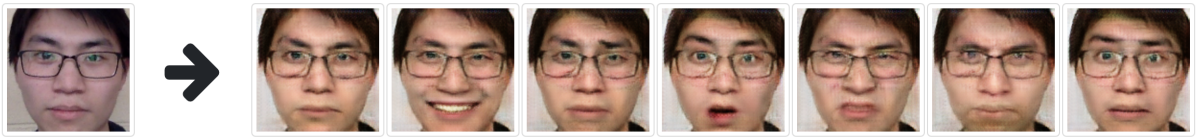
MANCHESTER 1824
The University of Manchester

Using Generative Adversarial Networks (GANs) for Facial Expression Generation

Third Year Project Report



Author:
Tim, Zhenzhong Xiao

Supervisor:
Dr. Tim Morris

Degree Program:
BSc (Hons) Computer Science with Industrial Experience

School of Computer Science

The University of Manchester

May 1, 2018

Abstract

The project experimented and created a facial expression modifying method, which is based on a novel deep learning architecture called Generative Adversarial Networks (GANs). The final result of this project is a model that takes a face image and generates a new image with the selected facial expression while keeping the identity of the subject in the input unchanged. The report starts with an introduction of the project followed by explanations for the underlining algorithms. Then, in the dominant part of the report, a number of important experiments are covered as the illustration of how the project progressed and reached its goal. In the end, the evaluation of the results is given, as well as the reflection and achievements.

Acknowledgements

I would like to express my gratitude to Dr. Tim Morris for his support and guidance throughout the year. I would also like to thank my family and friends for providing me help and company whenever I need it.

Contents

Chapter 1

Introduction

This project is to create a facial expression modifier using a deep learning method called Generative Adversarial Networks (GANs). The modifier is based on a variation of GANs trained using images of different facial expressions. It will read a face image and ideally produce a new image with selected expressions. At the same time, the identity of the face should be preserved. The following sections of this chapter will mention the motivation of the project, the deliverables, a summary of previous works and the report structure.

1.1 Motivation

Although facial expression editing has been achieved before with methods not involving deep learning, recent success in deep learning encourages us to explore how deep neural networks would perform similar tasks. In particular, GANs is a recently proposed deep learning method, which has many unknown properties. By applying it to various problems such as expression synthesis, we may be able to discover its new features and understand it better. Furthermore, the new method might be able to generate better results than the old ones.

Apart from the temptation of trying the new method, facial expression editing is an interesting and practical problem in itself. The ability to generate different expressions from an input image will allow us to predict emotions and help with facial recognition. Moreover, the ability of generating facial expression image with labels is very useful for training an emotion recognition system. Last but not least, editing facial expression is similar to applying filters in social applications like Snapchat and Instagram, it can be used for entertainment.

1.2 Deliverables

The initial plan of the project included three milestones, and each of them can be a potential deliverable of the project. The goal was to finish the first two and try to work towards the last one if the time allows. They are:

- Given a single face image of neutral expression, the model is able to generate an image of happy expression.

- Given a single face image of neutral expression, the model is able to generate an image of selected expression, which includes happy, sad, surprised, disgusted, angry and fearful.

- Given a single face image of any expression, the model is able to generate a series of images in the selected expression with intensities from weak to strong.

At the end of the project, the first two milestones and part of the last one have been achieved. The resulting model is able to generate images of selected expressions (including neutral, happy, sad, surprised, disgusted, angry and fearful) given a single face image of any expressions.

1.3 Related Work

Previous methods for generating facial expressions often adapt traditional computer graphic techniques. One of the popular methods is called expression mapping, which requires an image of neutral expression and an image of another expression from the same subject [30]. Facial features such as the mouth, eyebrows are located on both images either manually or automatically. By calculating the difference vector of two sets of feature positions, we can add the difference vector onto features of a new face image, which will generate the target expression of that face. Other methods also involve manipulating expressions in 3D space. Blanz proposed a method that first trains a 3D model by calculating the difference vector of facial expression in 3D space, then reconstructs the 3D face of the 2D input image, modifies the expression in 3D, and finally renders back to 2D to get the generated image [4]. Reasonably good results can be achieved by these methods. However, most of them require facial features in the training data to be labelled, which normally includes complex procedures and intensive image preprocessing.

1.4 Report Structure

The report is organised as follows: In chapter 2, a detail explanation of various concepts used in the project will be given, such as CNN, GANs, and Autoencoder. As the methods used in the project are relatively new (proposed within 4 years), there is only limited documentation online besides the papers that proposed the methods. Moreover, similar work has not been done by many people before, and there was no existing open-source implementation online. Therefore, this project involves a huge amount of experiments on reproducing and modifying methods from different papers. Chapter 3 will go through a list of important experiments in chronological order and the differences between each method, which can provide a general idea of how the model evolved as the project progressed. Evaluation of the final method will be covered in chapter 4, followed by reflection and achievements of the project in chapter 5.

Chapter 2

Basic Components

To achieve the goal of the project, three important building blocks are needed, and they will be mentioned frequently in the next chapter.

2.1 Convolutional Neural Network (CNN)

Overview

Convolutional Neural Network (CNN) is a type of deep neural networks designed for handling images [17]. It was first proposed by Yann LeCun in 1998 using an example called LeNet-5, which is able to classify images of digits and was deployed in the industry to recognise the handwritten number on checks. In recent years, as the computing power increased, CNN has been successfully applied to various problems such as video analysis and more complicated image recognition tasks.

Architecture

The design of CNN is based on regular multilayer neural networks, which means it is a collection of connected neurons that have trainable weights and biases. These weights and biases are trained using backpropagation algorithm. The difference between CNN and the regular neural network is mainly in their architectures. In a regular neural network, each neuron of a layer is connected with all neurons in the two neighbouring layers as shown in Figure 2.1. But in CNN, most of the neurons are only connected with a small number of neurons from adjacent layers (e.g. Figure 2.2), the layer formed by such neurons is called Convolutional Layer.

In addition, convolutional layers have a third dimension compared to the layers of a regular neural network. This dimension, normally denoted as channels (or depth), represents the number of feature maps (or activation maps) in the layer. As an example, the first hidden layer in Figure 2.3 is a convolutional layer with channel size 3. In other words, this layer has 3 feature maps, each of them is similar to the first hidden layer in Figure 2.2. A more intuitive way to understand the convolutional layer is to compare it with the convolution operation in image processing. Assume we have an image of size $n \times n$ and a kernel (or filter) of size $m \times m$, by sliding through the image using the kernel, we can calculate the value of each pixel in the feature maps from the weighted sum of the corresponding image patch covered by the kernel. The number of channels C can be interpreted as C different kernels are used for calculating C feature maps in one convolutional layer. However, unlike

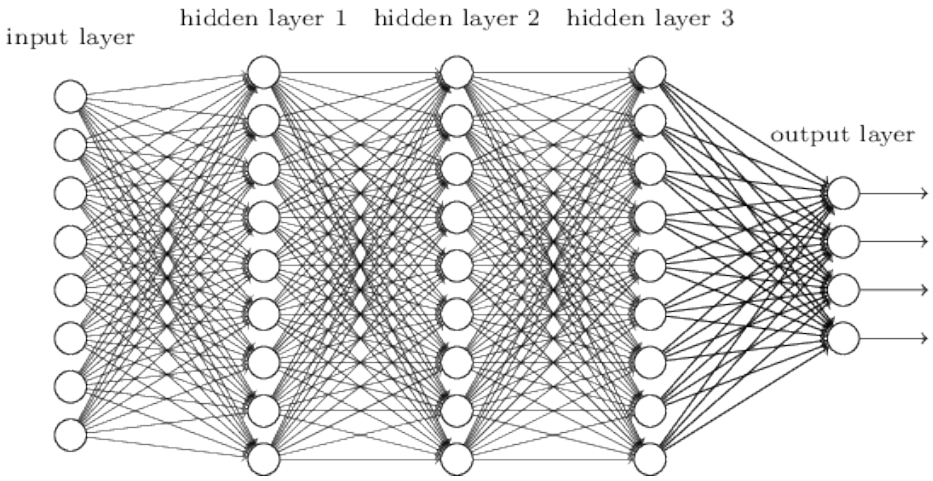


Figure 2.1: A five-layers regular neural network [25].

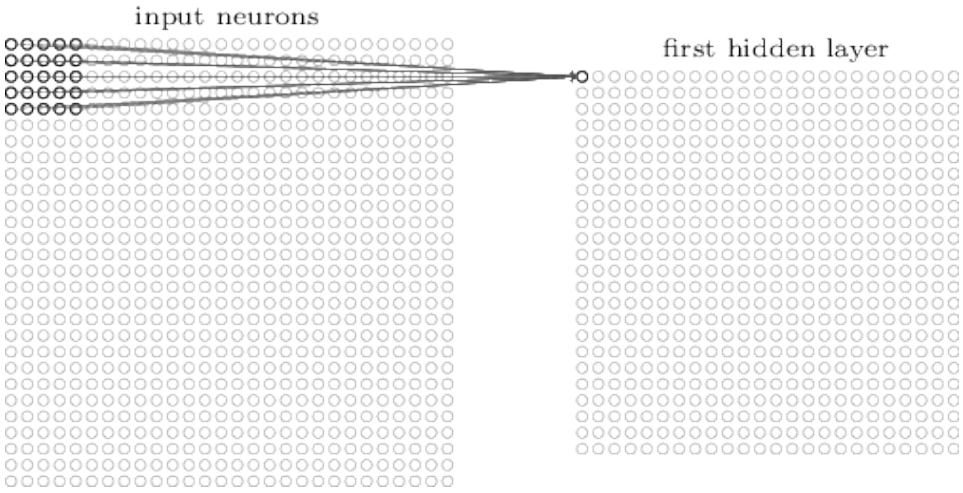


Figure 2.2: Connections of a convolutional layer [25].

image processing, the kernels used here are learned by the network itself rather than defined manually by the human.

The rest of CNN is simply a stack of various convolutional layers with downsampling (Pooling Layer) operations and non-linear functions[1] (e.g. ReLU, Sigmoid etc.) followed by layers from the regular neural network (Fully-Connected Layer). A sample CNN architecture is shown in Figure 2.4.

2.2 Generative Adversarial Networks (GANs)

Overview

Generative Adversarial Networks (GANs) is a generative model proposed by Ian Goodfellow in 2014 [10]. The generative model can be defined as: given a set of training data, which are samples of a probability distribution p_{data}, the model is trained to learn from the training set and estimate p_{data} using the learned probability distribution p_{model} [9]; if we can find a

[1] Also known as the activation function, which is to impose nonlinearity on the network. As it is not the main focus of the project, details are omitted here. More information can be found *online*.

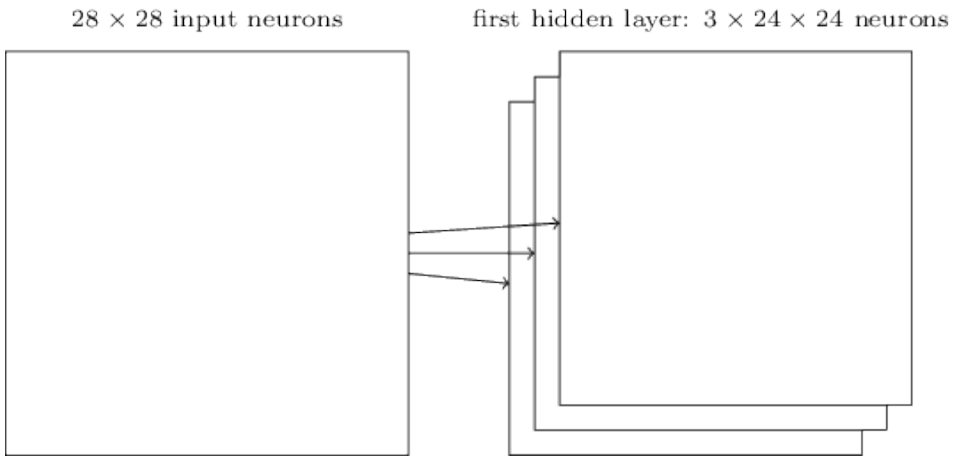


Figure 2.3: The convolutional layer that takes an input of size (**H**eight=28, **W**idth=28, **C**hannels=1), and produces an output of size (H=24, W=24, C=3) [25].

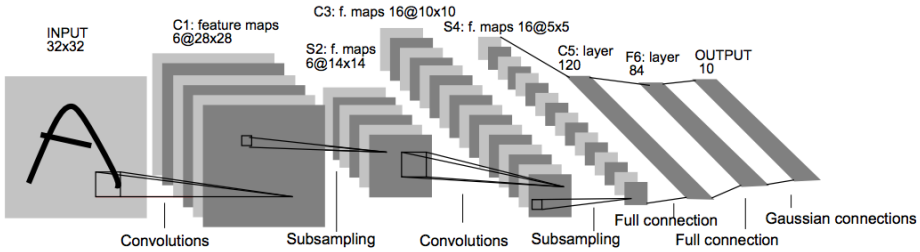


Figure 2.4: Architecture of LeNet, proposed in the paper that introduced CNN [17].

p_{model} that perfectly estimate p_{data} (i.e. $p_{\text{model}} = p_{\text{data}}$), theoretically we can generate data statistically indistinguishable from the real data. GANs learn p_{model} implicitly and generate sample directly from p_{model} as the output of the model.

The training process of GANs is what makes it unique and exciting, which is based on an adversarial framework. The model consists of two components: a generator G and a discriminator D. During the training process, G tries to learn a distribution p_{model} for generating data that cannot be discriminated with real data by D. At the same time, D needs to learn a probability distribution to distinguish real and fake data so that for any given data, D can classify it with high accuracy. Since G tries to fool D with fake data, and D attempts to identify deceptions from G, in a way, D and G are fighting against each other (e.g. Figure 2.5). In game theory, this process also known as the zero-sum game (or minimax game).

Training details

To describe the model formally, we denote the latent variable (sampled from a known distribution p_{z}) used for generating data as z, and the real data (came from the distribution p_{data}) examined by the discriminator as x. In this case, the generator is a function G with parameters θ_{G}, which takes z as input and produces $G(z)$. Similarly, the discriminator can be defined as a function D with parameters θ_{D}, which takes x or $G(z)$ and results in $D(x)$ or $D(G(z))$ — the probability of the given data being real. The overall structure is shown in Figure 2.6.

In general, both generator and discriminator are realised using neural networks. To train

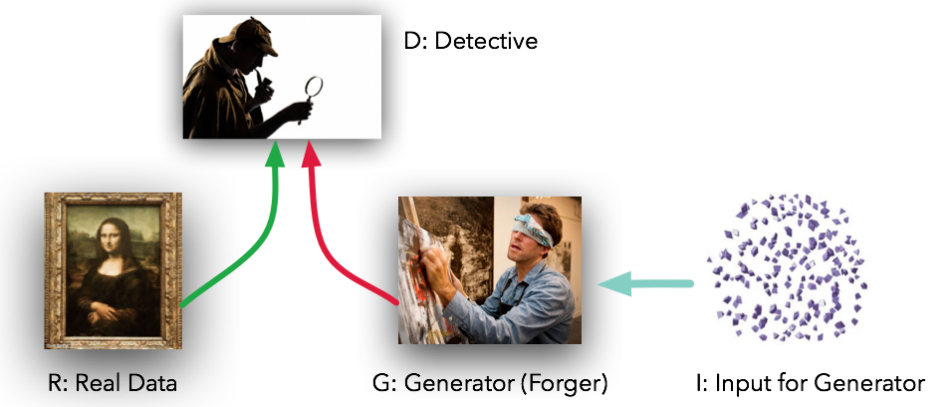


Figure 2.5: This diagram shows an analogy of GANs. The generator can be thought of as a forger, who tries to forge paintings. And the discriminator is like a detective that needs to identify the counterfeit and the authentic work. Ideally, increasing amount of training circles will drive both of them to improve their methods to such stage that the forger being able to forge counterfeits that are indistinguishable from authentic paintings, and the detective being unable to differentiate between the two. [24].

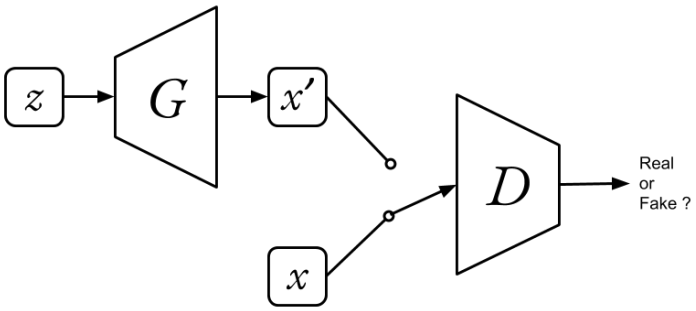


Figure 2.6: An illustration of GANs. D switches input between x' and x.

a neural network towards a specific task, we need to define an objective for the training process. Here, in GANs, we have a minimax objective:

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))].$$

Keeping θ_{G} unchanged, for D, we would like to maximise the objective function. Because for real data x, the probability that D predicts x to be real should be as close to *1* as possible, and the probability that D predicts $G(z)$ to be real should be as close to *0* as possible. Simultaneously, by keeping θ_{D} unchanged, we want to minimise the objective function for G. As the first half of the function is unrelated to G, it is the same as to minimise the second half of the function. That is to say, given z from distribution p_{z}, the expectation of log-likelihood that D predicts $G(z)$ as fake should be minimised.

Training a model is essentially optimising the objective function. A standard optimisation method is the stochastic gradient descent (SGD). Gradient descent finds a minimum value of a function by calculating the gradient of the function at the current point, then take a step towards the opposite direction of the gradient. By repeating the process, it will reach a local minimum point of the function. Adam, a variation of SGD, is often used in GANs, which

has been proven to be a good choice [15]. The whole process is summarised in Algorithm 1.

Algorithm 1 The training procedure for GANs [10].

for each training iteration **do**

- Sample a batch of m noise vectors $\{z^{(1)}, \ldots, z^{(m)}\}$ from p_{z};
- Sample a batch of m real data $\{x^{(1)}, \ldots, x^{(m)}\}$ from the training set;
- Update D by ascending its stochastic gradient:

$$\nabla_{\theta_{\mathrm{D}}} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\Big(x^{(i)}\Big) + \log \Big(1 - D\big(G(z^{(i)})\big)\Big) \right];$$

- Update G by descending its stochastic gradient:

$$\nabla_{\theta_{\mathrm{G}}} \frac{1}{m} \sum_{i=1}^{m} \log \Big(1 - D\big(G(z^{(i)})\big)\Big);$$

end for

The gradient updates can be done in any gradient based learning method. Most experiments in the project used Adam [15].

2.3 Autoencoder

Figure 2.7 illustrates the architecture of autoencoder. It mainly has two components, an encoder and a decoder. Both of them are neural networks, one with a decreasing number of neurons in each layer, the other with an increasing number of neurons in each layer. The result of this is that the encoder compresses (encodes) the input into a smaller size representation of the data (latent space representation, hidden variable). Then, the decoder generates new data from the latent space. The objective of autoencoder in the context of image generation is to generate a new image from the input image, such that the difference between the generated image and the target image is minimised.

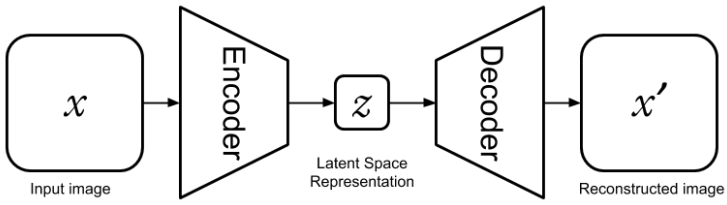


Figure 2.7: Structure of Autoencoder.

Chapter 3

Experiments

The main part of this project is to find a model that can realise the goal. This chapter records the evolution of the project. Each section describes a model that provided the solution to the problem of the previous model and outlines the new issue occurred. The final section of the chapter presents the model used currently as the ultimate solution to the project goal.

3.1 Environment Setup

Training a deep neural network requires extensive computing power. The machine used for this project has an NVIDIA GTX 1080 graphic card, which was a popular computing device for personal deep learning research. It provides more than 70 times speed up for training a deep neural network than CPU [2]. The operating system of the machine is Ubuntu 16.04. For the machine learning framework, I could choose between TensorFlow and PyTorch. After investigated online, although PyTorch seems to be more friendly to new users, TensorFlow has a larger user community. It means that if I encounter problems using TensorFlow, it is more likely to find an existing answer online comparing to PyTorch. Therefore, I chose TensorFlow[1] and its underlining language Python[2] for the project.

In order to do the experiment remotely, I also setup Jupyter Notebook server on the machine. By doing so, I can create and run my models on any machine that connects to the Internet and has a browser. Moreover, Git is used for tracking the history of codes.

3.2 Noise to Number

The original GANs paper illustrates the power of GANs by generating highly recognisable handwritten digits from random noises. Creating the similar result using GANs is a reasonable starting point. The dataset used for this part of the experiments is MNIST [17], which is used in almost all papers that involve experiments of handwritten digits. Each image of a digit is in the size 28×28.

[1] TensorFlow version: 1.3.0
[2] Python version: 3.5.2

3.2.1 DCGAN

Architecture

Although the initial GANs that utilises the regular neural network is able to generate fine small images, it has difficulties in handling large images [6]. DCGAN, short for Deep Convolutional GANs, is proposed to solve such problem [27]. The major modification is replacing the regular neural networks with convolutional networks for both discrimination and generator. As the result, the discriminator is now a CNN, and the generator has a structure close to a reversed CNN (e.g. Figure 3.1) which is achieved by deconvolution (transposed convolution) [29].

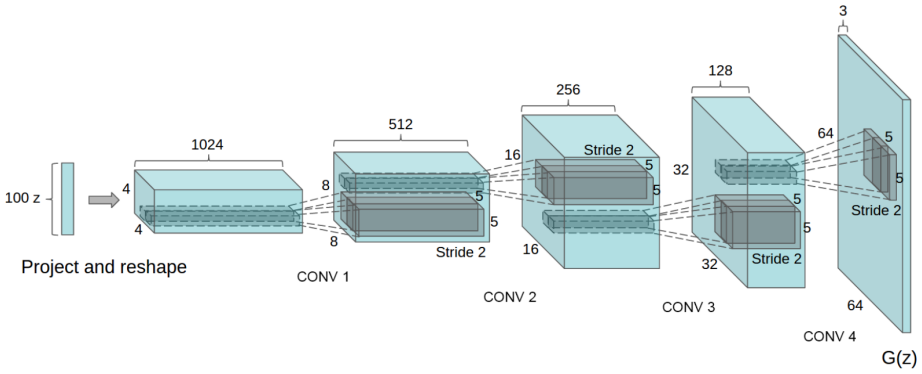


Figure 3.1: Structure of a deconvolutional network, used as the generator of a DCGAN [27].

Convolutional layer (conv layer) normally turns a large size input into a set of smaller size feature maps as mentioned in the previous chapter. Conversely, deconvolutional layer (deconv layer) is created to have the opposite effect that turns a small size input into a set of larger size feature maps. This is done by padding the input with zeros so that the size of input data is larger than the size of an output feature map. Then, we can apply the normal convolutional layer to get the required output as shown in Figure 3.2.

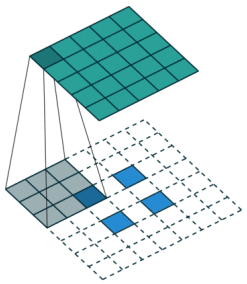


Figure 3.2: In this example, a 2x2 input is deconvolved into a 5x5 output through a deconvolution operation [7].

Another improvement is that DCGAN utilises the batch normalisation [13] technique (BN/batch norm) between the layers of discriminator and generator, which normalises the output of a layer for each training mini-batch. This technique allows us to use a higher learning rate for training the network. It means the network will converge faster and the training process will take less time.

Experiment

The model in this experiment was implemented in the way that the discriminator has three convolutional layers followed by a fully connected layer. All of the convolutional layers use kernels of size 4×4 and leaky ReLU [21] as the activation function. Batch normalisation was applied to the last two convolutional layers. For the fully connected layer, the activation function is sigmoid, since the output of sigmoid has a value between 0 and 1, which can be

interpreted as the probability of the input being real data. As for the generator, it consists of one fully connected layer and four deconvolutional layers. The first four layers apply batch norm, activation function ReLU and kernels of size 5×5. The output layer uses hyperbolic tangent (TanH) as the activation function, which puts data into range -1 to 1. The input noise has the size 100. In addition, since the image of the size 2^n is easier to handle, digit images are padded with zeros so that the input to the discriminator is in size 32×32 rather than 28×28.

Figure 3.3: Samples generated by the model as the training progressed.

Without any surprise, the results are mostly satisfying as shown in Figure 3.3, and they have the similar quality as those in the paper. However, the goal of the project includes being able to create an expression based on a given condition (e.g. happy, surprised, neutral). Therefore, in the context of digits, the model needs to generate the corresponding digit given the number as a condition.

3.2.2 Conditional DCGAN

Architecture

Conditional GANs was realised shortly after the creation of GANs in 2014 [23]. To construct a conditional GANs, we need to concatenate the condition y with the input of discriminator and generator (i.e. Figure 3.4). y is normally represented as a one-hot encoded vector. The rest of the training process is exactly the same.

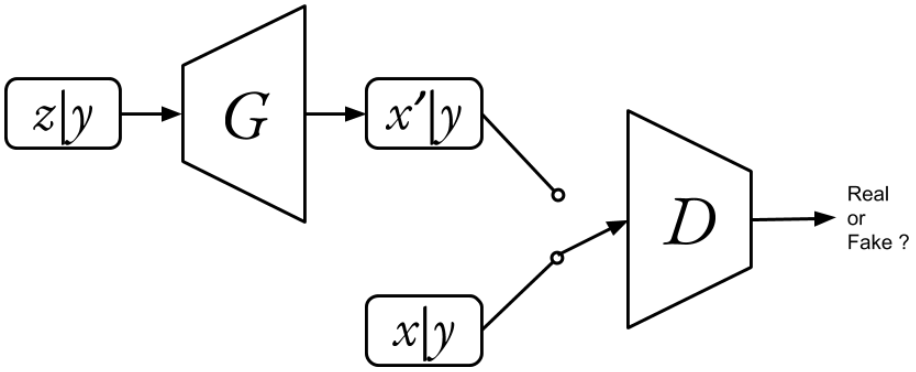


Figure 3.4: Structure of conditional GANs.

To adapt condition GANs to DCGAN, we have to use some tricks for the input data of discriminator, because to concatenate the condition y (a vector) with an image (a matrix) is not intuitive. Assume y has the size n, and the image $\mathcal{I}$ has the size $h \times w$. One way to do it is to: first, treat y as a list of n 1×1 images; then, upsample each 1×1 image to the

size $h \times w$ by duplicating its value $h \cdot w$ times. In this case, we have a list of n $h \times w$ images, which can be concatenated with the image $\mathcal{I}$ as channels [26]. Therefore, the data that will be put into discriminator is a volume with height h, width w and channels $n + 1$.

Experiment

Followed by some simple modifications of the previous DCGAN model, good results were received. Figure 3.5 shows a set of experiment samples. It is clear that the model is able to produce the specified digits. An interesting finding from the results is that the various digits generated using the same noise have the same handwriting style. This finding might imply that a good latent space representation of a face image can help the model to generate different expressions of the same identity.

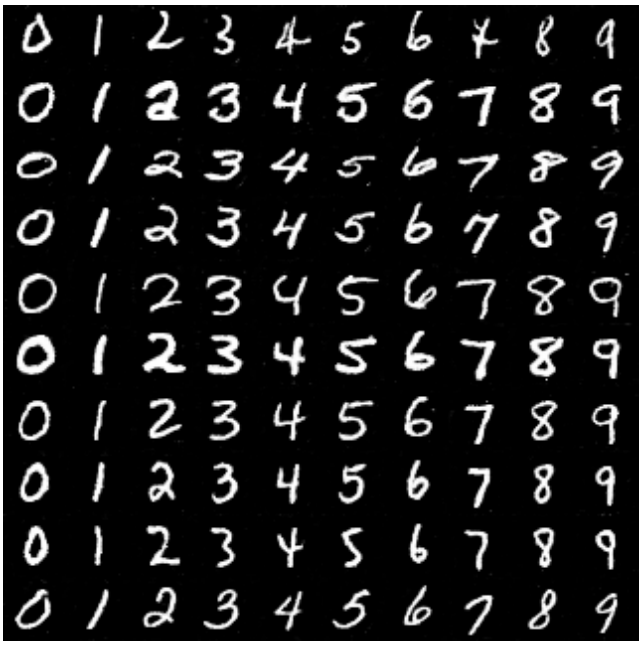


Figure 3.5: Samples from conditional DCGAN. Each row is generated using the same noise, but with different conditions from 0 to 9.

3.3 Noise to Face

Since the experiments in numbers are quite successful, the next step will be to apply the model to the task of generating face images. An obvious difference between face images and MNIST digits is that an identifiable face image requires higher resolution than MNIST digits. Although it sounds trivial, this difference revealed a crucial problem in the last model.

The face image datasets used in this section are CK+ [14, 19] and CelebA [18]. Images in CK+ are in greyscale and of the size 640×490 with expression labels. Although the quality of the images from CK+ is inadequate, the reason for using it at the beginning is that it is easy to download, and does not require manual permission from the owner. CelebA is a popular dataset for computer vision tasks that related to face images. It has over 200k celebrity images. In this project, CelebA dataset is used as the training data for generating face images without expression conditions, as well as providing testing data for models to generate selected expressions.

3.3.1 DCGAN

Architecture

The same model was used from the last section with no changes in architecture.

Experiment

In order to reuse the previous model directly, images for CK+ needed to be preprocessed before fed into the algorithm. Steps are as follows: OpenCV library is used to detect the face in the image; then the face region is cropped out as a new image; finally, it is resampled into the required size. Here, face images from CK+ was preprocessed into size 28×28, identical to the size of MNIST data. After trained using the new data, the model was able to produce reasonable results as shown in Figure 3.6.

Figure 3.6: Samples of size 28×28 generated using DCGAN trained by CK+ dataset.

Next, the model was tested under higher resolution images. The training data was preprocessed into size 64x64 before fed into the model. However, the model collapsed and was unable to produce stable face images. From Figure 3.7, we can see that the result generated by the model was either unsharp and with an unidentifiable face or simply meaningless.

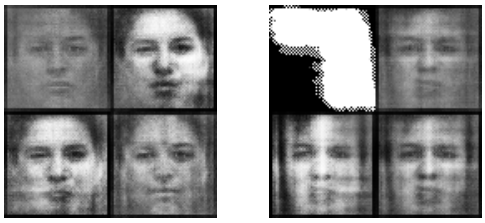


Figure 3.7: Samples of size 64×64 generated using DCGAN trained by CK+ dataset.

After investigations, it turned out the issue inherits from the objective function of GANs, which causes GANs hard to train in most situations. This problem is called mode collapsing. It means that the generator has found a way to fool the discriminator using a similar image, therefore, no further improvement will be made by the generator. The underlining problem of the objective function can be understood using knowledge from probability theory. It was proposed in the original GANs paper that to optimise the objective function is similar to minimizing the Jensen-Shannon divergence[3] between the distribution p_{data} and the distribution p_{model} [10]. Jensen-Shannon divergence is a method used in probability theory for measuring the likeness of two distributions. And a later paper argues the Jensen-Shannon divergence that GANs tries to minimise might not be continuous for the parameters of the generator, which is what makes GANs difficult to train [1].

3.3.2 WGAN

Architecture

A similar model from the last section is used but with a different objective function. The new objective function was proposed by the same paper that has identified the root cause of the training difficulty for GANs. Instead of using the initial Jensen-Shannon divergence to

[3]Details are not covered in this report as they involve advanced materials from probability theory. More information can be found *online*

calculate the distance between p_{data} and p_{model}, the paper states that Wasserstein distance is a better measure of such problem [1]. The improved model is called Wasserstein GANs (WGAN).

Wasserstein distance can be understood intuitively as the minimum cost of rearranging the mass of one distribution so that it will result in the same distribution as the other, and the cost will be the changed mass times the corresponding moved distance. In computer science, it is also known as the Earth Mover distance because of the above analogy. Here, the Wasserstein distance is continuous and differentiable nearly everywhere under some trivial assumptions [11]. The authors of the paper construct the improved object function as:

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}}\big[D(x)\big] - \mathbb{E}_{z\sim p_z}\big[D\big(G(z)\big)\big],$$

and in order to enforce the assumptions used for approximating this objective function[4], the parameters of the discriminator has to be clipped to lie in a small range $[-c, c]$ [1].

Experiment

Simply adapting the Wasserstein distance as the new objective function and the weight clipping in the training process, we will receive the new model from the previous one. The result is shown in Figure 3.8. It has much better faces compared to the previous outcome, and most importantly, the model did not collapse for the same training data. However, as we can see from the samples, many of them are still in low quality or even have frightfully strange appearances. If the goal of this project was to generate zombie-like images, this model should have already achieved it.

Figure 3.8: Samples of size 64×64 generated using DCGAN with Wasserstein distance trained by CK+ dataset.

[4]The derivation of this objective function and its approximation involve extensive proofs that require knowledge of advanced mathematics. It is impossible for this report to cover the details of the proofs to the extent that a normal computer science undergraduate can understand. Therefore, the details are omitted. (Some example concepts used in the proofs: Kantorovich-Rubinstein duality, Lipschitz continuity)

3.3.3 BEGAN

Architecture

A variation of GANs that is specialised in generating high-quality high-resolution sharp image was published nearly a year ago [3]. It is called Boundary Equilibrium GANs (BEGAN). The samples from the paper look very promising, and it seems to be able to produce much better images than the model from the last section. The major differences between BEGAN and the model from the previous section are the objective function and the architecture of the discriminator. Rather than using a CNN as the discriminator, BEGAN makes use of an Autoencoder instead (the structure is shown in Figure 3.9). Therefore, the objective of the discriminator is not to predict whether an input is real or fake anymore.

The new discriminator is trained to minimise the difference between the real images and their reconstructions, and to maximise the difference between the generated images and their reconstructions. We called the difference between the input and the reconstruction as the loss $\mathcal{L}$ (or the reconstruction loss) of an autoencoder; and the measurement of the difference as the loss function. Thus, the generator is now trained to match the loss distributions of real images and generated images, while originally it is trained to match the distribution of p_{data} and p_{model}. Moreover, Wasserstein distance from WGAN remains as the measurement for the difference of two distributions.

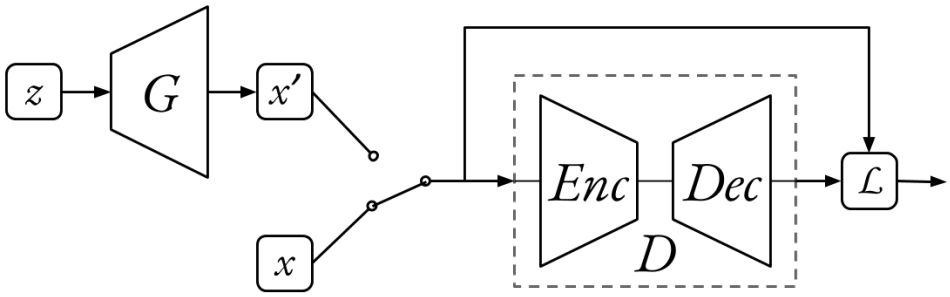


Figure 3.9: Structure of BEGAN. $\mathcal{L}$ is calculated by measuring the difference between the input and the output of the discriminator D.

In addition, the paper introduces a parameter γ [3] for controlling the equilibrium between the generator and the discriminator. Ideally, if the generated images are very similar to the real images for the discriminator, we will have:

$$\mathbb{E}\big[\mathcal{L}(x)\big] = \mathbb{E}\big[\mathcal{L}\big(G(z)\big)\big],$$

which means the expected loss of the real image is equal to the expected loss of the generated image. By applying:

$$\gamma = \frac{\mathbb{E}\big[\mathcal{L}\big(G(z)\big)\big]}{\mathbb{E}\big[\mathcal{L}(x)\big]} \qquad \gamma \in (0, 1), \tag{$*$}$$

we can adjust the focus of the discriminator between reconstructing real images and differentiating inputs. For example, if we change γ towards 0, the model will be generating images with lower diversity as the focus of the discriminator is on autoencoding real images, but at the same time, the resulting images will be sharper and have more details.

For the changes above, the paper derived and formulates the new objective, which is to

minimise the following losses of both generator and discriminator:

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x) - k_t \cdot \mathcal{L}(G(z)) & \text{for } \theta_D \\ \mathcal{L}_G = \mathcal{L}(G(z)) & \text{for } \theta_G \\ k_{t+1} = k_t + \lambda_k(\gamma \cdot \mathcal{L}(x) - \mathcal{L}(G(z))) & \text{for each training step } t \end{cases}$$

Here, k is calculated in each step and feed towards the next step in order to maintain the equation $(*)$. λ is the learning rate for k.

Experiment

By changing the architecture of the discriminator from the previous section, and updating the objective function, we will have BEGAN. Since the experiments had resulted in good quality images, images with colour were brought in for testing. The samples shown in Figure 3.10 were generated by the model trained using the CelebA dataset, which was first preprocessed into images of size 64×64. Without a doubt, the new samples are sharper and clearer than all others from above.

Figure 3.10: Samples of size 64×64 generated using BEGAN trained with CelebA dataset

3.4 Face to Face

After previous sections, the current model is able to generate high-quality images from noises. Now is time to consider how to generate images from images, as well as adding back the conditions into the model.

Since we need to include conditions of the expression, the training data used in this part are all front facing face images with labelled expressions. The training set is a combination of 3 datasets: FACES [8], RaFD [16] and KDEF [20]. Totally, there are about 1200 subjects, and each has 7 labelled coloured images with expressions: neutral, happy, sad, surprised, disgusted, angry and fearful. To increase the amount of training data, each image was

duplicated and flipped horizontally. In this case, it is similar to have training data from 2400 subjects. Likewise, all of them are preprocessed into the size of 64×64 and 128×128. The testing set remains CelebA [18].

3.4.1 Conditional BEGAN

Architecture

Building on top of the last architecture, BEGAN, the new generator is changed from a deconvolutional network to an autoencoder, which is able to generate an image from an image. Moreover, conditions are added in a similar fashion as conditional DCGAN (in section 3.2.2). As we are inputting an image rather than a noise vector to the generator, the condition is concatenated with the latent space variable in the middle of the autoencoder. The full structure is illustrated in Figure 3.11. The objective function from BEGAN remains unchanged.

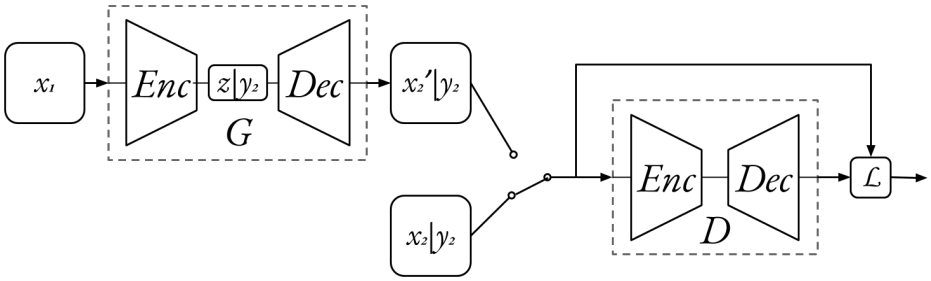


Figure 3.11: Structure of conditional BEGAN. y_2 is the label of x_2. The generator tries to generate x_2 from x_1 by concatenating y_2 with the latent variable z.

Experiment

The result was not as expected. The model seems to suffer again from the issue of mode collapsing because all the generated results are similar. Figure 3.12 shows several samples that were generated from totally different images. Although different parameter settings and slightly altered architectures were tested, none of them has helped. Due to the time constraint of the project, further investigation was not conducted, therefore, the cause of the issue is still unknown. More detail analysis is left to future works.

Figure 3.12: Samples from conditional BEGAN in the size 64×64. Each sample was generated using a totally different image.

3.4.2 Conditional Autoencoder

Architecture

The failure of the last model made me wonder whether I should approach the goal differently starting from a simple autoencoder considering autoencoder is easy to understand and handle. Without inheriting previous models, an autoencoder was created with the concatenation of the conditions in the latent space (i.e. Figure 3.13). The training objective of this new model is to minimise the absolute difference $\mathcal{L}$ between the generated image x_2' of the selected expression y_2 and the real image of that expression x_2, which comes from the same subject as the input. In other words, to minimise $\mathcal{L} = ||x_2 - x_2'||_1$.

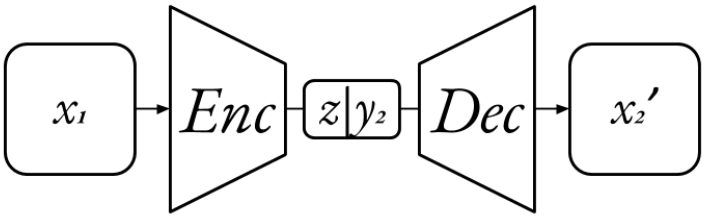
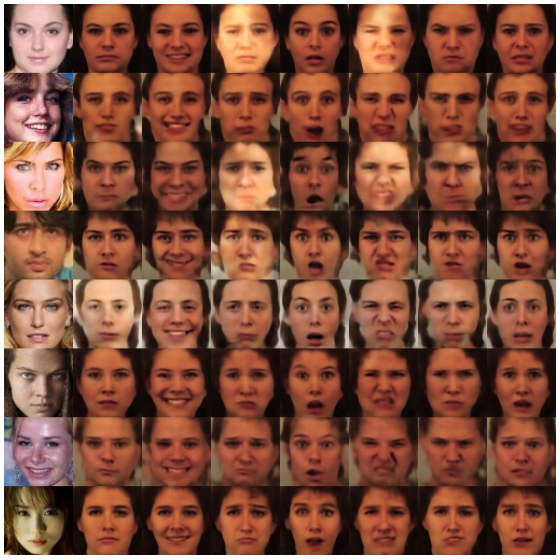


Figure 3.13: Structure of conditional autoencoder. y_2 is the label of x_2. The model tries to reconstruct x_2 using x_1 with y_2 as the condition.

(a) Training samples. The 1st and 4th columns display the input images, 2nd and 5th are generated images, 3rd and 6th are the corresponding targets.



(b) Testing samples. The 1st column displays the input images. The rest of the columns (from left to right) are the generated images under conditions: neutral, happy, sad, surprised, disgusted, angry and fearful.

Figure 3.14: Samples of size 64×64 generated by conditional autoencoder.

Experiment

Figure 3.14 shows a list of samples generated from the training set and the testing set. The quality of the training samples is expected with the use of autoencoder. However, the testing sample does not preserve the identity, though the expression matches the selected condition.

This is an indication that the model was overfitted to the training data, which means it will try to generate images similar to those it has seen.

One possible explanation is: during the training process, the model has found a way to encode the input image, which is tailored to the training data in order to generate the closest image to the target. This learned encoding method may only focus on some specific features of the image, which can help to achieve the defined training objective. Therefore, when the model sees a new image, it will attempt to encode it in a way that is easy to reproduce the training data. An interesting analogy is the saying from Maslow: "If all you have is a hammer, everything looks like a nail"[22].

3.4.3 Additional Feedforward Autoencoder

Architecture

A potential solution was found after researching on the overfitting problem from the last model. A paper claims that by adding an extra feedforward connection from the middle of the encoder to the middle of the decoder (as shown in Figure 3.15), the autoencoder will be able to circumvent overfitting [31]. The authors explain the additional connection will force the model to learn the low-level feature of the expression, instead of the whole image. Furthermore, it is easy to see that the connection will pass more high-level identity information to the decoder, which can help to preserve the identity of the subject. Although the same paper also proposes adding two more discriminators for the latent variable and the generated image, with some experiments, the complex structures did not seem to improve the quality of the image significantly. Therefore, this version of the model only adapts the feedforward connection, rather than the entire model from the paper.

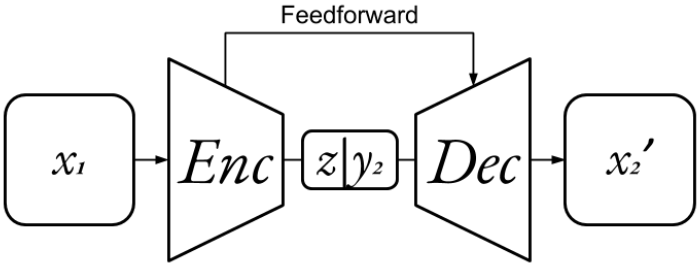
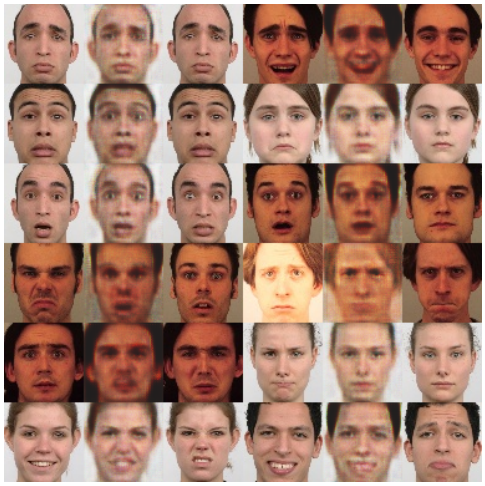


Figure 3.15: Structure of additional feedforward autoencoder. It is realised by copying the feature maps of a layer in encoder, and concatenating them with feature maps of the corresponding layer in decoder.

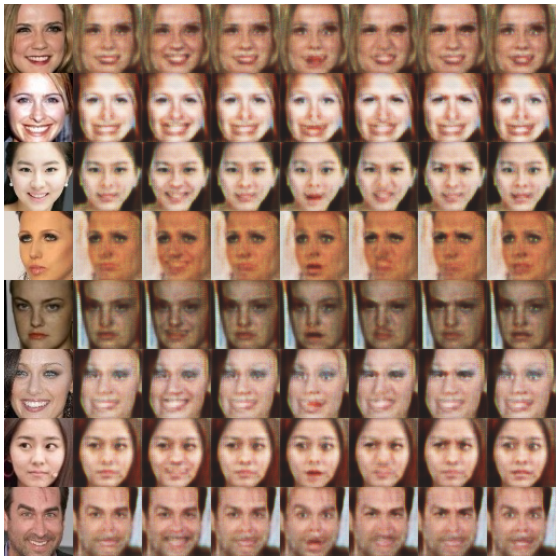
The objective function is similar to the last model, but this time the sum of square difference (also known as L2-norm) is used, while previously the absolute difference was used(L1-norm). Thus, the new objective is to minimise $\mathcal{L} = ||x_2 - x_2'||_2$.

Experiment

The training data for the experiment is still in the size 64×64. As shown in Figure 3.16, although the accuracy of the generated expressions have decreased, the identity of the subject is better preserved than before. Interestingly, if the input image has a neutral expression, the model will generate more accurate target expressions. On the other hand, for those with a big smile in the input, the model finds it hard to manipulate the mouth in the result. The major problem with this model is that the generated images are not sharp enough, which causes the hardness in recognising the identities of the testing samples.



(a) Training samples. The 1st and 4th columns display the input images, 2nd and 5th are generated images, 3rd and 6th are the corresponding targets.



(b) Testing samples. The 1st column displays the input images. The rest of the columns (from left to right) are the generated images under conditions: neutral, happy, sad, surprised, disgusted, angry and fearful.

Figure 3.16: Samples of size 64 × 64 generated by additional feedforward autoencoder.

3.4.4 StarGAN

Architecture

Now as we understand how the desired images can be generated from an input image using a simple autoencoder, we can bring back GANs to improve the sharpness of the image. The recently proposed StarGAN is a realisation of such architecture, but with a more complex training objective [5]. The model in this section is based on StarGAN, but with some modifications. The overall structure is shown in Figure 3.17.

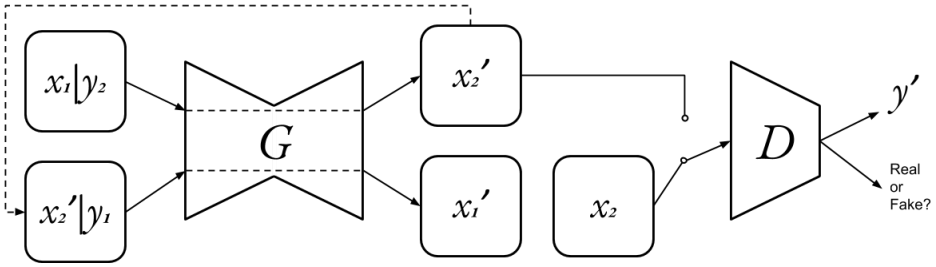


Figure 3.17: Structure of StarGAN. x_2' is generated from x_1 with target expression y_2. x_2' and the target image x_2 are fed into D for discrimination and classification. y' is the classification result. x_1' is a reconstruction of x_1, which is generated by putting x_2' and y_1(the label of x_1) into G.

The new generator is a variation of autoencoder, instead of using a deep encoder (compresses the data into a much smaller vector) and decoder with the additional feedforward connection, the updated encoder and decoder only has 3 layers of convolution/deconvolution. Moreover, the encoder does not compress the data into a vector, it keeps the result as a set of feature maps and applies 6 residual convolutional blocks [12] to them in the latent space before

feeding them into the decoder. The resulting generator takes the concatenation of a condition and an image as the input and generates the target image. For the discriminator, it uses a CNN as before, but besides determining whether the input is real or fake (denoted as D), it also has to classify the input into one of the seven expressions (denoted as D_{cls,y_i}).

The new objective involves three losses, namely Adversarial Loss, Classification Loss and Reconstruction Loss. **Adversarial Loss** is the usual objective of GANs (denoted as $\mathcal{L}_{adv}$), which is to help the generator to create images indistinguishable from the real ones. Here, we use the Wasserstein distance with gradient penalty [11] for calculating the loss. The discriminator is trained to maximise the loss, which is equivalent to minimise $-\mathcal{L}_{adv}$, while the generator is trained to minimise $\mathcal{L}_{adv}$.

Classification Loss is a measurement of how close the generated expression matches the given expression, and how accurate the discriminator classifies the expression in a real image. Therefore, we want to train the discriminator to minimise the loss

$$\mathcal{L}_{cls}^{r} = \mathbb{E}_{x_2,y_2}[-\log D_{cls,y_2}(x_2)],$$

where $D_{cls,y_2}(x_2)$ represents the probability that given a real image x_2, the model classifies it into y_2, which is the real label of x_2 (following the illustration of Figure 3.17). The pair x_2 and y_2 comes from the training data. It is not hard to see the higher accuracy will lead to the lower loss. As for the generator, we want it to minimise the loss

$$\mathcal{L}_{cls}^{f} = \mathbb{E}_{x_1,y_2}[-\log D_{cls,y_2}(G(x_1,y_2))],$$

where $G(x_1, y_2)$ is the generated image of the expression y_2 from input x_1. As before, $D_{cls,y_2}(\cdot)$ is the probability of an input image being classified as y_2. y_2 is the label for the target expression, which can be selected randomly. Therefore, to minimise the loss is the same as to generate images that are closer to the target expressions.

Reconstruction Loss is used because by only minimising the first two losses, indeed, the generated images will look realistic and have the correct expression, but there is nothing stopping the model to generate an image with a different identity. To guide the model away from this trap, we can try to make sure that if we take the generated image $x_2' = G(x_1, y_2)$ and put it back to the generator with condition y_1 (the expression of x_1), the generator is able to reconstruct x_1. This objective can be achieved by minimising the loss

$$\mathcal{L}_{rec} = \mathbb{E}_{x_1,y_2,y_1}\big[||x_1 - \underbrace{G(\overbrace{G(x_1,y_2)}^{x_2'},y_1)}_{x_1'}||_1\big],$$

where $G(x_1, y_2)$ is the generated image of expression y_2, $x_1' = G(G(x_1, y_2), y_1)$ is the reconstructed image of the original x_1. The difference of x_1 and x_1' is calculated using L1-norm (the absolute difference).

To add above losses together, we have the full objective:

$$\begin{cases} \mathcal{L}_D = -\mathcal{L}_{adv} + \lambda_{cls} \cdot \mathcal{L}_{cls}^{r} \\ \mathcal{L}_G = \mathcal{L}_{adv} + \lambda_{cls} \cdot \mathcal{L}_{cls}^{f} + \lambda_{rec} \cdot \mathcal{L}_{rec} \end{cases}$$

Here, λ_{cls} and λ_{rec} are the weights for the importance of the corresponding loss. During the training process, the generator is kept unchanged when minimising the loss $\mathcal{L}_D$ for the discriminator, and the discriminator is kept unchanged when minimising $\mathcal{L}_G$.

Experiment

As the resulting samples had the best quality among above experiments, better resolution dataset (128×128) were put into tests. The outcome is shown in Figure 3.18. Although there are some minor problems in some of the samples, it is much better than previous results in terms of sharpness, identity preservation and expression accuracy. One interesting sample in Figure 3.18 is the fifth row. Although there is no side face image in the training set, the model is still able to handle the expression of a side face.

Figure 3.18: StarGAN samples of size 128×128. The 1^{st} column displays the input images. The rest of the columns (from left to right) are the generated images under conditions: neutral, happy, sad, surprised, disgusted, angry and fearful.

Chapter 4

Evaluation

There are many ways to evaluate the final model from the last chapter, this report primarily focuses on assessing the quality of the generated expressions given refined inputs and try to find out how good the results match the target expressions as well as how well the identities are preserved. The measurements used for the evaluation are questionnaires, a trained classifier and an online identity recognition service.

4.1 Samples for Evaluations

Two images with similar quality as the training data were selected to generate samples for evaluations. Here, "Similar quality" is defined as that the image is sharp, well illuminated, the subject in the image is facing straight and does not have heavy facial hair. The reason for only using two inputs is to limit the number of questions in the questionnaire so that people will be more willing to complete it. The generated samples are showed in Figure 4.1.

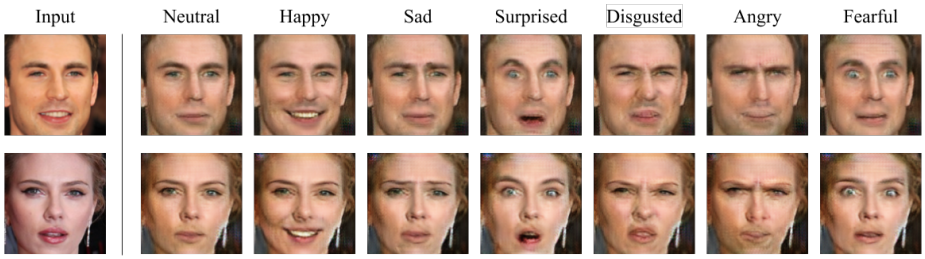


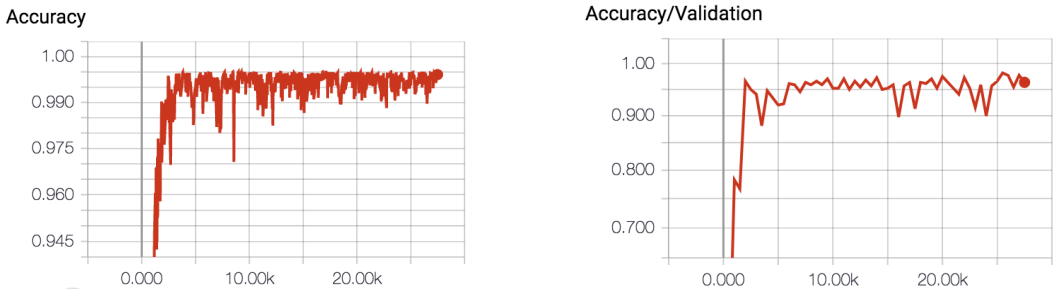
Figure 4.1: Samples used for evaluations.

It is reasonable to say that the identities in the samples are well maintained, and the expressions are looking natural.

4.2 Using Trained Classifier

A classifier is trained for measuring whether the generated samples match the target expressions. It was build using a ResNet[12] and trained on the three labelled datasets (FACES [8], RaFD [16] and KDEF [20]) with 80%/10%/10% splitting for training set/validation set/testing set. The resulting model has 98.41% accuracy for classifying expressions on the testing data. The training and validation accuracy are showed in the Figure 4.2. With this

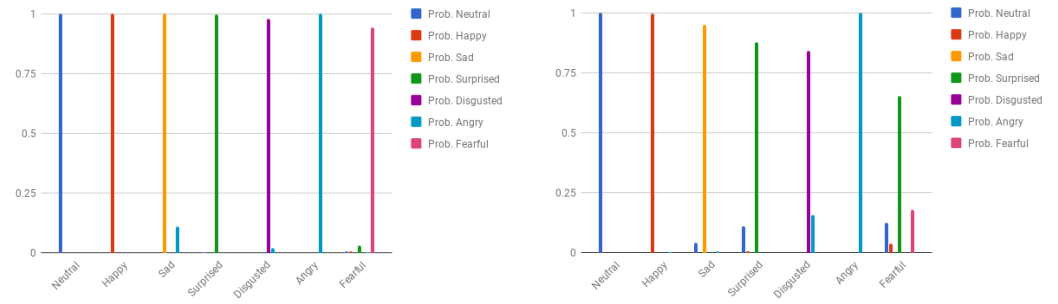
level of accuracy, we can assume if an image has one of the seven expressions, the classifier can identify it with nearly no mistake.



(a) Training accuracy **(b)** Validation accuracy

Figure 4.2: Accuracy of the trained classifier. x-axis represents the training steps.

After applying the classifier to the evaluation samples, only 1 out of 14 was classified into the expression that is different to its target. In other words, if we assume the classifier is able to tell the ground truth, then the generative model being evaluated has 92.85% accuracy for generating target expressions. Figure 4.3 shows the result and how confident the classifier was when putting the images into each expression. Samples from the first subject were all classified into the target expressions with high confidence. As for samples from the second subject, the only mistake was the Fearful sample, which was classified as Surprised. However, it is understandable, according to psychologists 'Fearful' is often confused with 'Surprised' and hard to recognise even for human [28]. Therefore, from the perspective of the classifier, the generative model is able to produce a convincing and accurate result for the selected expression.



(a) 1st set of samples **(b)** 2nd set of samples

Figure 4.3: Probabilities of each sample being classified to each expression.

4.3 Using Survey

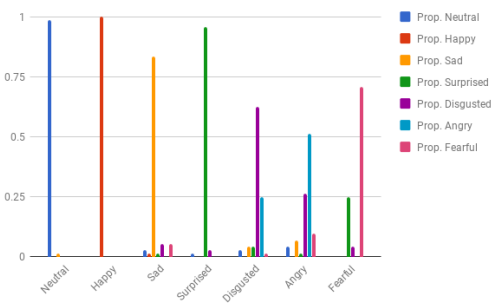
As the images are generated for the human, how the human perceives the samples becomes an important evaluation of the result. In this case, questionnaires were used, which were sent out on the social network. There are 71 responses in total. A copy of the questionnaire can be found in Appendix A. The questionnaire contains one set of real data (shown in Figure 4.4), and two sets of generated samples shown previously in Figure 4.1. The respondents need to match each image with an expression. The real data in the survey is used as a baseline for comparing the differences between how people perceive the real and the generated data.

Instead of treating each response individually, the results of the survey are analysed collectively. That is to say, given an image, the proportion of votes towards a certain expression

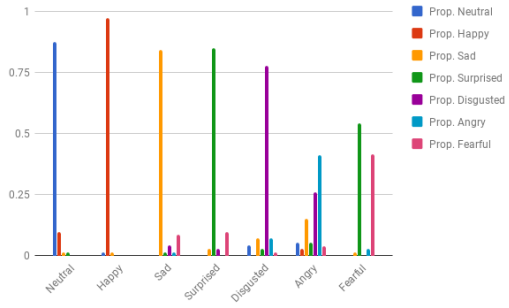
Figure 4.4: The real data in the questionnaire. Drew from RaFD [16].

is interpreted as the probability of the image having such expression, and the one with the highest probability (i.e. has the majority votes) is believed to be the real expression of the image. This method is used as the meaning of expressions are given by the majority of society in nature.

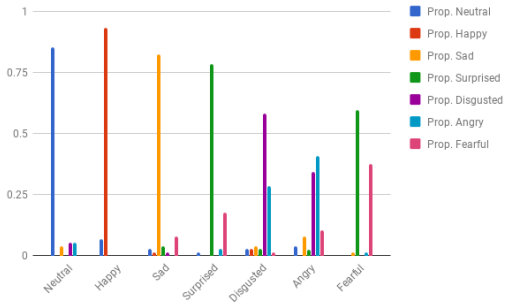
Results are shown in Figure 4.5. The responses of the baseline match its original labels, but there is a high uncertainty for Disgusted, Angry and Fearful. For the generated samples, the two images, which were supposed to be Fearful, are labelled as Surprised by the majority of the respondents. Apart from that, others are labelled the same way as their target expressions, although the uncertainty for Angry is higher than the baseline in both sample sets. Overall, based on the survey, the model is able to generate the target expression with $12/14 = 85.71\%$ accuracy. In particular, the expressions of Neutral, Happy, Sad and Surprised are generated with high accuracy.



(a) Real data (Baseline)



(b) 1st set of samples



(c) 2nd set of samples

Figure 4.5: Proportion of responses in each expression.

4.4 Using Identity Recognition Service

Another criteria for a good quality model is whether the generated image has the same identity as the input. The evaluation for this part is done using an online identity recognition

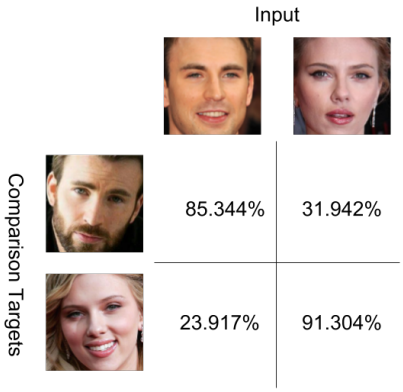


Figure 4.6: A test of the identity recognition service, given images of the same and different identities. The percentages indicate how confident the algorithm believes the two images came from the same person.

service provided by Face++[1]. The company claims their algorithm has a very high accuracy in the task of recognising the same person in two different images. Figure 4.6 is a result of a test for the identity recognition service. Indeed, the algorithm returned a high probability for two faces that come from the same person and returned a very low probability otherwise.

Since the service is reliable, the generated samples were put to the test. Results are shown in Figure 4.7. All generated samples are believed to have the same identity as the input by the identity recognition service with very high confidence. Thus, we can say that the identity of the input is well preserved by the model.

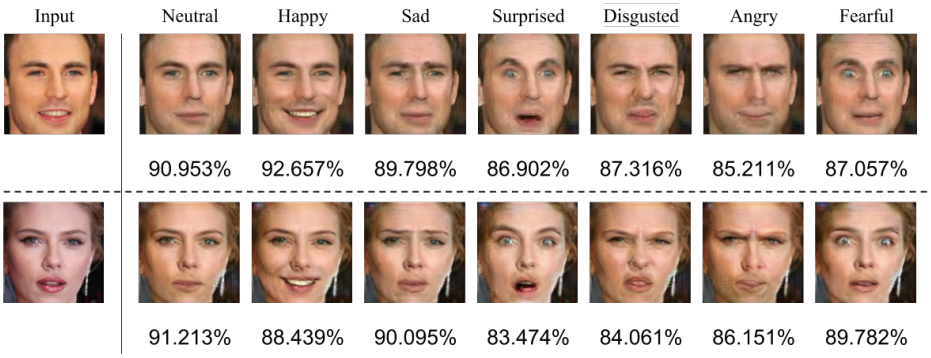


Figure 4.7: Identity evaluation using the service provided by Face++. The percentages indicate how confident the algorithm believes the two images came from the same person.

4.5 Conclusion

From the evaluations using the classifier and the survey, it is clear that given a refine input, the model is able to generate high quality and accurate images in one of the seven expressions. However, Fearful is the hardest expression to generate, which often results in the expression of Surprised. In addition, the further evaluation also shows that the model is able to maintain the identity of input in the generated image. Altogether, the model provides a satisfying solution to the project.

[1]Face++ : `https://www.faceplusplus.com/face-comparing/`

Chapter 5

Reflection and Achievements

This chapter includes the reflection of the project, which specifies the reasons that the project was organised in the way it was, as well as the difficulties encountered during the project. Finally, the chapter will be concluded by a summary of achievements made across the year.

5.1 Planning and management

Looking back, the project started with an organised pace. From the literature review, learning the TensorFlow framework and the basic components to creating the first GANs and DCGAN, everything was following the plan[1]. However, after trying to adapt DCGAN for face images, the project was not on the plan anymore.

One reason is that the work after is all experimental and problems occurred one after another. The time taken for solving those problems were indeterminate, as it depended on whether I could find the solutions. One may argue that I could have implemented the StarGAN at the beginning, but actually, it was not possible as the StarGAN paper was only published during the later stage of the project. Therefore, the experiments were necessary.

Another obstacle that made the project hard to plan in the later stage is the freshness of the methods used in the project. The whole idea of GANs was created 4 years ago, the variations and methods of improvements were proposed within a year or two. In other words, there are only a few or even no realisations of the methods online. Thus, very often I had to reproduce the methods only based on the specifications in papers, which requires understandings of the underlining knowledge of the method. Moreover, the realisation process was time-consuming, as samples from the reproduced model might not match those in the paper, in which case I had to consider whether it was the problem of my implementation or the problem of the paper.

Despite the unforeseen problems, the project still reached the second milestone and was on the way to the third. With more time, I would try to improve the image quality and find a way to preserve the subject identity better. Overall, I believe the project progressed at a managed pace. If I could start over, I would try to push the project harder at the beginning, and hopefully create a significant amount of original research that could be published in a conference.

[1] A copy of the initial project plan is attached in Appendix B.

5.2 Achievements

The most obvious achievement is the creation of the final model, which is able to generate the selected expression of the input image. And as the model is deployed online[2], it can be accessed and played by everyone. More intangible achievements are the new machine learning knowledge I gained, and the skills of understanding and reproducing ideas from research papers. The latter is the main reason I proposed this project in the first place, which in a way is more important than the project itself, and I am glad that I have accomplished it. In the future, I want to continuously utilise these skills, and hopefully contribute to the machine learning research community.

[2] `http://timx.me/Facial_Expression_Modifier/`

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[2] Alejandro Baldominos. A comparison between nvidia's geforce gtx 1080 and tesla p100 for deep learning, Oct 2017.

[3] David Berthelot, Tom Schumm, and Luke Metz. BEGAN: boundary equilibrium generative adversarial networks. *CoRR*, abs/1703.10717, 2017.

[4] Volker Blanz, Curzio Basso, Tomaso Poggio, and Thomas Vetter. Reanimating faces in images and video. *Computer Graphics Forum*, 22(3):641–650, Nov 2013.

[5] Yunjey Choi, Min-Je Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. *CoRR*, abs/1711.09020, 2017.

[6] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1486–1494, Cambridge, MA, USA, 2015. MIT Press.

[7] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

[8] Natalie C. Ebner, Michaela Riediger, and Ulman Lindenberger. Faces—a database of facial expressions in young, middle-aged, and older women and men: Development and validation. *Behavior Research Methods*, 42(1):351–362, Feb 2010.

[9] Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

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

[11] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 5769–5779. Curran Associates, Inc., December 2017. arxiv: 1704.00028.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[14] T. Kanade, J. F. Cohn, and Yingli Tian. Comprehensive database for facial expression analysis. In *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, pages 46–53, 2000.

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

[16] Oliver Langner, Ron Dotsch, Gijsbert Bijlstra, Daniel H. J. Wigboldus, Skyler T. Hawk, and Ad van Knippenberg. Presentation and validation of the radboud faces database. *Cognition and Emotion*, 24(8):1377–1388, 2010.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[18] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, Dec 2015.

[19] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews. The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, pages 94–101, June 2010.

[20] D. Lundqvist, A. Flykt, and A. Öhman. The karolinska directed emotional faces (kdef). Stockholm: Department of Neurosciences Karolinska Hospital, 1998.

[21] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[22] Abraham H. Maslow. *The Psychology of Science*. New York: Harper & Row, 1966.

[23] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[24] Dev Nag. Generative adversarial networks (gans) in 50 lines of code (pytorch), Feb 2017.

[25] Michael A. Nielsen. Neural networks and deep learning, 2015.

[26] Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, and Jose M. Álvarez. Invertible conditional gans for image editing. *CoRR*, abs/1611.06355, 2016.

[27] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[28] Annie Roy-Charland, Melanie Perron, Olivia Beaudry, and Kaylee Eady. Confusion of fear and surprise: A test of the perceptual-attentional limitation hypothesis with eye movement monitoring. *Cognition and Emotion*, 28(7):1214–1222, 2014. PMID: 24460373.

[29] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[30] Qingshan Zhang, Zicheng Liu, Gaining Quo, Demetri Terzopoulos, and Heung-Yeung Shum. Geometry-driven photorealistic facial expression synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):48–60, Jan 2006.

[31] Y. Zhou and B. E. Shi. Photorealistic facial expression synthesis by the conditional difference adversarial autoencoder. In *2017 Seventh International Conference on Affective Computing and Intelligent Interaction (ACII)*, volume 00, pages 370–376, Oct. 2017.

Appendix A

Questionnaire

Facial Expressions Evaluation

This questionnaire contains 3 multiple choice questions. Each of them cover 7 face images with expressions: neutral, happy, sad, surprised, disgusted, angry and fearful. Please select what you think is the best match. Many thanks!!

1. **Subject 1**

(1) (2) (3) (4) (5) (6) (7)

Mark only one oval per row.

	Neutral	Happy	Sad	Surprised	Disgusted	Angry	Fearful
Image (1)							
Image (2)							
Image (3)							
Image (4)							
Image (5)							
Image (6)							
Image (7)							

2. **Subject 2**

(1) (2) (3) (4) (5) (6) (7)

Mark only one oval per row.

	Neutral	Happy	Sad	Surprised	Disgusted	Angry	Fearful
Image (1)							
Image (2)							
Image (3)							
Image (4)							
Image (5)							
Image (6)							
Image (7)							

3. **Subject 3**

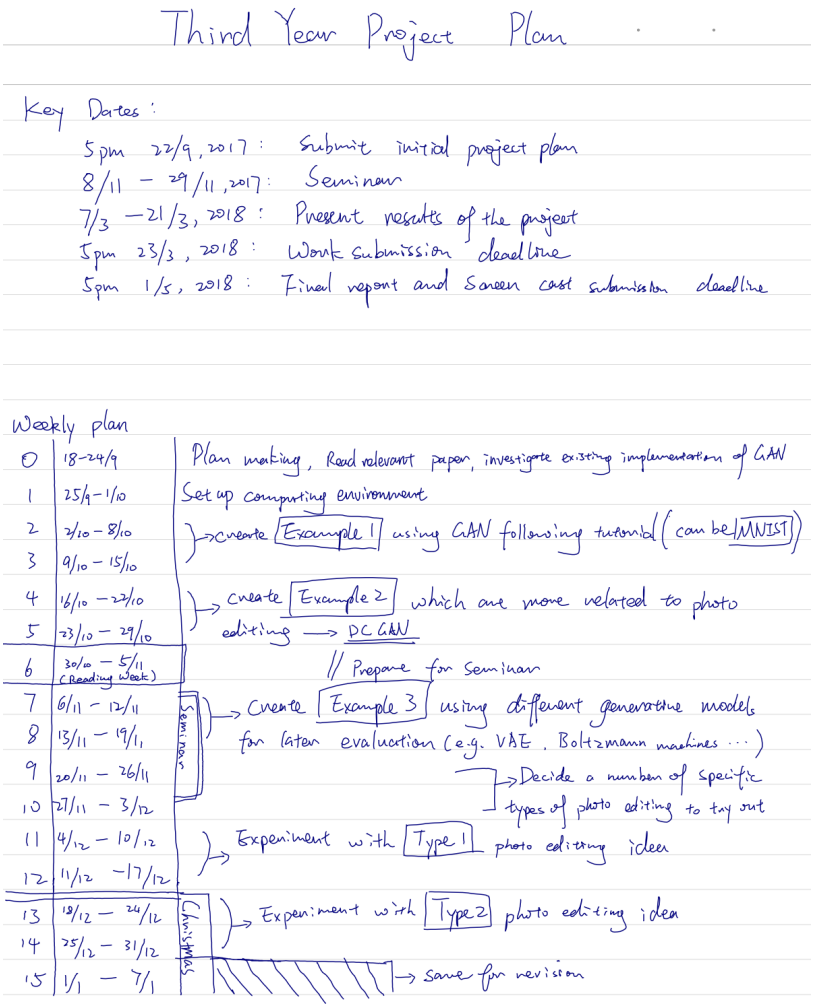
(1) (2) (3) (4) (5) (6) (7)

Mark only one oval per row.

	Neutral	Happy	Sad	Surprised	Disgusted	Angry	Fearful
Image (1)							
Image (2)							
Image (3)							
Image (4)							
Image (5)							
Image (6)							
Image (7)							

Appendix B

Initial Project Plan



Third Year Project Plan

Key Dates:

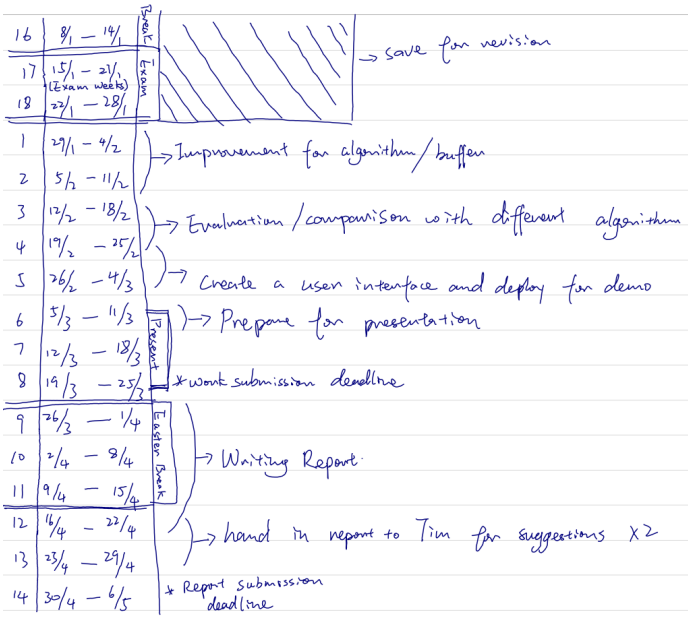
5pm 22/9, 2017: Submit initial project plan
8/11 – 29/11, 2017: Seminar
7/3 – 21/3, 2018: Present results of the project
5pm 23/3, 2018: Work submission deadline
5pm 1/5, 2018: Final report and Screen cast submission deadline

Weekly plan

0	18-24/9	Plan making, Read relevant paper, investigate existing implementation of GAN
1	25/9-1/10	Set up computing environment
2	2/10 – 8/10	→ create Example 1 using GAN following tutorial (can be MNIST)
3	9/10 – 15/10	
4	16/10 – 22/10	→ create Example 2 which are more related to photo editing → DC GAN
5	23/10 – 29/10	
6	30/10 – 5/11 (Reading Week)	// Prepare for Seminar
7	6/11 – 12/11	→ Create Example 3 using different generative models for later evaluation (e.g. VAE, Boltzmann machines ···)
8	13/11 – 19/11	
9	20/11 – 26/11	→ Decide a number of specific types of photo editing to try out
10	27/11 – 3/12	
11	4/12 – 10/12	→ Experiment with Type 1 photo editing ideas
12	11/12 – 17/12	
13	18/12 – 24/12	→ Experiment with Type 2 photo editing ideas
14	25/12 – 31/12	
15	1/1 – 7/1	→ Save for revision

Seminars

Christmas


16 8/1 – 14/1
Revise
17 15/1 – 21/1 (Exam weeks)
18 22/1 – 28/1
Exam
→ save for revision
1 29/1 – 4/2
2 5/2 – 11/2
→ Improvement for algorithm / buffer
3 12/2 – 18/2
4 19/2 – 25/2
→ Evaluation / comparison with different algorithm
5 26/2 – 4/3
→ Create a user interface and deploy for demo
6 5/3 – 11/3
7 12/3 – 18/3
8 19/3 – 25/3
Present
→ Prepare for presentation
* work submission deadline
9 26/3 – 1/4
10 2/4 – 8/4
11 9/4 – 15/4
Easter Break
→ Writing Report
12 16/4 – 22/4
13 23/4 – 29/4
→ hand in report to Tim for suggestions X2
14 30/4 – 6/5
* Report submission deadline