample the quality of images to try and match the input values of the dataset. It expands images from features.

#### 4.2.2 Discriminative Network

The Discriminator Network takes in inputs from both the Real data and the generator network and its objective is to identify if the input received is authentic or forged. The network takes input x from real data and then solves a ***binary classification*** problem with the output in the range of 0 to 1.

## 4.3 Architecture

### 4.3.1 Deep Convolutional Generative Adversarial Network

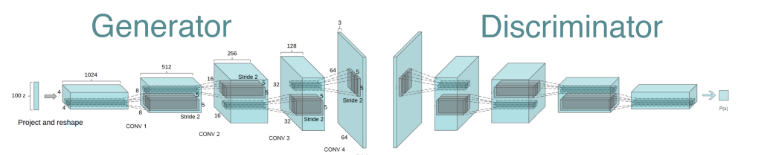


Figure xv Deep convolutional generative adversarial network

The generator neural network uses a 100-dimensional distribution space and **Z** which will be divided into smaller segments through convolutional operations.

#### 4.3.1.1 Guidelines

* Replace pooling layers with striped convolutions in the discriminator and fractional strided convolutions in the generator network.
* In both neural network’s generator and discriminator use batch norm.
* Use the ReLu activation function in all layers of the generator neural network except in the final layer where ***tanh*** function is used.
* Use ReLu leaky activation function in all layers of the discriminator network.
* Remove all fully connected layers and use average pooling at the end of deep architectures.

(Ganguly, 2017)

#### 4.3.1.2 DCGAN Demonstrated

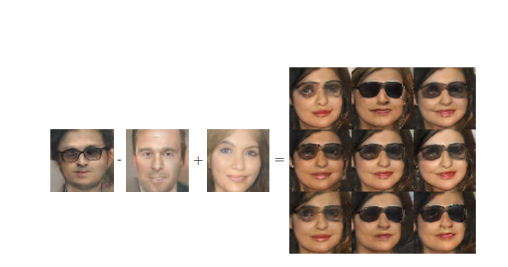


Figure xvi DCGAN demonstrated

(Goodfellow, 2016)

## 4.4 Conditional GAN

A conditional GAN is an extension to the base of the original GAN concept. The conditional GAN uses characteristics supplied by the user instead of random noise of the original GAN concept (Gauthier, 2015). The equation for the conditional GAN can be seen below,



Figure xvii Conditional GAN formula

In figure xviii the formula is the same as the original GAN with an extra parameter **y** which is used for the user to supply a certain characteristic/condition. The value of **y** is a piece of information which is sampled from the training data that is supplied. This information is then passed to the generator network for the creation of images.

The Generator model now takes in G(z , y) , **z** represents the noise data along with the condition **y** and produces an image.

The Discriminator model accepts an image **x** and condition **y** and predicts the probability that the under the condition of **y** the image **x** came from the data set rather than the generator model (Gauthier, 2015).

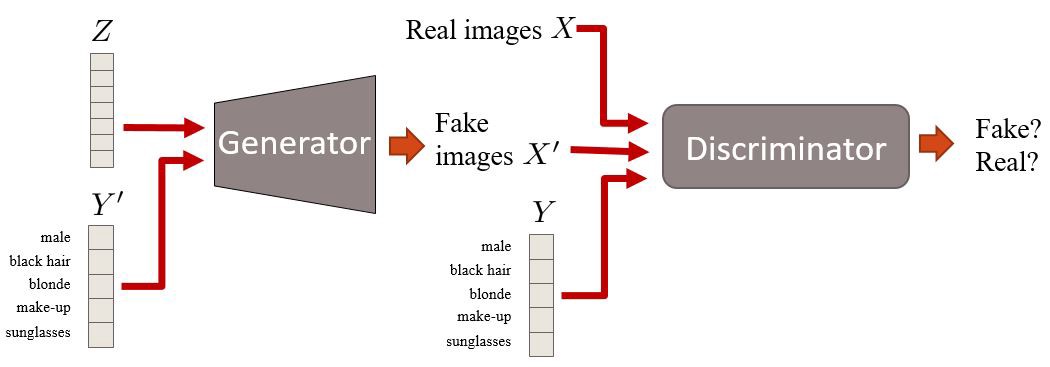


Figure xix Conditional generative adversarial network process

(Radhakrishnan, 2018)

## 4.5 Image Models

Generative image models fall into two categories parametric and non-parametric (Alee Radford & Luke Metz, 2016).

### 4.5.1 Parametric

Parametric Models are used for generating images and have been explored a lot over the last number of years. Many approaches over the past couple of years have suffered for from generated objects being too wobbly, looking distorted and blurry. A study conducted by (Denton, 2015) who used the Laplacian Pyramid in conjunction with Generative Adversarial Networks seen improved results of high-resolution images.

### 4.5.2 Non-Parametric

The non-parametric models usually do match from a database of images and is used to match patches of images and has been used in texture synthesis.

## 4.6 Functions

### 4.6.1 Loss Function

The loss function takes is used to identify the loss in the network. In the formula below, we can let (**X)** equal to the dataset otherwise known as the real data. We can let **(Z)** equal to the input noise which will be fed to **(G)** generator). **D(x)** is the formula used for the probability that the data came for the real dataset **(X). (D)** which is the discriminator network is trained to max the probability of  and train **(G)** to minimise.

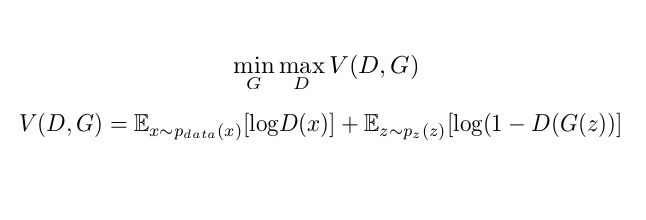


Figure xx Generative adversarial network loss function

(Goodfellow, 2014)

## 4.7 Techniques

Some training approaches have been outlined by (Ganguly, 2017) in order to overcome specific difficulties when training GANs.

### 4.7.1 Normalizing Inputs

It is good practice to normalize the inputs between a range from -1 to 1 using the tanh function at the last layer of the generator (Ganguly, 2017).

### 4.7.2 Optimizers

There are many different optimizers available to use for the optimization of the network. The main optimizers of choice are Adam and Adagrad. According to a journal by (Mohak Srivastava, 2018) in which an experiment was carried out to compare both the Adam and Adagrad optimizers it was shown that the Adam optimizer has better results where the generator network obtained a less loss result whereas with the Adagrad optimizer the loss obtained was greater for the generator network which in return shows that by using the Adam optimizer the generator is better at deceiving the discriminator which will lead to better results.



Figure xxi Generative adversarial network optimizer Adam

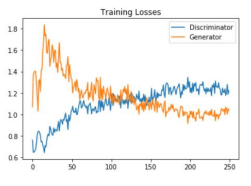


Figure xxii Generative adversarial network optimizer Adagrad

From Figure xxii you can see the generator loss drops dramatically whereas in figure xxithe generators loss gradually drops overtime.

### 4.7.3 Batch Normalization

The idea of batch normalization is to create mini-batches of real and fake images, each mini-batch should either contain all real images or all fake images. With batch normalisation it is said to increase computational efficiency.

### 4.7.4 Train with Labels

It is said to better to train the networks with labels for the networks as it improves optimization and provides better results (Goodfellow, 2016). A study carried out by (Denton, 2015) who built and CGAN retrieved better sampled images when training with labels. There is no set reason as to why training with labels increases performance. It may be due to giving class information to the training process that improves optimisation.

## 4.8 Image Synthesis

Image Synthesis is the process of generating the image from inputs given. These inputs can be given from noise. Synthetic images are often used to verify the correctness of an operator by comparing them to a known image (Huang, 2018).

## 4.9 Gradient Descent

The Gradient Descent algorithm is used to train the network. In Generative Adversarial Networks gradient descent will be used to train both the generator and discriminator networks respectively. Below is a diagram of gradient descent,

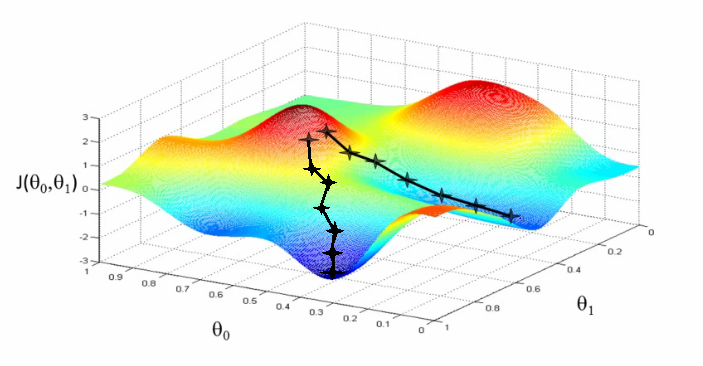


Figure xxiii Gradient Descent

(Prada, 2018)

Gradient descent works by continually tweaking the parameters of both networks in a descent fashion until it reaches a point of convergence(local) minimum. The goal is to minimize the loss function therefore leading to an increase in accuracy in overall results of the model which happens through tweaking parameters of both networks.

### 4.9.1 Steps

The Gradient descent step can be identified with the following equation,



(Prada, 2018)

Θ⁰ marks the current position and Θ¹ is the new position.

The α represents the learning rate. The learning rate is used so that steps taken are of a substantial size. Steps too large may cause a disruption to the (local) minimum whereas steps too small will lead for a long time for the algorithm to converge (Qiao, 2016). The steps taken should be taken in the opposite direction of the gradient. The step for the function can be defined as J(Θ) equal to −α∇J(Θ). There the new position can be calculated by subtracting the current position from the new value. Below is an example of a learning rate,

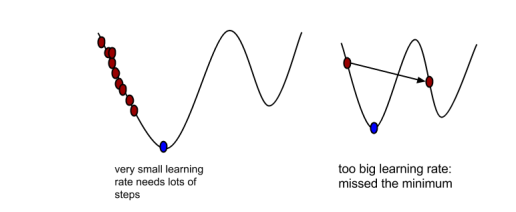


Figure xxiv Calculating steps

(Prada, 2018)

From the graph on the left, the red dots represents small steps taken which will result in many steps and long learning time for the algorithm to converge. The right-hand graph represents massive steps taken which in return will overshoot the local minimum and miss it. The blue dot represents equilibrium i.e. the best learning rate.

### 4.9.2 Calculate the gradient

#### 4.9.2.1 Cost Function

The cost functions curve can be used to update parameters of the model to make it more accurate. The cost function can be used to obtain estimates which in return can be used to identify which is the correct direction for optimal parameters.

### 4.9.3 Convergence

Convergence happens at the point at which the best set of parameters have been found. At convergence the model has been successfully trained.

## 4.10 Problems

The original GAN architecture has many shortcomings and has many stabilizing problems including the following,

### 4.10.1 Mode Collapse

Mode Collapse is where the generator collapses into a narrow distribution and the image samples created from the model are not diverse (Thanh-Tung, 2018).

### 4.10.2 Convergence Metric

There is no well-defined metric to tell about convergence between the generator’s loss and the discriminators loss.

## 4.11 Wasserstein GAN

The Wasserstein GAN can be used to overcome some of the shortcomings of the original GAN architecture. The original GAN concept is known to not stabilize well to the distribution supplied and results in non-convergence. An experiment carried out by (Arjovsky, 2017) the creator of the Wasserstein GAN ran experiments on image creation using the Wasserstein concept and showed improved benefits including,

* Improved stability of the optimization process.
* Creates a loss metric that correlates with generator loss to improve the samples quality.

With the use of a Wasserstein it takes away the assumptions of the human brain on which models are doing better than others. The Wasserstein overall provides better stability throughout the network. It allows to train the **critic** until optimality. When the **critic** is trained to completion it provides a loss the generator network. The better the **critic** the better the quality of gradients that will be produced to train the generator (Arjovsky, 2017).



Figure xxv Wasserstein generative adversarial network results

(Arjovsky, 2017)

In Figure xxv both segments left and right are trained with a DCGAN architecture. The left segment of the image is trained WGAN algorithm and the right segment is trained with the standard GAN formula. Both algorithms produce high quality samples.

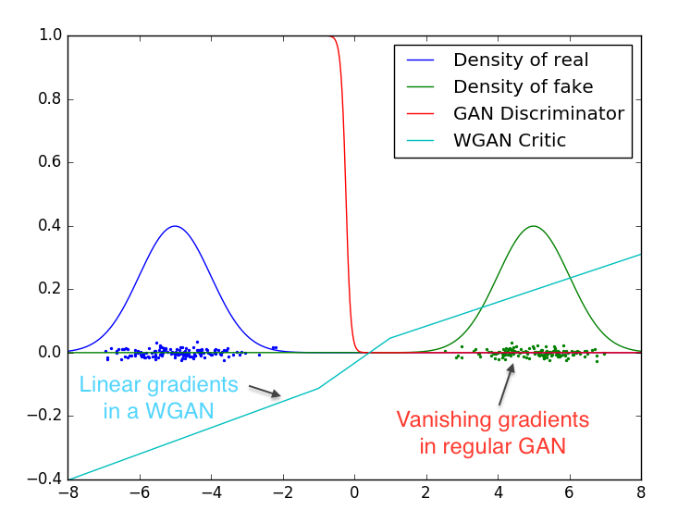


Figure xxvi Wasserstein generative adversarial network training visualisation

In the above Figure xxviwe can see theempirical results returned after experiment completion. From the above we can see with the discriminator (**red** line) makes the gradients vanish over the space over a period whereas with the WGAN gives a good looking gradient over the space.

### 4.11.1 Lipschitz Continuity

The Lipschitz constraint is required for the critic network.

### 4.11.2 Problems

The Wasserstein GAN also has its problems sometimes causing abnormal behaviours like generating poor image samples. This can arise due to **weight clipping** which is a terrible way to implement the architecture. If the clipping parameter is large it will take longer to train to critic to optimality. On the other hand, if the clipping parameter is small it can lead to vanishing gradients.

### 4.11.3 Gradient Penalty

The Gradient Penalty WGAN was introduced by (Gulrajani, 2017) an was used to overcome some of the shortcomings of the original WGAN concept. The WGAN-GP proposed a way that instead of clipping would penalize the norm of the gradient with respect to the input.

# Chapter 5: Methodology and Design

## 5.1 Key Findings

The chapters researched have provided an understanding of the key principles and concepts on how to develop an artificial neural network. In the artificial neural networks chapter an overview is given of all the components/nodes. The chapter also indicates the different layers involved in network including the input, hidden and output layers. The different functions that can be used, in particular the sigmoid function which is able to compress the output, were reviewed

Research undertaken in the convolutional neural networks reflected the history of the network and also training the different models which include GoogleNET . Also, an extensive review of the layers including convolutional, pooling and ReLu was carried.

In the generative adversarial networks chapter an overview is provided on how to develop a GAN with the required components. A detailed description of how the generator network works by feeding it a random input vector for a training set distribution which is then fed to the generator to output a sample image. For the discriminator it was shown how it can distinguish between real and fake images. A deeper review was also show in how to optimize the networks loss function after iterations of the training process and how gradient descent and backpropagation can be used to increase the learning rate of the model. The chapter also assessed other models of the Generative Adversarial Network including the Wasserstein GAN which can be used to improve the stability of the network and the Conditional GAN which can be used to supply certain conditions to the generator for the generation of an image sample.

An investigation of some of the current libraries and technologies provided a detailed description of their advantages in the development environment.

## 5.2 Research Question

To generate images through image synthesis using generative Adversarial networks

## 5.3 Proposal

The aim of this project is to create an application that can be used for graphic designers creating mockups. The application will take in a sketch of an object and generate a real image based off the input sketch. The application will be created using python, Tensorflow and Keras. The interface will be created in python and will allow the user to insert a collection of images, the images will then be sent into the discriminator neural network where it will be compared with the generator neural network. After a number of iteration it shall be able to output a real life image.

## 5.4 Vision Document

### 5.4.1 Problem Statement

**The Problem of**: designers not being able to create an accurate enough drawing of the specified object to team.

**Affects:** Affects

**The Impact of which is:**

**A Successful Solution would be:**Create a windows application to generate images from sketches.

### 5.4.2 Product Position Statement

**For the:** any designers who want to improve productivity under tight timelines.

**Who:**wants to be able to identify the health status of his/her cattle and to be able to increase the production quality of meats on his/her farm.

**Sketch:** is adesign application that will enhance the technologies used in design to create a better environment.

### 5.4.3 Stakeholder and User Descriptions

 Stakeholder Summary

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Responsibilities** |
| Project Manager | This stakeholder leads the development of Sketch Windows Application. | Creates Plans, gathers resources. |
| Software Engineer | This stakeholder is the primary lead in the development of Sketch. | Creates design of application and implements design. |

User Summary

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Stakeholder** |
| Graphic Designer | This user will be dealing with the  End product. | Graphic Designer |

Stakeholder Profiles

**Graphic Designer**

|  |  |
| --- | --- |
| **Description** | A Graphic designer who wants to generate realistic images on the fly. |
| **Type** | This is a casual user who wants to generate realistic images from sketches. |
| **Responsibilities** | Graphic Designer sketches an object and neural network will return a realistic image. |
| **Success Criteria** | The success is the user's continuous use and satisfaction of the mobile application |

Key Stakeholder and User Needs

|  |  |  |  |
| --- | --- | --- | --- |
| **Need** | **Priority** | **Concerns** | **Solution** |
| User Friendly | High | Allows graphic designer to quickly receive realistic images. | Create application with minimum amount of clutter and have a directed path through system. |
| Produce quick fast and accurate results. | Medium | Allow for multiple sketches to be passed through. | Minimise the amount of background processes in application. |
| Low Cost | Low | Graphic Designers that don’t have enough disposable income | Creating a free application with addon features that will cost extra. |

### 5.4.4 Product Overview

Product perspective:

The product will be running on an windows-based system and will be an application.

Summary of capabilities

|  |  |
| --- | --- |
| **Customer benefit** | **Supporting features** |
| Generate realistic images easily. | The application will take in the users sketch and generate/produce a realistic image. |
| Low cost. | The application will be free on the windows app store. |
|  |  |
|  |  |
|  |  |

### 5.4.5 Assumptions and Dependencies

* It is assumed that the user wanting to download the application will be on the windows platform.
* It is assumed that the user will have previous experience with the windows platform.

### 5.4.6 Product Features

Feature 1

* Open Application

Feature 2

* Sketch Design

Feature 3

* Press generate button

Feature 4

* Result to show realistic image options.

### 5.4.11 Moscow Method

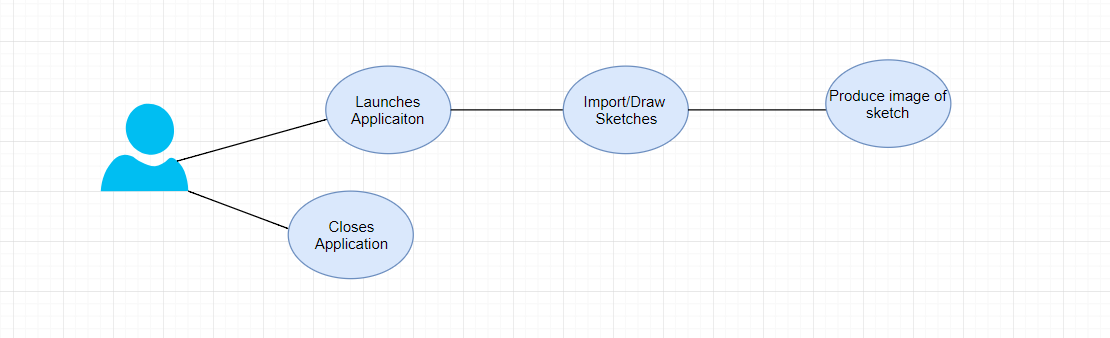
|  |  |
| --- | --- |
| MoSCoW Method | |
| 1. Must Have | 1. Could Have |
| 1. Should Have | 1. Won’t Have |

|  |  |  |
| --- | --- | --- |
| **ID Number** | **Description** | **MoSCoW Priority** |
| 001 | Set up environment in PyCharm IDE. | 1 |
| 002 | Install packages for Python. | 1 |
| 003 | Install Kera's and TensorFlow packages of PyCharm. | 1 |
| 004 | Test out environment by creating test neural network | 3 |
| 005 | Look for available datasets online. | 1 |
| 006 | Import dataset images into development environment. | 1 |
| 007 | Train the neural network to analyse the images. | 1 |
| 008 | Train the neural network to analyse images in different rotations. | 2 |
| 009 | Analyse neural network results. | 2 |
| 010 | Determine if neural network classification is accurate | 2 |
| 011 | Test network with different types of datasets | 4 |
| 012 | Design application in Python | 3 |
| 013 | Develop application in Python | 1 |
| 014 | Permission logic to access front facing camera. | 1 |
| 015 | Permission logic to access photo album | 1 |
| 016 | Test application on android device/emulator. | 1 |
| 017 | Create application on Windows/IOS | 4 |
|  |  |  |
|  |  |  |

## 5.5 Functional Specification

### 5.5.1 User Stories

#### 5.5.1.1 Use Case Diagram



|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **Walkthrough** | |
| **Use Case Id** | 0001 | |
| **Priority** | High | |
| **Source** |  | |
| **Primary Business Actor** | User | |
| **Other Participating Actors** |  | |
| **Description** | The User will open the application and either upload a collection of sketches or draw a sketch and an image will be returned of the object. | |
| **Preconditions** |  | |
| **Trigger** |  | |
| **Typical Scenario** | **Actor Action** | **System Response** |
|  | **Step 1:** The user will launch the Sketch application.  **Step 4:** The user will enter their data/sketches. | **Step 2:** The application will load and a UI will be displayed.  **Step 3:** The application will have been trained on some common objects.  **Step 5:** The application will then take that data and pass it to the neural network.  **Step 6:** The application will then display a result/image. |
| **Alternate Scenarios** | **Actor Action** | **System Response** |
|  |  |  |
| **Conclusions** | The Neural Network will convert sketch to image. | |
| **Post conditions** |  | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

### 5.5.2 Risk Analysis

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | Risk Source | Probability  Low Med  High  1-3   4-7   8-10 | Impact  Low Med  High  1-3   4-7   8-10 | **Result**  Prob. X  Impact | **Impact Areas**    Cost Sch.Perf. | Response |
| 1 | Installing Python | 3 | 10 | 30 |  | Watch tutorial on best python version for neural networks. |
| 2 | Installing Tensorflow and Keras libraries in Python | 6 | 10 | 60 |  | Make sure version of Python is compatible with Python version. |
| 3 | Installing addition packages to Python | 2 | 5 | 10 |  | Look into pip tools to install extra libraries. |
| 4 | Finding a dataset with cows experiencing stress | 7 | 10 | 70 |  | Try and look in websites that are specifically for datasets. |
| 5 | Not having capable hardware to run neural network | 3 | 8 | 24 |  | Identify PC in college with good specification. |
| 6 | Neural Network Accuracy | 7 | 7 | 49 |  | Train neural network with lots of data even augmented. |
| 7 | Finding hard to understand Tensorflow and Keras. | 5 | 7 | 35 |  | Watch videos or take out books from library. |
| 8 | Finding it hard to understand Python Syntax | 4 | 8 | 32 |  | Look into Python tutorials and old notes. |
| 9 | Finding it hard to create a neural network. | 6 | 10 | 60 |  | Re-Create a test neural network from start to finish. |

### 5.5.3 Project Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Prototype** | **Start Date** | **End Date** | **Hours** |
| Choose and Neural Network Library/Libraries to create network | 25 October | 28 October | 10 Hours |
| Create a test neural network with MNIST dataset. | 29 October | 31 October | 15 Hours |
|  |  |  |  |

Chapter 6: Development Environment

The development of this project will be developed in TensorFlow which is an open source library used for machine learning primarily in the area of neural networks. Keras will also be used in development and will run on top of TensorFlow.

6.1 TensorFlow

TensorFlow is an API that can be used in conjunction with Keras. TensorFlow is a python-based API. TensorFlow was developed by the Googles Brains Team (Tensorflow, n.d.). TensorFlow is used for computation of numerical data and also data graphs. TensorFlow allows a user to create models from scratch and offers a wide variety of classes and functions.

TensorFlow allows a developer to create a data graph and can derive how the data goes through the graph. “Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array or tensor”(Yegalup,2018).

A lot of companies use TensorFlow in day to day activities including Google, OpenAI, Snapchat etc. TensorFlow has many features including checkpoints which can be used during experimentation. TensorFlow also has a portability feature where you can spread out the stress among one or more CPU/GPUS (Tensorflow, n.d.)

6.2 Keras

Keras is an open source neural network library written in python. Keras is a deep learning library. Keras allows for rapid prototyping through modularity, extensibility and user friendliness. Keras does not use a lot of memory on the CPU and runs smoothly.

Keras reduces load on the CPU and offers simple APIs reduces the number of actions needed and provides feedback when an error occurs (Keras, n.d.).

A model can either be a graph or a sequence. Standalone models can include activation functions, optimizers, cost functions and initialization/regularization schemes. Modules can be added easily. This allows user to make to experiments more diverse.

7 Implementation

## 7.1 Sprint 1

|  |  |  |
| --- | --- | --- |
| Sprint No. | Start Date | Finish Date |
| 1 | 10th January 2019 | 20th January 2019 |

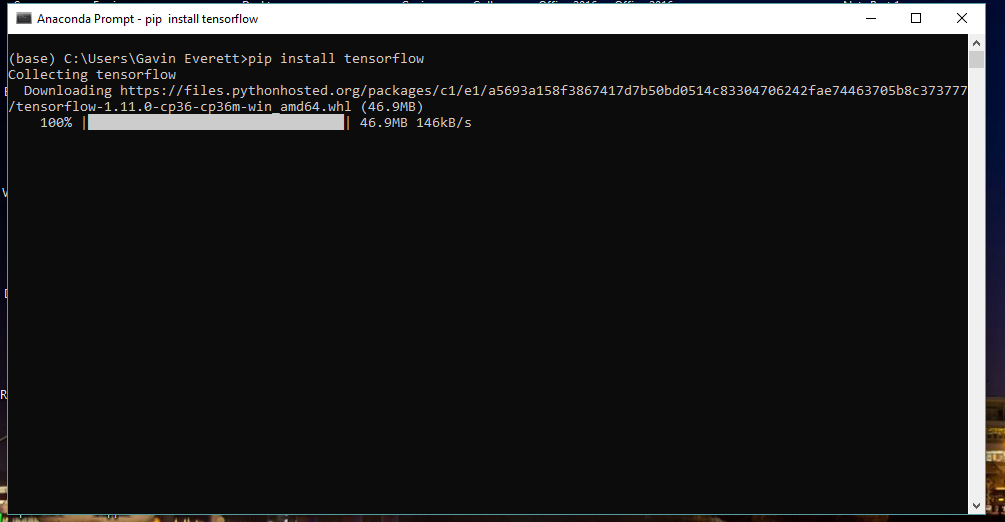
|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Setup Development Environment(Windows) | Complete |
| 2 | Download and Install Anaconda | Complete |
| 3 | Install Packages for Python 3.6 on Anaconda Prompt. | Complete |
| 4 | Install Tensor-flow and Keras Packages. | Complete |
| 5 | Install Jupyter Notebook on Anaconda | Complete |
| 6 | Create Test Neural Network using MNIST Dataset | Complete |
| 7 | Look for Datasets with relevance to project. | Complete |

Firstly Anaconda was installed. Anaconda is a software application that packages python and is used for machine learning. Anaconda purpose is to make it easier to manage packages and deployment.

The version on Python used was 3.6. This was used due to compatibility issues with TensorFlow and Keras packages.

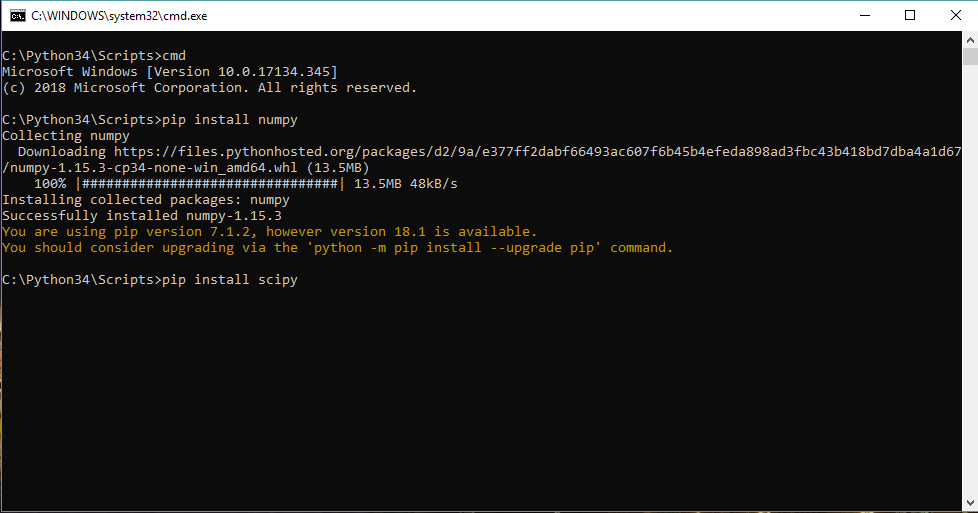
To install the library's Keras and TensorFlow library’s it is less complex to install them both through the Anaconda Prompt using the **pip** command. The pip command is generally used for installing python packages.

**Installation of TensorFlow**

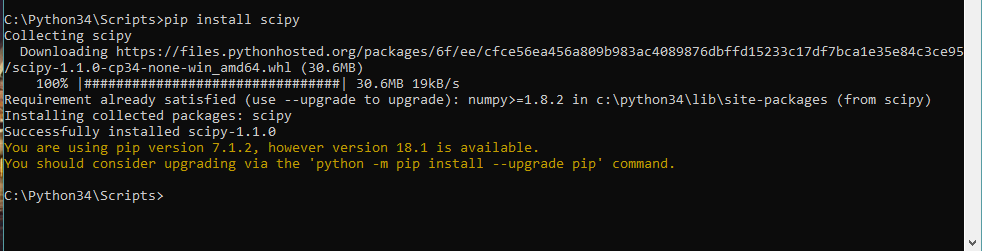


Some extra packages including NumPy and SciPy which are used for scientific calculations and are used in neural networks.

**Installation of NumPy**



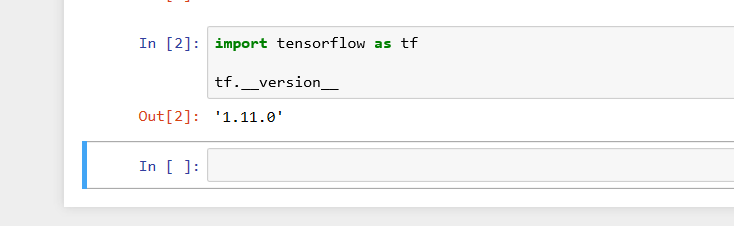
**Installation of SciPy**



One those packages were installed I used Jupyter Notebook to begin development of my neural network. Jupyter Notebook allows development in many different languages including Python.

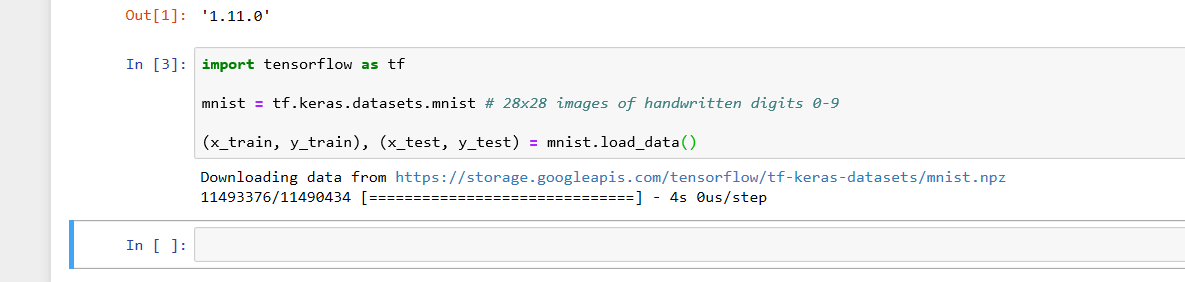
The next step was to import the TensorFlow library into my environment,

Importing TensorFlow

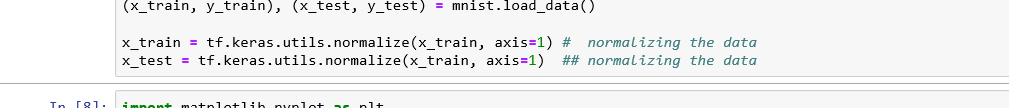


For the purpose of this prototype the MNIST dataset which is the most commonly used dataset for beginners.

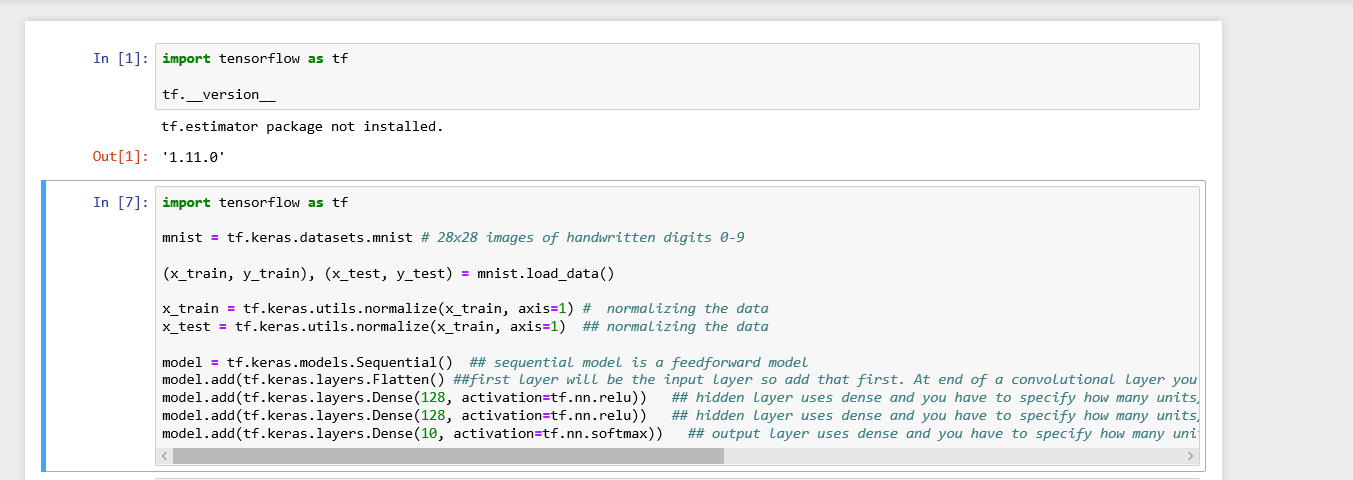
Importing MNIST Dataset

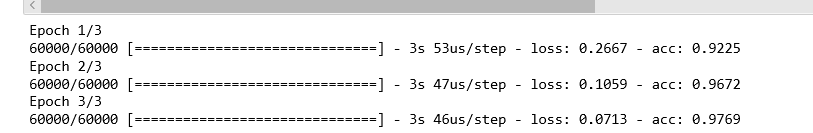


The neural network is now setup and can readily start creating the first model.



I



In the above the data is normalized. The model used for this was the sequential model which is a feedforward model. The first layer or the neural network is the input layer. At the end of the convolutional layer it is then flattened. Hidden layer dense method and it has to specify how many neurons then pass type of function to make the neuron fire. In this instance it is ReLu. In the output layer it also uses dense and the number of units has to be specified it will not have 128 it will have a set amount. This layer uses the SoftMax function.

Once the network is run with a total of 3 Epochs which shows the loss and accuracy.



## 7.2 Sprint 2 Data and Environment Preparation

|  |  |  |
| --- | --- | --- |
| Sprint No. | Start Date | Finish Date |
| 2 | 20th January 2019 | 27th January 2019 |

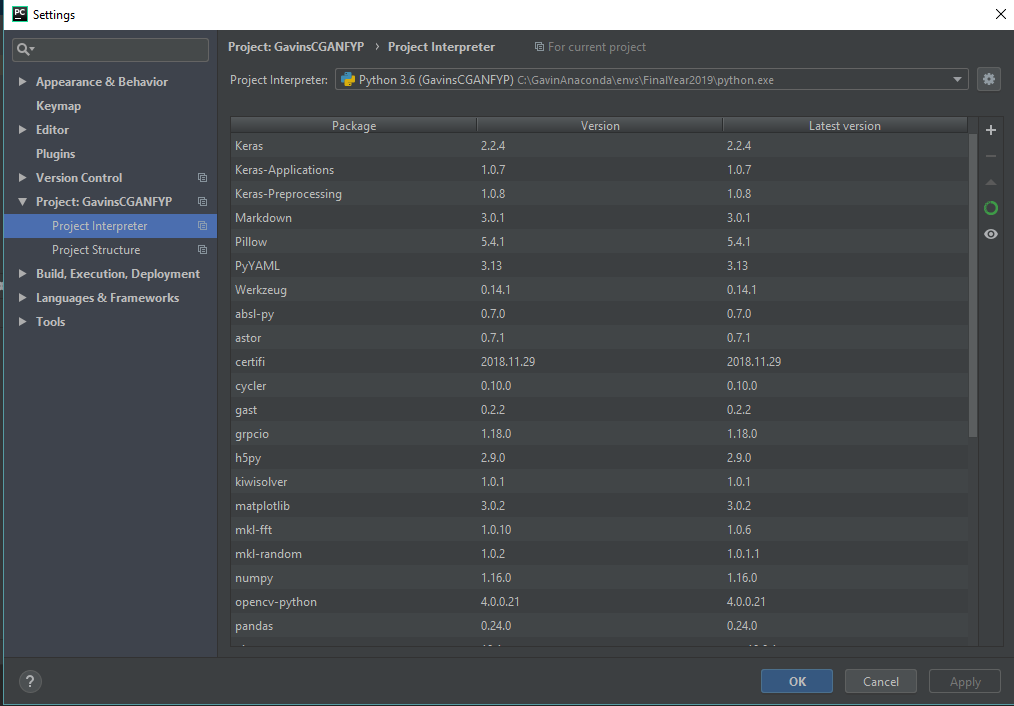
|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Setup individual Anaconda environment and connect to PyCharm Interpreter | Complete |
| 2 | Find a sketch dataset with images of sketch + image and import into application. | Complete |

### Task 1

The first task was to setup an individual conda environment which will be specific to the creation the generative adversarial network.



The above code creates an individual environment called “**generativeenv**” which will be used for the development of this project.



The conda environment can then be connected to the PyCharm IDE through /Settings/Project/Project Interpreter. The interpreter can be set by browsing for your local environment. A list of packages then shows of what is installed on that specific environment. The packages shown above have all been individually installed.

### Task 2

The aim for Task 2 was to acquire a dataset which contains real authentic images and sketches of those objects. The dataset used was the CUHK dataset which is used for face sketch recognition. It includes 188 faces from a Chinese University of Hong Kong. The first task was to format the images which included creating more readable names,

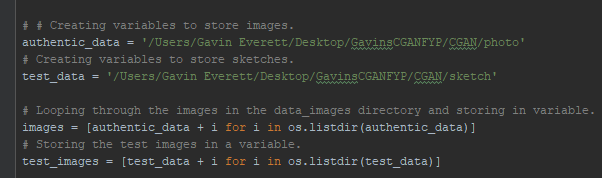


Current file name of image.



Current file name of sketch.

The below code shows the first steps of data preparation. Both the photo and sketches data were saved in separate folders and their path was saved into variables authentic\_data and test\_data respectively. To retrieve the names of all the entries in the dataset the listdir() was used for both the images and sketches.



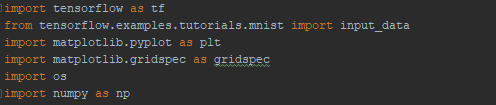
## 7.3 Sprint 3

|  |  |  |
| --- | --- | --- |
| Sprint No. | Start Date | Finish Date |
| 3 | 27th January 2019 | 3th February 2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Import libraries such as Tensorflow, matplotlib, NumPy. | Complete |
| 2 | Format images from CUHK dataset. | Complete. |
| 3 | Create a variable for sketches and also for images to store data. | Complete. |

### Task 1

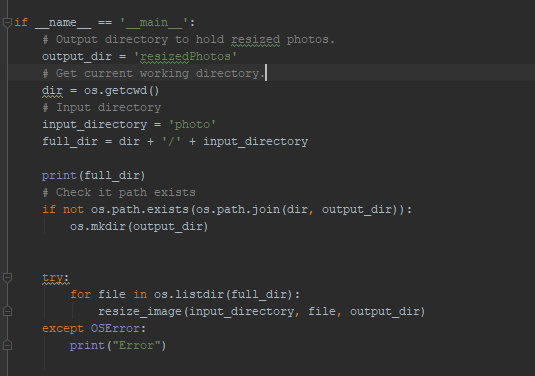
The libraries being used in this project will be Tensorflow, matplotlib and NumPy.



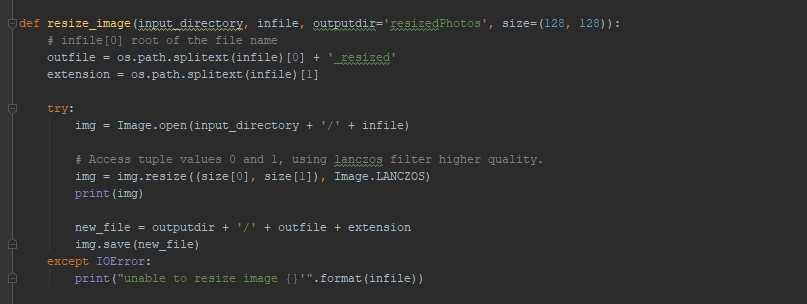
Other libraries that also will be used are PIL which will be used for image processing, OpenCV, OS , pandas.

### Task 2

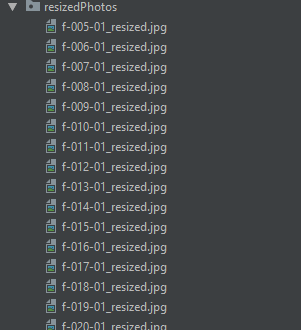
The aim for task 3 was to format the images and sketches into 128x128 respectively and store them in a new folder.



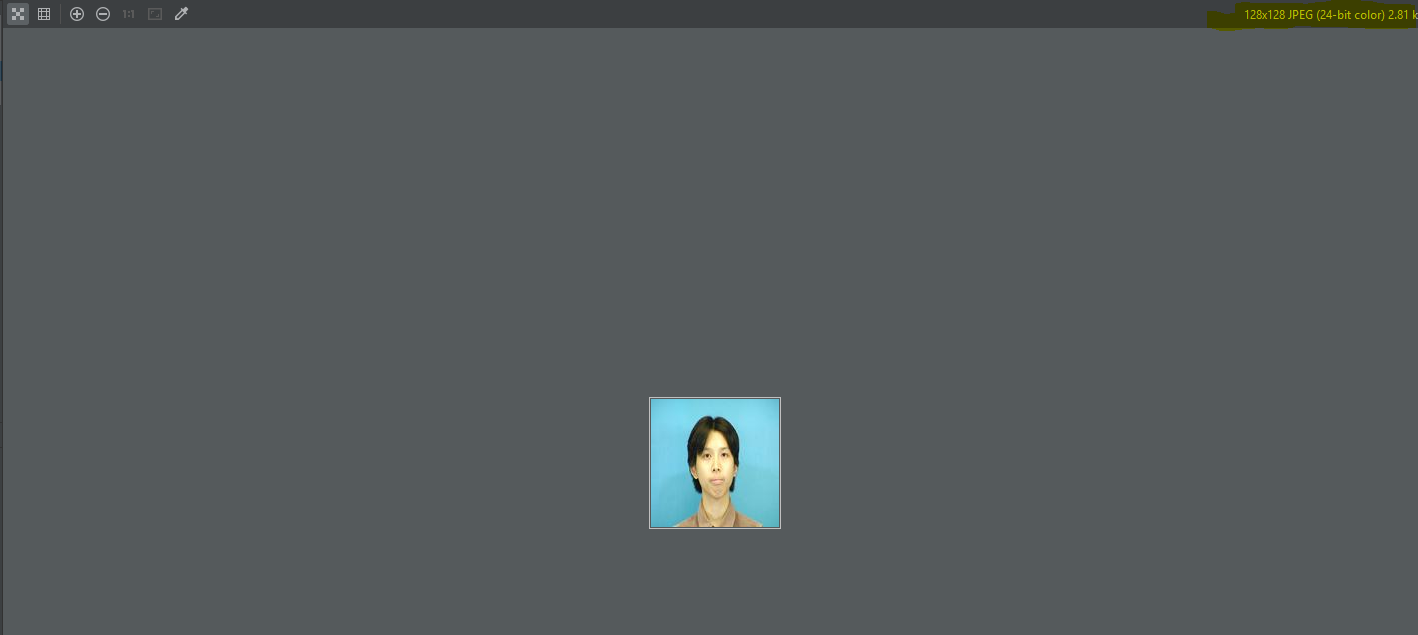
In above code an output directory is firstly created which will hold all of the resized photos the same applies to the sketches. The input directory defined is photos which currently holds all photos of random sizes. The directory for the newly resized photos is then created if it doesn’t exist already. A for-loop is then used to loop through all files in the directory and pass them into a function **resize\_image** which will be used for resizing.



The resize\_image function takes in 4 arguments, the input directory which contains the images to be resized, infile which is used for the input (i.e. the images) the output directory and the new size of the images. The variable outfile is used to obtain the root of the file name and the variable extension is used to add the extension. The images are then opened and and resized using the array tuple an using the filter lancczos which can create higher quality images. The new file generated is then stored into the output directory ‘resizedphotos’.



After running the application the resizedPhoto folder has been created with all the photos in the input directory.



Final outputted image after resizing.

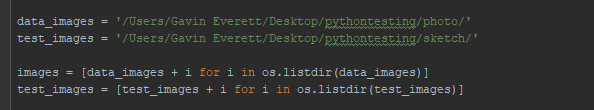
The next aim was format the format the filenames to make them more easily recognizable. The function below was implemented,



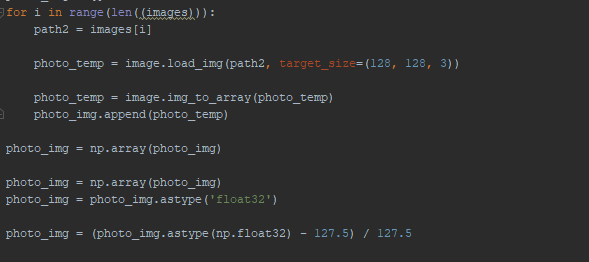
The goal was to rename both the photo and sketch filenames to more concise identifiers. For photos the filenames were changed from f-005-01\_resized.jpg to Photo-\*.jpg and sketches identifier was changed from m2-005-01\_resized.jpg to Sketch-\*.jpg. The code above loops through both directories and uses OpenCV to format the images.

### Task 3

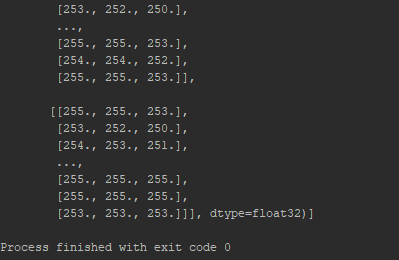
The aim for task 4 was to obtain the newly resized images and store them into an array to create a single location for the training data. To do this a loop over the directory was used and the image was stored. Keras pre-processing was used to prepare the images.



The range module was used to find all photos in the path open them and store them in the arrays images and sketches.



After retrieving the images from the directory, the task then was to process the images using the pre-processing function by Keras. The images retrieved are passed through the **img\_to\_array** function which will translate the PIL image to a NumPy array value. The values are then appended to an NumPy array. The as type method is used on the array to cast to float for safety. Mean is then used to obtain the mean of the NumPy array values which will be used in the development of the networks. The images are now fully processed and ready for use to feed into the desired networks.



The output of some of the NumPy array can be seen above.

## 7.4 Sprint 4

|  |  |  |
| --- | --- | --- |
| Sprint No. | Start Date | Finish Date |
| 4 | 3rd February 2019 | 17th February 2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Create parameters and set values for the model including noise size, batch size, epochs and samples | Complete. |
| 2 | Set hidden units for both the Generator and Discriminator Networks. | Complete. |
| 3 | Define Generator Network with layers and return model. | Complete. |
| 4 | Define Discriminator Network with layers and return model. | Complete. |
| 5 | Create the model | Complete. |
| 6 | Add a variable Y which will store the conditional value to be picked from distribution | Complete. |

### Task 1, 2, 3

The aim of this task was to define the parameters and conditions needed to feed into the network. The set of parameters that were created for the images are outlined below,

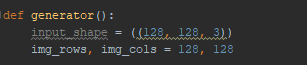
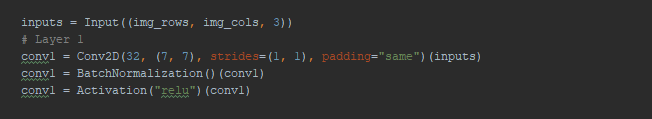


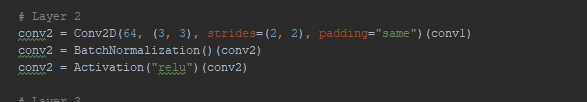
Image Parameters.

The image input will be 128,128,3 which has already been defined in the pre-processing stage. The image rows and image columns follows the same format.

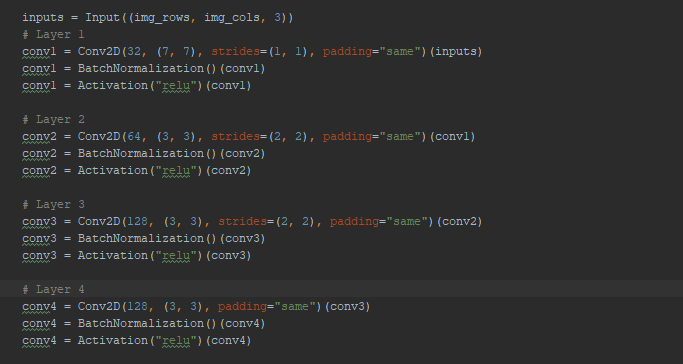


Structure of GAN

All layers in the GAN will contain a convolutional layer, batch normalization layer and activation layer in this instance. Conv2D will be used as a stack of images will be inputted into the network. The batch normalization layer is used to improve the stability of the network as a whole. The final layer is the activation layer in which ‘relu’ activation will be used. This will be the basis of the generator network in terms of the structure of the layers. More blocks will be added in order to improve the accuracy as development continues.

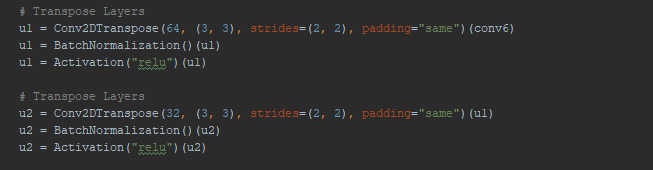


The second convolution will contain the same structure as the first, the only difference is that first and second parameters have been adjusted. The filter value (the first parameter) has been adjusted from 32 to 64. This practice has been recommended for CNN architectures. The second parameter (the kernel\_size) has also been updated from (7,7) to (3,3). Since the images inputted are equal to 128x128 it is good practice to use a kernel size greater than 3 to learn the small features and then gradually learn larger features on the images. Strides default implementation will be used and the padding will be ‘same’ which preservers spatial dimensions.



Overall Structure

Since the network needs to go in the opposite direction Conv2dTranspose will be used ‘from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution.’ The Con2dTranspose layer will only be used for the generator network. The transpose layer applies a (2,2) ratio stride across the input and in smaller regions.



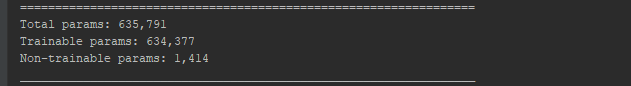
Conv2dTranspose Layers

After defining the structure, the model is then returned. The model takes in both inputs and output tensors. The input in this case will be ‘inputs’ which is the first layers and the outputs will be the last layer i.e. ‘conv7’.



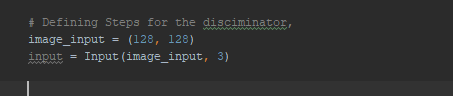
Defining the generator model.

A summary of the model is shown below,



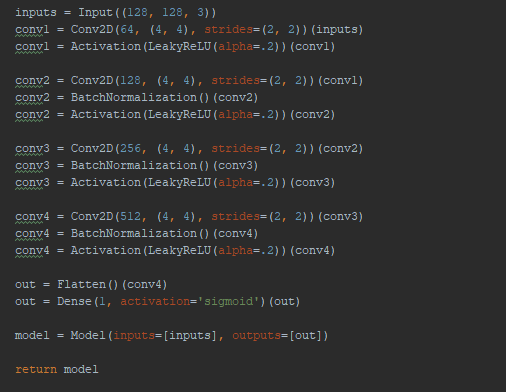
### Task 4

The aim of this task is to define the steps in order to create the Discriminator network. The discriminator network will be the ‘cop’ in the network trying to identify real input from fake input. The network itself is pretty much the same as the generator network with slight variations.



Parameters for Discriminator.

Once again, the parameters will be the same for both the generator and discriminator networks. As for the convolutions they will be the same. The discriminator is a 4 strided convolution with batch normalization except for the input layer as shown below,



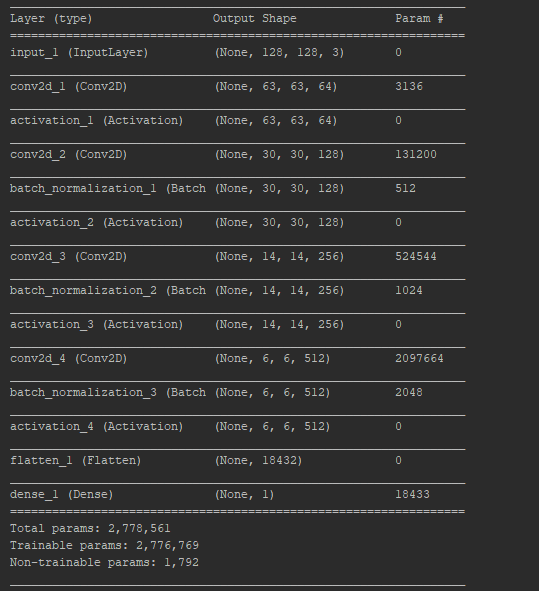
Discriminator Structure

From the above image you can see that the first layer does not use batch normalization, this is because it can produce unstable results if batch normalization is used in the input layer as cited by DCGAN. The reason for the use of the LeakyReLU is to allow for gradients to flow easier through the network. Sometimes when the discriminator receives input from the generator the network may get stuck in what is known as the ‘**dying state**’. In the dying state the network won’t produce anything is stuck. By using LeakyReLU it can prevent this state and allows for negative values to pass through. If the network receives negative values the LeakyReLU operation uses the ‘**alpha’** argument to create tolerance through by allowing negative values to pass through**.** Overall by using this operation it will improve the stability of the network. The final step in developing the discriminator was to define the sigmoid function this is used as the discriminator needs to output a probability if the image is real or fake. The discriminator is also using the opposite structure to the generator also know as DCGAN. This is where batch normalization is done last because the output is the probability.



Sigmoid function.

The sigmoid function compresses the output to a value in the range between 0 and 1. 0 being fake and 1 being real.



Model Summary

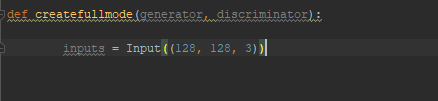
The image above shows the summary of the discriminator model with the layers, output shape and parameters.

Statistics

* **Total parameters:** 2,778,561
* **Trainable parameters:** 2,776,769
* **Non-trainable parameters:** 1,792

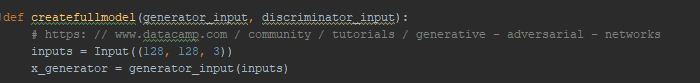
### Task 5

The aim of this task is to define a function that takes in both the generator and discriminator networks and forms one complete model which can then be used for training.



Defining parameters for full model.

The parameters for the model were the same as the overall structure of the generator and discriminator networks. The generator will be stored in a variable ‘**generator\_input**’.



Acquiring the generator inputs.

In the above image the ‘**x\_generator**’ was being assigned to the generator network with the defined row, cols and channels. The rows are 128, cols are 128 and channels is 3.



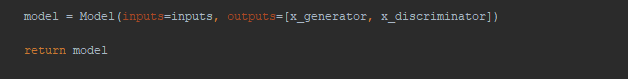
Setting the discriminator to false.

By setting the discriminator to false this means that the generator network will run before the discriminator which is what is needed. The next step is to acquire the output produced by the discriminator i.e. the probability of the generated image being real or fake,



Acquiring the probability.

In the above image I am passing the generators noise(image) into the discriminator which will output the probability of the image. Optimisation and loss functions will be created later in development. The model can now be created. The inputs for the model will be the dimensions 128x128x3 and the outputs will be the probability of the image being real or fake.



Returning the final model.

## 7.5 Sprint 5

|  |  |  |
| --- | --- | --- |
| Sprint No. | Start Date | Finish Date |
| 5 | 17th February 2019 | 24th February 2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Create Generator loss function + optimizers | Complete |
| 2 | Create Discriminator loss function + optimizers | Complete |
| 3 | Compile full model. | Complete |

### Task 1

The aim of this task was to create a loss function for the generator. For simplicity the loss function was derived from the Keras API. The optimizer chosen was the Adam Optimizer as this has been shown to produce the best results. The default parameters for the Adam optimizer are as follows,



Default Parameters.

After tweaking the parameters to improve the gradient spread the following updated parameters were used,



Updated Parameter Settings.

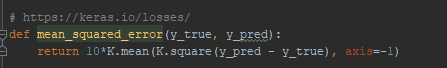
The ‘amsgrad’ parameter was removed as it applies the ‘amsGrad’ variant which has not be researched in this report. The epsilon parameters value was updated to ‘1e-8’ which is the default value. The learning rate was tweaked in between the running of the network to achieve an optimal output image.

RMSProp was also used from Keras as it can be used for gradient optimization. The parameters are not to be changed except for the learning rate which can be tweaked freely.



RMSProp Setting.

The next task was to create the loss functions that will be used for the network. Most GANs use the mean squared error loss function which helps with backpropagation process.



Mean Squared Error.

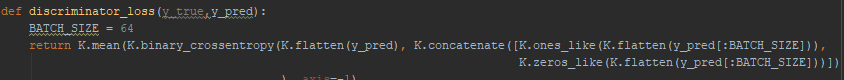
The ‘mean\_squared\_error’ loss function is available on the Keras API. Once the method was defined the generator can now be compiled using the compile method from the model class API. The compile method takes in parameters such as the loss function and the optimizer.



Compiling Generator Model.

### Task 2

The aim of this task was to create a loss function for the discriminator model. The discriminator will also use the optimizers Adam and RMSProp which is the same as the generator model. The discriminators loss function was to compare the batches of images for the real data and fake data,



Discriminator Loss.

### Task 3

Once the loss functions and optimizers have been applied to both the generator and discriminator models a call to the full model function can them be called to apply the loss and optimizer to the whole network shown below,



Full Model Loss.

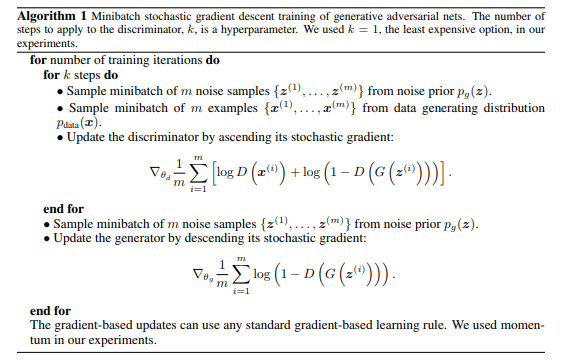
## 7.6 Sprint 6

|  |  |  |
| --- | --- | --- |
| Sprint No. | Start Date | Finish Date |
| 6 | 24th February 2019 | 1st March 2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Create functionality to train model. | Complete |

### Task 1

The training loop can now be implemented. The training loop was proposed by Ian Goodfellow one of the founders of the Generative Adversarial Network.



Algorithm Implementation.

The steps of the algorithm in a simplified version are as follows,

**Steps**

**1.**Get a sample batch of noise and a sample batch of the real data of size m.

**2.** Feed this data into the discriminator network.

**3.** Sample a different subset of noise of size m.

**4.** Train the Generator with this data.

**5.** Repeat process.

The first step is to define the number of epochs and batch size. The epoch size for this network will be 1300. The batch size will also be 87.

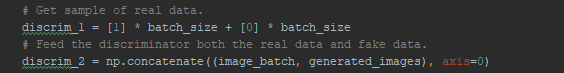
A loop was created with the specified number of epochs which is currently 1300. A loop was then created for the batches created. In the second loop for the batches the first step of the algorithm in being performed. Looping over the sketches and sampling a batch size of **m**.

To predict the probability of the image generated is real or fake an instance of the generator model stored in variable ‘**generator\_input**’ was needed. A batch of sketches was then stored in a variable ‘**sketch\_batch’** and these sketches where then passed into a prediction function from the Keras API. The predict function takes in the sketches, batch\_size and verbose. The verbose variable will be set to 1 just for output purposes. The generated images are then stored in the ‘**generated\_images**’ variable.



Generator training loop structure.

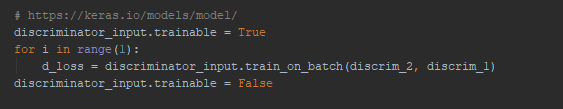
The next aim to create the same structure for the discriminator model and by using the **concatenate function** from the Keras API which takes in an input of tensors and returns a single tensor.



Discriminator training loop structure.

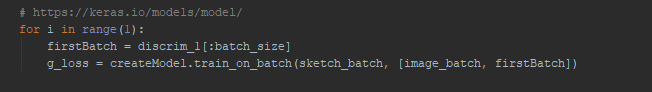
The ‘**image\_batch’** variable holds a batch of images from the image’s directory. The **‘discrim\_2’** concatenates the real data which is the authentic images with the generated images. The last task is to scale the images. Next the discriminator can now run by setting the Trainable variable back to True.

The next task was to obtain both the losses for the generator and discriminator networks. The discriminator loss was obtained through the following,



The discriminator is set to true so it can be trained. A for loop is then created in the range of ‘1’ which mean 1 image per epoch/iteration. A variable ‘**d\_loss**’ is then defined which is equalled to the discriminator network which uses the ‘**train\_on\_batch**’ function from the Keras API which takes in a test value and a target value. In this case the test value will be the generated images and the target value will be the real training data. The discriminator network is then set to false to allow for the generator loss to be calculated.

The generator loss is calculated through the following,



It first uses a for loop in the range of ‘1’ which takes in 1 image per epoch/iteration. A batch is then created from the real data. A variable ‘g\_loss’ is then created which is equalled to the full model which uses the ‘train\_on\_batch’ function and takes in a test value which will be the batch of sketches and the real data. This the returns the calculated value.

Below is the overall structure of the training loop,



Training Loop Structure.

It begins by looping over a set number of epochs in this case is 1300 which can be changed at any time. A temporary variable was then created to store the images in a NumPy array as you cannot directly loop over the shape in the array. The next step was to create batches of images in this instance 87 was used but can be changed to any number. A loop was then created to loop over the length of the batch count. Variables were then created to access both the discriminator and generator networks. A batch of 87 was then taken from the generator and the predict function was used to output the probability of it being authentic or fake. For the discriminator model the first step was to acquire a batch of real data from the dataset. Then the aim was to concatenate both the real data and generated data into a full model.

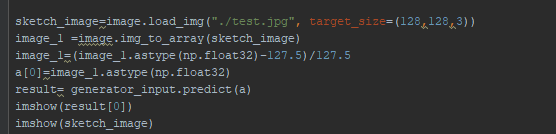
## 7.7 Sprint 7

|  |  |  |
| --- | --- | --- |
| Sprint No. | Start Date | Finish Date |
| 7 | 1st March 2019 | 14st March 2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Print results of training. | Complete |

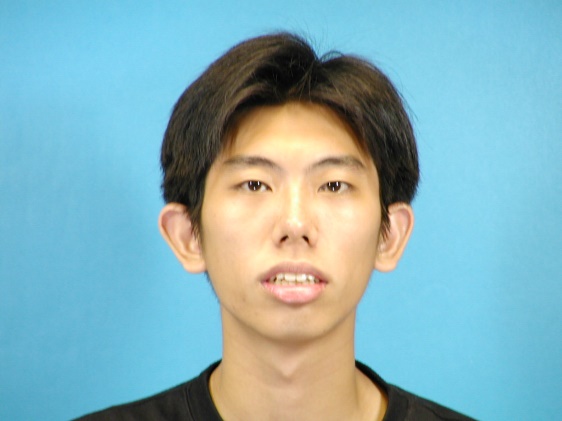
### Task 1

The aim of this task was to output the results after the model has finished training on the specified input sketch. To output the result, the image was passed to the generators prediction function which will output the fake generated image at the end of the training period. The sketch will be formatted and then passed to the generators prediction function and then be displayed.



Display Result.

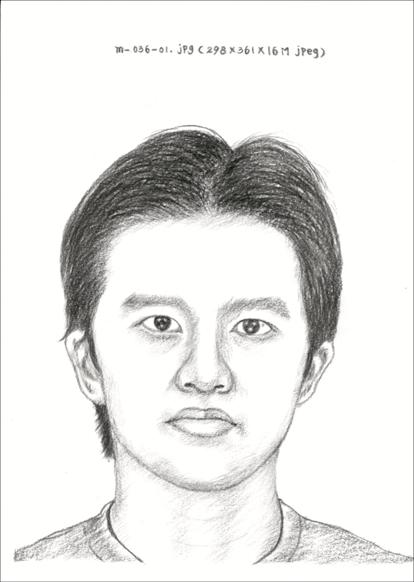
Once the sketch has been formatted it was then be passed to the prediction function and then will be displayed through ‘imshow’ which a method is to display images. Below is the result of the network after running for 1300 epochs and a batch size of 87.



Results.



` Results.



Results.



Generator Accuracy.

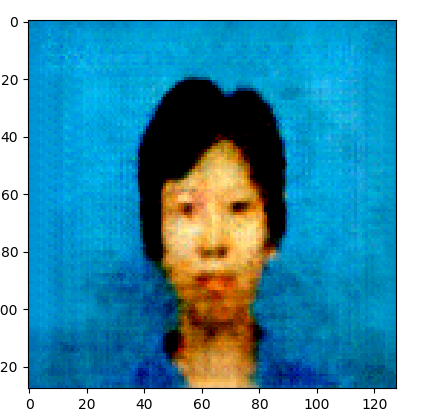
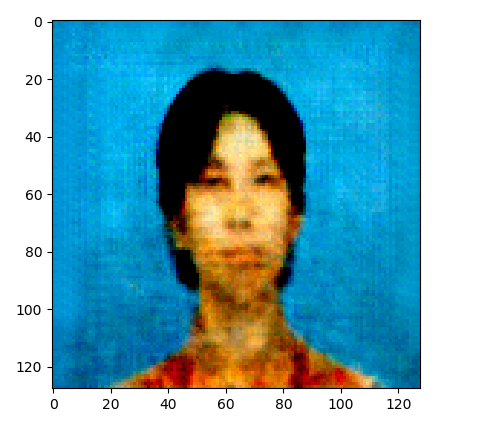
## 7.8 Sprint 8

|  |  |  |
| --- | --- | --- |
| Sprint No. | Start Date | Finish Date |
| 7 | 15st March 2019 | 1st April 2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Compare optimizers / adam / adagrad. | Complete |

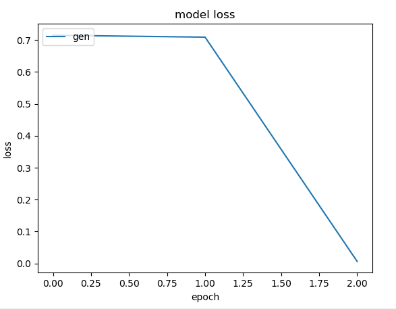
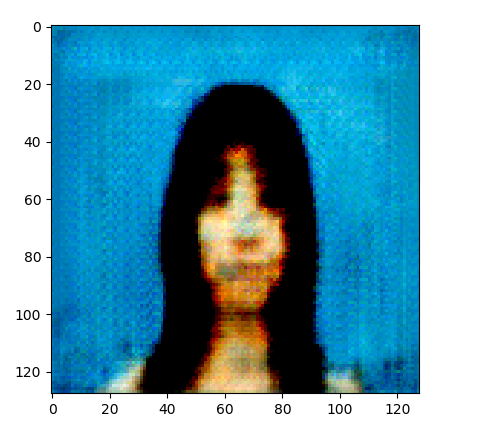
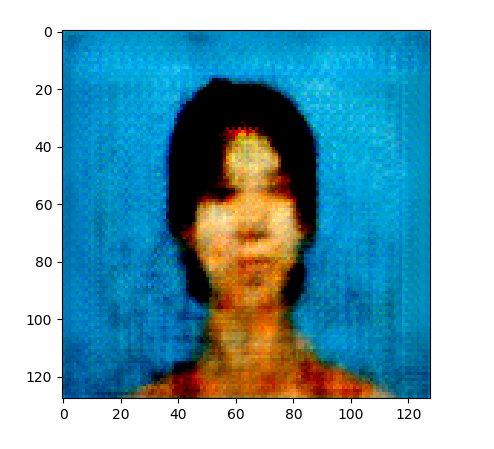
### Task 1

The goal of this task is to compare two optimizers which have been investigated as the most optimal for results for the generator adversarial network. The first optimizer to be tested was the Adam optimizer which returned respectable results and was able to deceive the discriminator pretty well. Both optimizers used the default parameters and used 1300 epochs and a batch size of 64. Below are the results produced by the Adam optimizer.



Adam Optimizer Results

The next goal was to test out the adagrad optimizer with the same parameters. The results are shown below,



Adagrad Results.

From the above results you can see that the Adam optimizer was much more sucessful at deceiving the discriminator and producing a more authentic result.

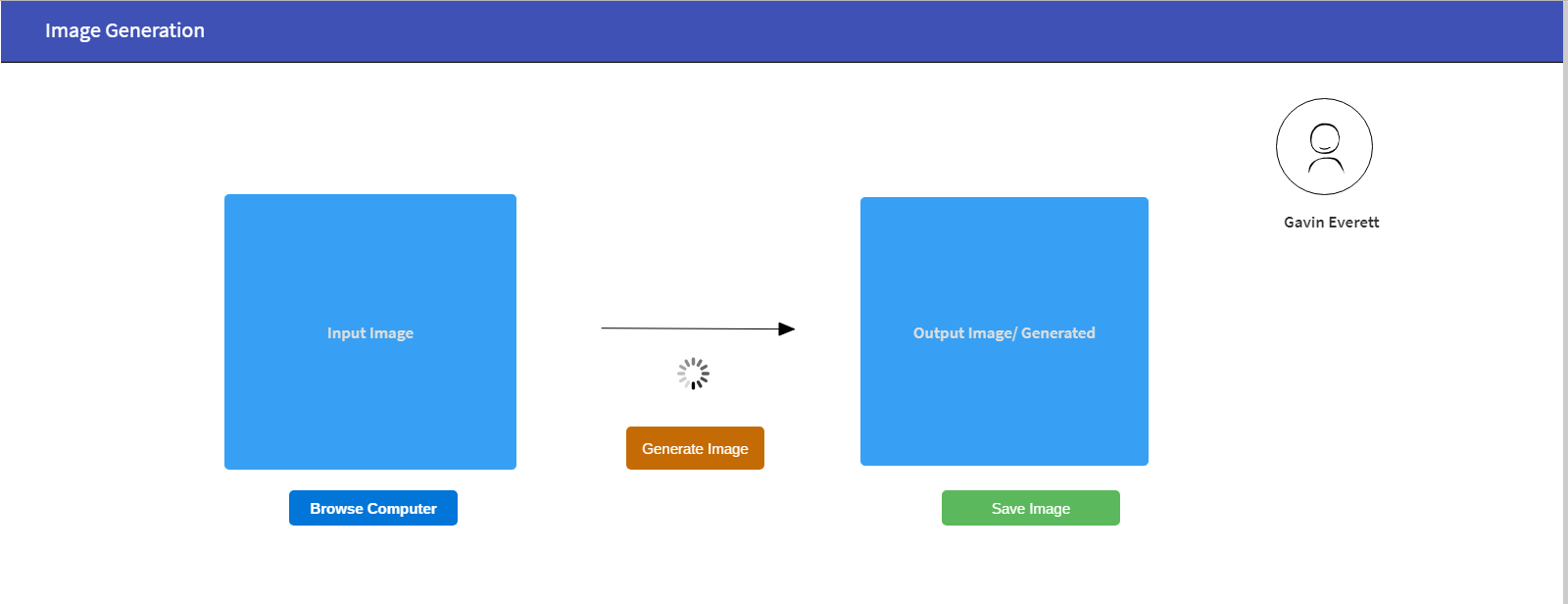
## 7.9 Sprint 9

|  |  |  |
| --- | --- | --- |
| Sprint No. | Start Date | Finish Date |
| 9 | 2nd April 2019 | 15th April 2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Design mock view of interface | Complete |
| 2 | Identify framework to connect reactjs and python | Complete |
| 3 | Install framework, xampp, atom. | Complete |
| 4 | Create a model and save to .h5 file | Complete |
| 5 | Create React JS application | Complete |
| 6 | Import Particle JS | Complete |

### Task 1

The aim of this task was to create a front-end website/application using React JS. The website will have options to select one of the sketches from the dataset. There will also be a generate button which will generate the output of said image which will return the output from the generator network. An example of the website was designed in MockFlow,



Example design of application.

The front end of the application will be run on ReactJS and the back end will be run on python. The user will browse the dataset provided and click ‘generate image’. On the click of generate image the back end will start running the network and generate the image into the ‘generated image’ space.

### Task 2

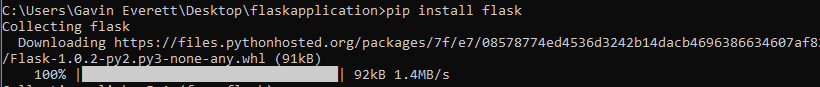
The framework that has been identified in order to connect the front-end in react js and the back-end python is Flask. Flask allows for there to be a connection between the front and back end.



(Flask, 2019)

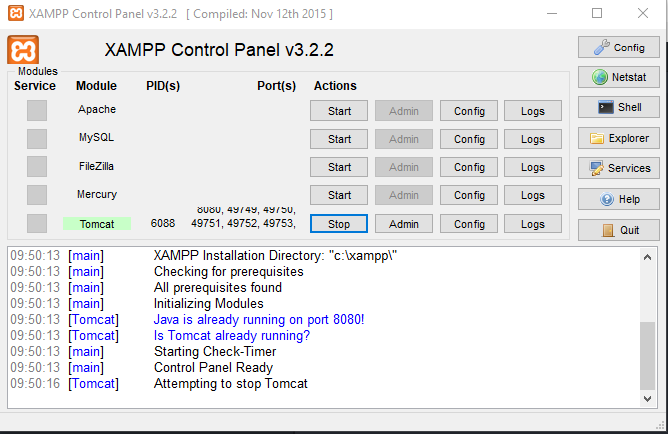
### Task 3

The aim of this task was to install all of the required components to develop the application. The first component needed was the flask framework which was installed using pip through the command line,



Installing flask framework.

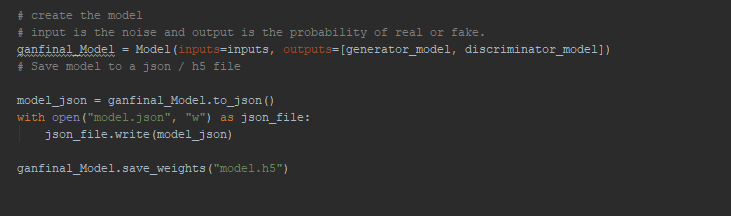
The next component to be installed was XAMPP control panel which will give access to the react front-end on apache on port 5000.



XAMPP Control Panel.

### Task 4

The aim of task 4 was to run the network and save the model. The weights will be saved into what is called a h5 file which will consist of weights and other configurations. The architecture of the application will be stored in a JSON file. The model will be saved after creating the full model,



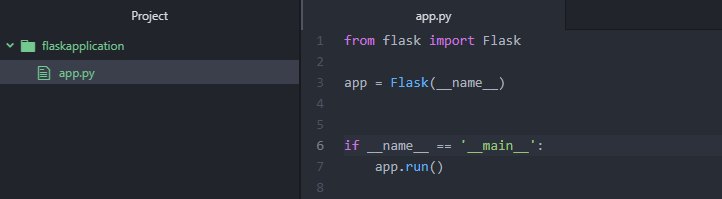
Saving GAN Model

The saved model will be used for in the flask application to get predictions on images passed to it. Keras provides the save format using the HDF5 saving format which will represent the file as a binary blob. This type of save is useful for using in the web browser.

Once the network has finished running the model will be then saved to a .h5 file.

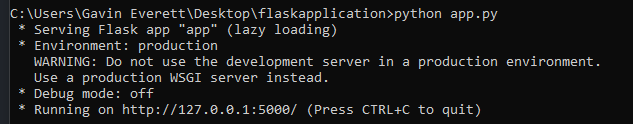
### Task 5

After all components have been setup correctly the app can now be developed. The IDE for development will be Atom IDE.

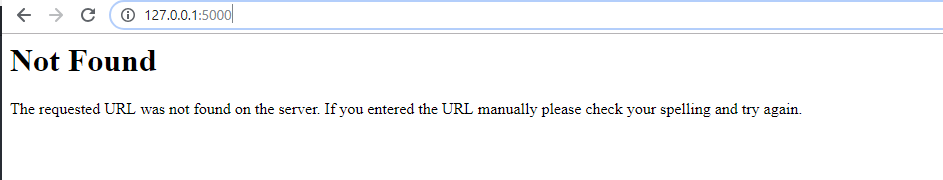


Basic Flask Structure

The application can be run through the command line and will open up on port 5000.

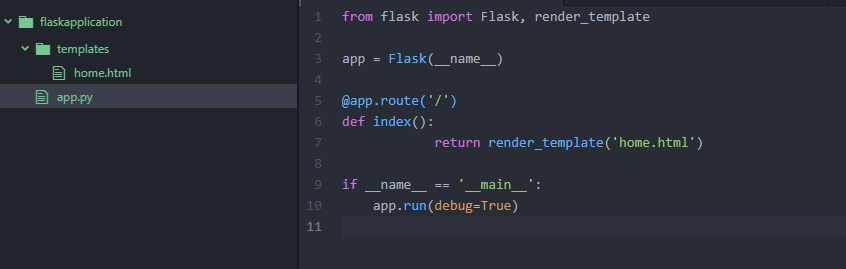


Port 5000



Browser view

The page brings back a not found error due to not having the home paths setup. To setup a home path flask has a component called ‘**render\_template’** which allows for a html page to be passed in as an argument,



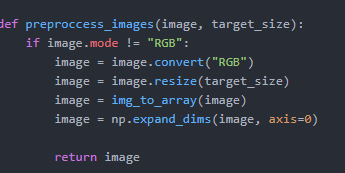
Flask Structure

In the above the home.html is the template for the flask application. The next steps involve retrieving the h5 model and decoding the encoded data from the model.



Retrieving the model.

First step was to retrieve the model from the specified directory and store it in a variable called model.



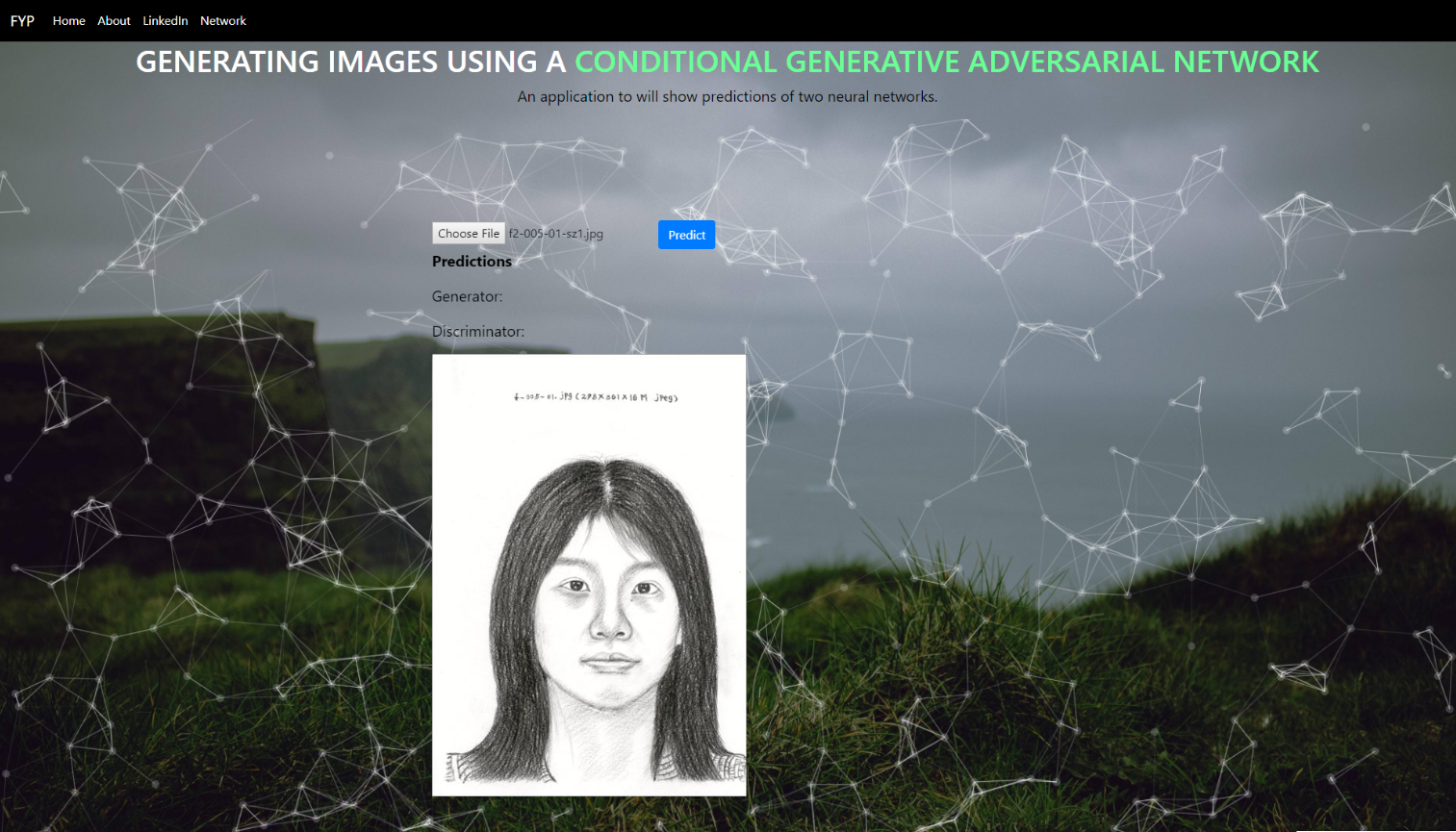
Pre-Process Images.

The next step involved pre-processing images. The images will have to be pre-processed once they were selected from the specified directory and returned.



Decoding the images.

The final step involves decoding the image data and creating a prediction based off the decoded data. The methods used is **‘POST’.** This app.route will sit behind a button on the UI. The method takes in an encoded image and decodes that image and then the image is processed. A prediction is then made based off the image selected. The return value is a json string.



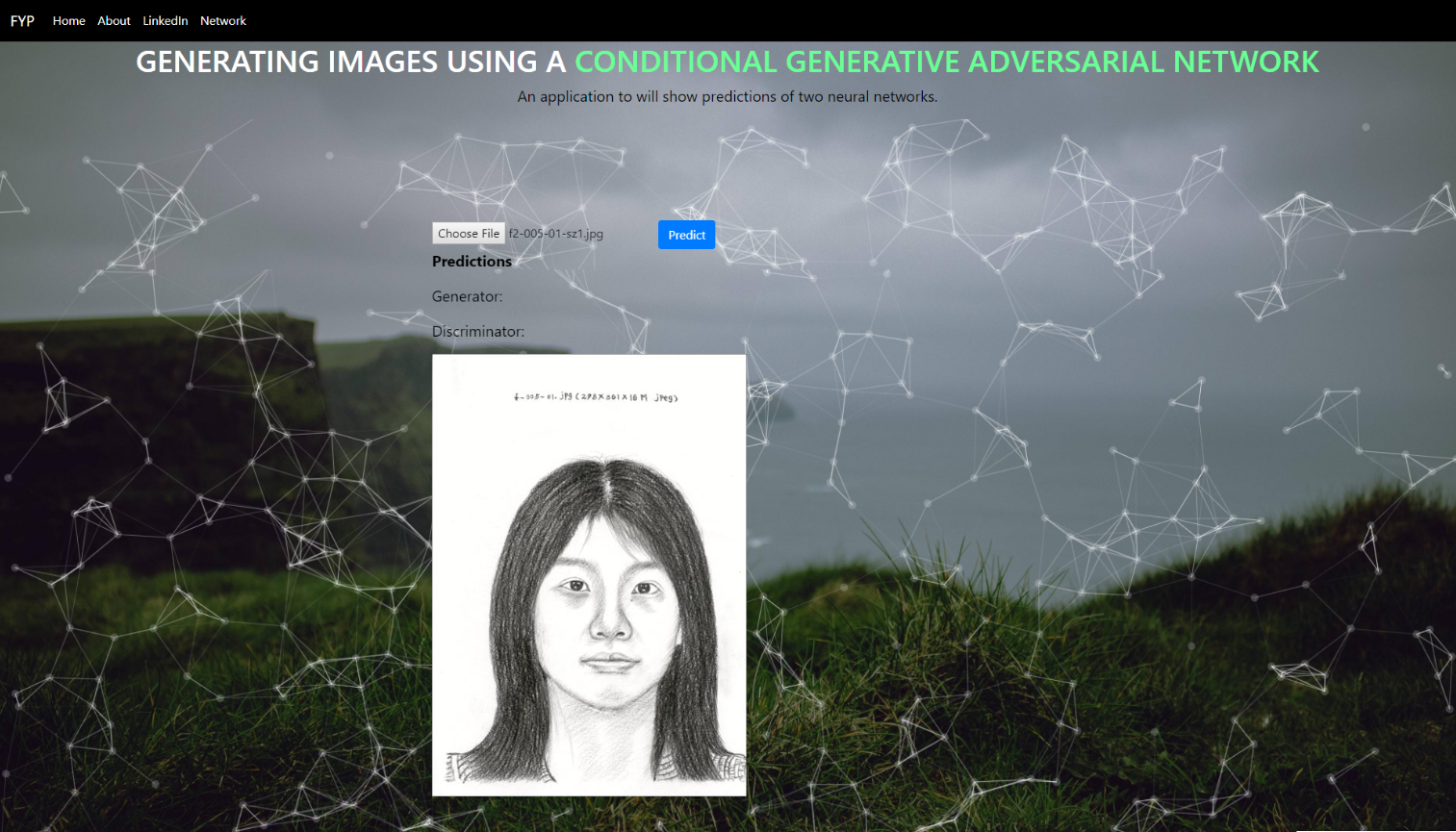
Front-End UI in Flask.

The front-end interface consists of an image selector, prediction function and two text outputs for probability, generator and discriminator. The prediction button is used to predict the probability of the image given,



Predict Image.

The above code is linked behind of the prediction button. The image will be formatted and then a JSON response will be set to the two text fields through jQuery. An example output is shown below,



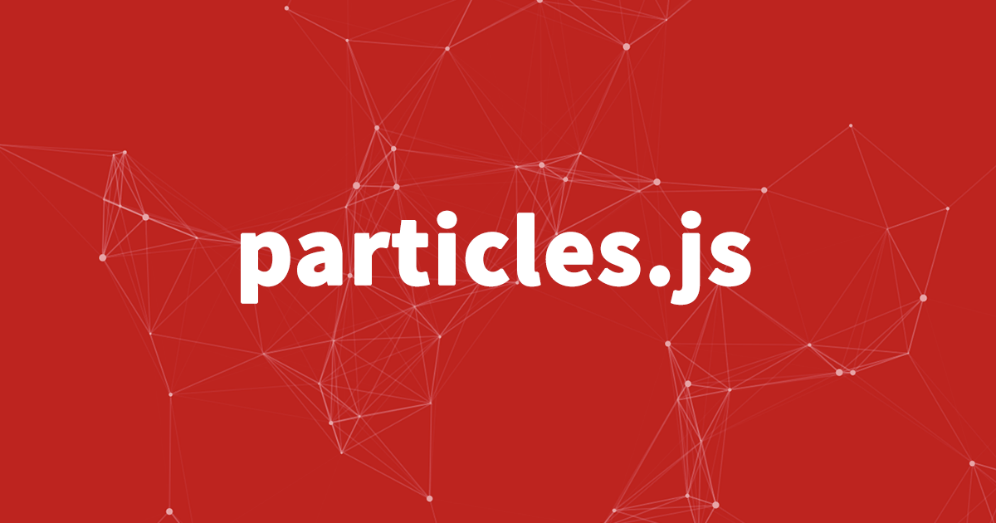
Output predictions from model.



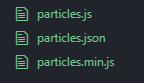
Output from console window.

### Task 6

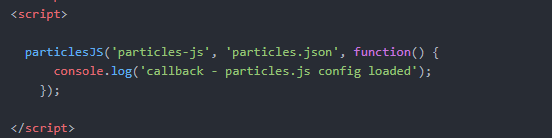
The aim of this task was to implement a light-weight JavaScript library to add aesthetics to the front- end interface. The library used was particle.js which is open source.



The implementation of the JavaScript library was straight-forward. It involved applying some JavaScript and json files to the root of the project folder.



The basic configurations for particle.js is already setup in the files shown above. In order to activate the particle effect, it can be done so through a jQuery function,



jQuery Activation.

This activates the json file which in returns displays the particle effect on screen.

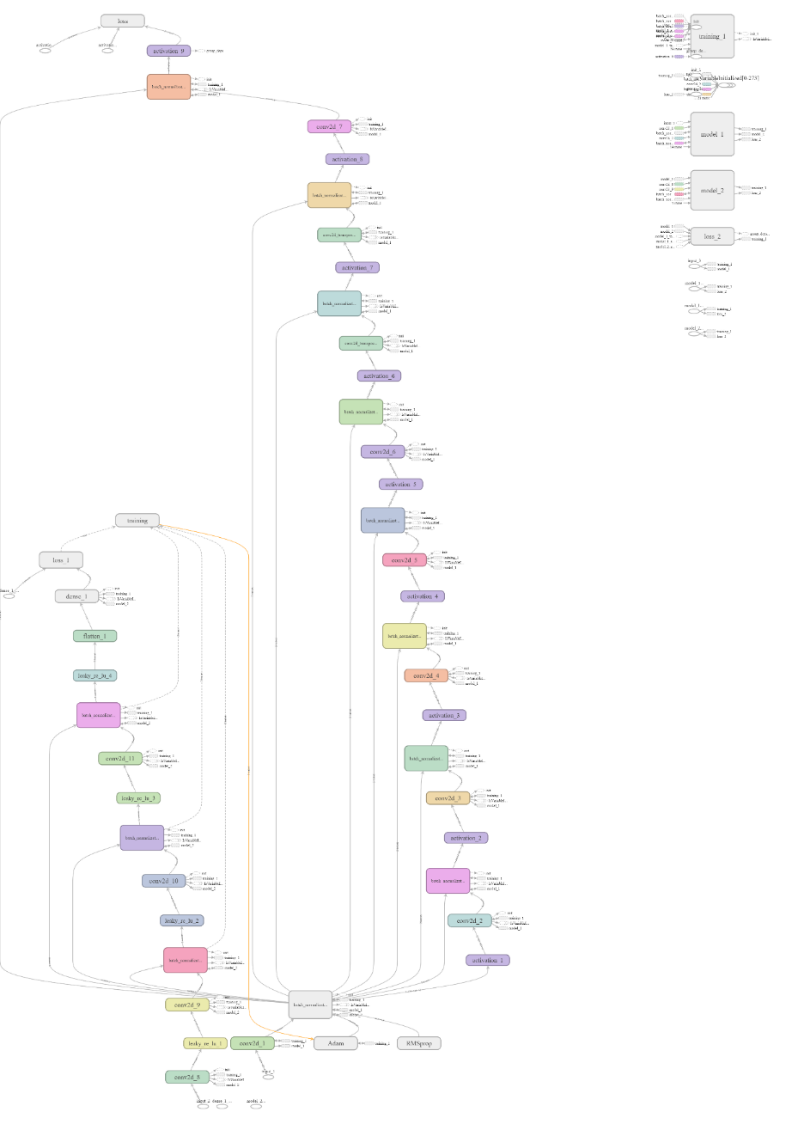
## 7.10 Sprint 10

|  |  |  |
| --- | --- | --- |
| Sprint No. | Start Date | Finish Date |
| 9 | 2nd April 2019 | 15th April 2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Visualise Model and Analyse data with Tensor board. | Incomplete. |

### Task 1

The aim of this task was to visualise the data retrieved from the model and plot it on a graph to analyse the data of the both networks. Below is a current view of what the full model looks like,



The model above contains two networks the generator and the discriminator network.

# 8 Findings and Conclusions

## 8.1 Computational Performance

Figure xxvii below demonstration of the networks running on a GeForce GTX 1070 Ti over a period of training of 1300 epochs.

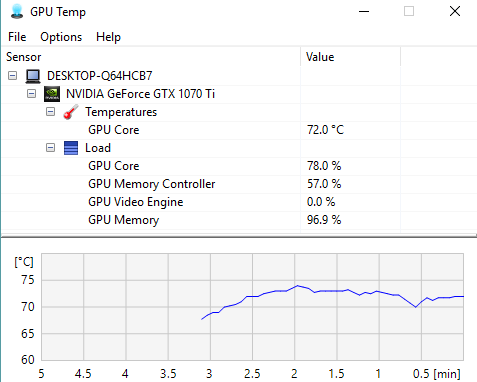


Figure xxvii Computational Performance

From the above the Graphic Processing Units temperature raises significantly upon starting the network and then gradually becomes static and slowly declines. Tensorflow GPU was used for the computations and can give very efficient and accurate predictions of data.

## 8.2 Training Observations

Training observations were made through matplotlib and output from the console window in PyCharm. The optimizer used throughout the training period was the Adam optimizer. The Adam optimizer proved best for performance in the sense that the generator model was able to deceive the discriminator network much better which resulted in more authentic images produced.

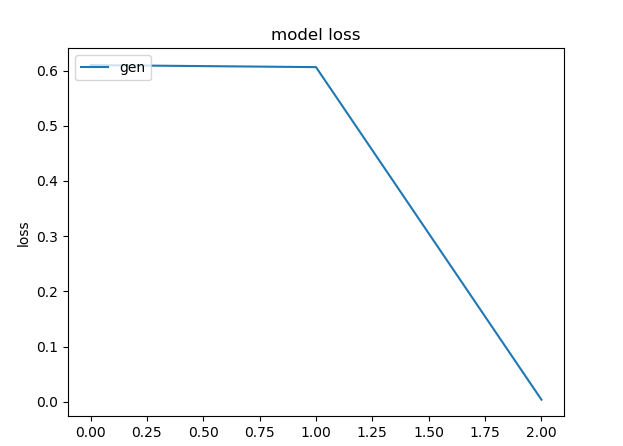


Figure xxviii Generator Loss

The above graph was created using matplotlib. The results illustrate that when using the Adam optimizer, the generator gradually begins to deceive the generator after a certain number of epochs which results in better quality images produced by the model. The result of **85%** can be seen as a relatively respectable figure due to the fact the generator adversarial network is a relatively new subset to the artificial family. There is no benchmarks currently available.

The networks both improved in stability when batch normalisation was not included in the final output layer of the generator network and the input layer of the discriminator network. LeakyRelu activation from the discriminator network also provided a more even flow of gradients through the network by using the alpha function to provide tolerance for negative values. By adding more layers to the generator network this improved the result of the generated images by tweaking the kernel and filter size at each layer in the network.

Different optimizers were compared such as Adam and Adagrad optimizers which were trained with the same set of criteria which resulted in the Adam optimizer being able to deceive the generator at a faster rate in comparison to the Adagrad optimizer.

When creating the full model as there was two networks involved the discriminator network had to be set to false to allow the generator to run. When acquiring the loss values the discriminator was set to true and then set to false once the accuracy was computed so that the generators accuracy could then be computed. Both of the networks were running concurrently during the training loop.

## 8.3 Research Question Answered

The Conditional Generative Adversarial Network (CGAN) was successfully implemented to create image representations from sketches. The training of the network was carried out on Nvidia GeForce GTX 1070 Ti. In addition to this the accuracy obtained can be seen as relatively accurate due to the power of the graphics card. In terms of the quality of the image produced by the Conditional Generative Adversarial Network it is limited in the sense that the dataset only has a small number of images and sketches in total there were 87 sketches and 87 photos. Additional techniques that could have been used for the implementation included acquiring a larger dataset of image and sketches or the testing of different optimizers.

# Bibliography

Aghdam, H. H., 2017. *Guide to Convolutional Neural Networks.* 1st ed. Tarragona: Springer.

Albawi, S., Mohammed, T. A. & Al-Zawi, S., 2017. Understanding of a convolutional neural network. *Understanding of a convolutional neural network,* p. 6.

Alee Radford & Luke Metz, S. C., 2016. *Unsupervised representation learning with deep convolutional generative adversarial networks.* Boston, ICLR, p. 16.

Alpaudyn, E., 2010. *Introduction to Machine Learning.* 2nd ed. London: The MIT Press.

Anon., 1987. *IEEE First Annual Press Conference On Neural Networks.* San Diego, IEEE.

Anon., 2017. HEp-2 Cell Image Classification With Deep Convolutional Neural Networks. *HEp-2 Cell Image Classification With Deep Convolutional Neural Networks,* p. 13.

Anon., 2018. *How does a Generator (GAN) create samples similar to the data space from a vector of random numbers?.* [Online]   
Available at: https://stats.stackexchange.com/questions/278623/how-does-a-generator-gan-create-samples-similar-to-the-data-space-from-a-vecto  
[Accessed 03 12 2018].

Antonia Creswell, T. W. V. D. K. A. S. A. A. B., 2018. *Generative Adversarial Networks.* [Online]   
Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8253599  
[Accessed 16 November 2018].

Arjovsky, M., 2017. Wasserstein GAN. *Wasserstein GAN,* 1(1), p. 20.

Bishop, C. M., 2006. *Pattern Recognition and Machine Learning.* Cambridge: Springer.

Cun, Y. L., 2010. *LeNet-5, convolutional neural networks.* [Online]   
Available at: http://yann.lecun.com/exdb/lenet/  
[Accessed 15 March 2019].

Dave Anderson, G. M., 1992. *Artifical Neural Networks Technology.* New York: s.n.

Denton, E., 2015. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. *Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks,* 1(1), p. 9.

Dertat, A., 2017. *Applied Deep Learning - Part 4: Convolutional Neural Networks.* [Online]   
Available at: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2  
[Accessed 8 December 2018].

Dez Song, K. G., 2017. *Automated Collaborative Observatory for Natural Enviroments.* [Online]   
Available at: http://telerobot.cs.tamu.edu/cone/acone/  
[Accessed 7 November 2018].

elitedatascience, 2017. *Overfitting in Machine Learning: What It Is and How to Prevent It.* [Online]   
Available at: https://elitedatascience.com/overfitting-in-machine-learning  
[Accessed 8 December 2018].

Flask, 2019. *Flask.* [Online]   
Available at: http://flask.pocoo.org/  
[Accessed 21 April 2019].

Fyfe, C., 2000. *Artificial Neural Networks and Information Theory.* 1.2 ed. Paisley: s.n.

Ganguly, K., 2017. *Learning Generative Adversarial Networks.* 1st ed. London: Packt.

Gauthier, J., 2015. Conditional generative adversarial nets for convolutional face generation. *Conditional generative adversarial nets for convolutional face generation,* 1(1), p. 9.

Gleerup, K., 2017. *Identifying Pain Behaviors in Dairy Cattle.* [Online]   
Available at: https://www.researchgate.net/publication/317400878\_Identifying\_Pain\_Behaviors\_in\_Dairy\_Cattle  
[Accessed 25 10 2018].

Gong, B., 2009. *A Novel Learning Algorithm of Back-Propagation Neural Network,* Harbin: s.n.

Goodfellow, I., 2016. NIPS 2016 Tutorial : Generative Adversarial Networks. *NIPS 2016 Tutorial : Generative Adversarial Networks,* I(1), p. 57.

Goodfellow, I. J., 2014. Generative Adversarial Nets. *Generative Adversarial Nets,* 10 June.p. 9.

Graupe, D., 2007. *Principles of Atificial Neural Networks.* 2nd ed. s.l.:World Scientific Publishing.

Gulrajani, I., 2017. Improved Training of Wasserstein GANs. *Improved Training of Wasserstein GANs,* 1(1), p. 20.

Hinton, G., 2014. *Boltzmann Machines.* New York: Springer Science Media.

Honglak Lee, R. G. R. R. a. A. Y. N., 2011. Unsupervised Learning of Hierarchial Representations with Convolutional Deep Belief Networks. *Unsupervised Learning of Hierarchial Representations with Convolutional Deep Belief Networks,* p. 9.

Huang, H., 2018. An Introduction to Image Synthesis with Generative Adversarial Nets. *An Introduction to Image Synthesis with Generative Adversarial Nets,* I(1), p. 17.

Karim, R., 2016. *Deep Learning via Multilayer Perceptron Classifier.* [Online]   
Available at: https://dzone.com/articles/deep-learning-via-multilayer-perceptron-classifier  
[Accessed 8 December 2018].

Keras, n.d. [Online]   
Available at: https://keras.io/

Kishan Mehrotra, C. K. M. S. R., 2000. *Elements of Artificial Neural Networks.* Massacusetts: Second Printing.

Larissa Ferreira Rodrigues, M. C. N. J. F. M., 2017. HEp-2 cell image classification based on. *HEp-2 cell image classification based on,* p. 6.

Lehr, B. W. a. M. A., 1993. *Artificial Neural Networks of the perceptron, Madaline and backpropagation alglorithm.* California: Elesevier Science Publishers.

Mohak Srivastava, S. S. C. G., 2018. Comparison of optimizers implemented in Generative. *Comparison of optimizers implemented in Generative,* 119(special), p. 5.

O’Shea, K., 2015. An Introduction to Convolutional Neural Networks. *An Introduction to Convolutional Neural Networks,* 1(1), p. 11.

Olivier Michel, A. H. F. C., 2018. *Single-Layer Perceptron Neural Networks.* [Online]   
Available at: https://lcn.epfl.ch/tutorial/english/perceptron/html/intro.html  
[Accessed 7 October 2018].

Prada, J. C. B., 2018. *Hello, Gradient Descent.* [Online]   
Available at: http://blog.datumbox.com/wp-content/uploads/2013/10/gradient-descent.png  
[Accessed 7 December 2018].

Qiao, Y., 2016. Fast Automatic Step Size Estimation for Gradient Descent Optimization of Image Registration. *Fast Automatic Step Size Estimation for Gradient Descent Optimization of Image Registration,* 35(2), p. 12.

R. Vaillant, C. Y. L. C., 1994. Original approach for the localisation of objects in images. *Original approach for the localisation of objects in images,* p. 6.

R.Sathya, A. A., 2013. Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification,* 2(2), p. 4.

Radhakrishnan, P., 2018. *Generating Images from Brain Signals.* [Online]   
Available at: https://hackernoon.com/generating-images-from-brain-signals-a286cb002aa7  
[Accessed 16 January 2019].

Ricard, R., 2017. *Generative Adversarial Networks Part 1 - Understanding GANs.* [Online]   
Available at: http://www.rricard.me/machine/learning/generative/adversarial/networks/2017/04/05/gans-part1.html  
[Accessed 7 December 2018].

Roell, J., 2017. *From Fiction to Reality: A Beginner’s Guide to Artificial Neural Networks.* [Online]   
Available at: https://towardsdatascience.com/from-fiction-to-reality-a-beginners-guide-to-artificial-neural-networks-d0411777571b  
[Accessed 8 December 2018].

Saha, S., 2018. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way.* [Online]   
Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53  
[Accessed 30 April 2019].

Santosh Kumar, S. K. S., 2016. Visual animal biometrics: survey. *Visual animal biometrics: survey,* 6(3), p. 14.

Stutz, D., 2014. Understanding Convolutional Neural Networks. *Understanding Convolutional Neural Networks,* 1(1), p. 23.

Tensorflow, n.d. [Online]   
Available at: https://www.tensorflow.org/

Tensorflow, n.d. [Online]   
Available at: https://www.tensorflow.org/api\_docs/python/tf/trainable\_variables

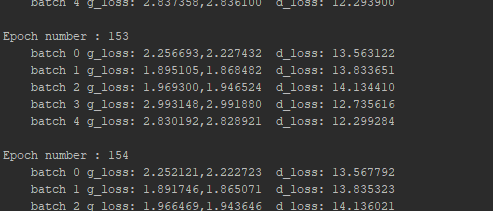
Thanh-Tung, H., 2018. On catastrophic forgetting and mode collapse in Generative Adversarial. *On catastrophic forgetting and mode collapse in Generative Adversarial,* 1(1), p. 10.

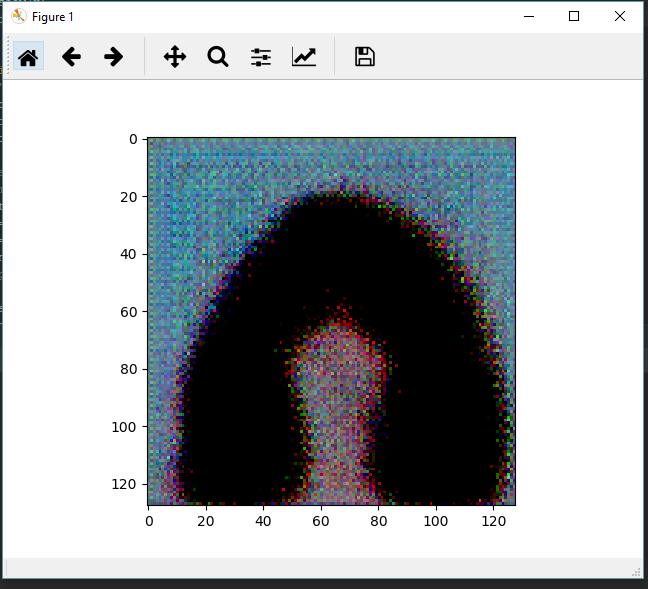
Timothy J. Jassmann, R. T. a. R. M. P., 2015. Leaf Classification Utilizing a Convolutional Neural. *Leaf Classification Utilizing a Convolutional Neural,* p. 3.

Yan Le Cun, Y. B., n.d. Convolutional Neural Networks for Images, Speech, and Time-Series. *Convolutional Neural Networks for Images, Speech, and Time-Series,* p. 14.

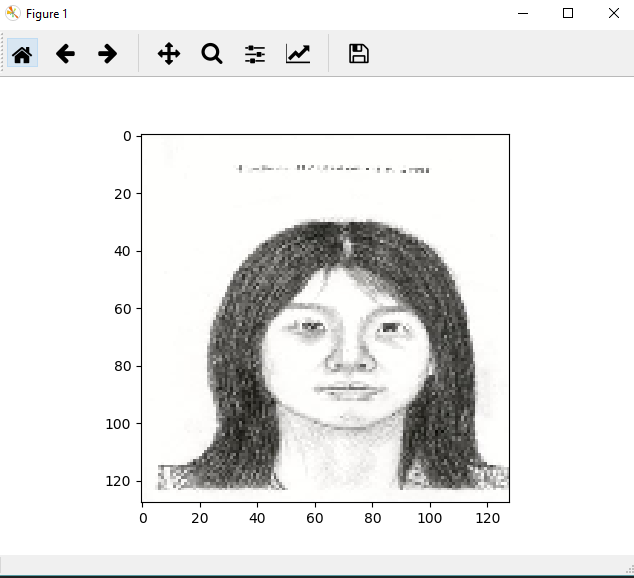
Yliopisto, T., 2017. [Online]   
Available at: http://www.uta.fi/sis/tie/neuro/index/Neurocomputing2.pdf  
[Accessed 5 10 2018].

# Appendix





Current Tests – 600 Epochs



Test Sketch Given