INM705 – Deep Learning for Image Analysis

James Foster: 150012985
Vittoria Castelnuovo: 190042232

Image Generation using a Deep Convolutional Generative Adversarial Network

Table of Contents

1. Introduction

1.1 Overview

The field of computer vision has garnered much attention and become an active area of research through the introduction of deep learning algorithms. Introduced by Goodfellow et al. in 2014, Generative Adversarial Networks (GANs) have become the state-of-the-art for many computer vision tasks. Some of these include resolution enhancement, data generation and augmentation, face generation and image translation.

GANs are comprised of two neural networks: a Discriminator and a Generator. These compete against each other to achieve their respective objectives. The Discriminator network is responsible for outputting the probability that an input belongs to a given data distribution. Instead, the Generator network aims to generate images that resemble the distribution, in an attempt to "fool" the adversary network into misclassifying its sample.

1.2 Scope of the Project

This project will outline the development of a Deep Convolutional Generative Adversarial Network (DCGAN) for image analysis and generation, using the MNIST dataset. More specifically, if successful, the images generated by the network will resemble the ones found in the dataset.

We will measure the performance our model by analysing the loss calculations for each of the individual networks as well as the feasibility via probability of an output being real or fake.

Figure 1: example of images from a DCGAN trained on MNIST

1.3 Dataset Analysis

The dataset chosen for this project is the MNIST set. This distribution contains 60,000 grayscale images of handwritten digits (seen Figure 1). This dataset was selected for several reasons. Firstly, both Goodfellow et al.'s GAN and Radford et al.'s DCGAN implementations employ this dataset. Moreover, many other state-of-the-art machine learning algorithms have been trained and tested on this dataset. This is significant as it shows that the dataset is widely used in the field and can be used as a benchmark comparison when analysing the yielded results of different models.

When training an artificial intelligence system, it is crucial that the dataset used to train the model is balanced. If not, the model may learn to discriminate a specific class in the distribution

and may yield meaningless or inaccurate results. The MNIST dataset is a very balanced dataset, as each class has roughly 6,000 instances.

Class	0	1	2	3	4	5	6	7	8	9
Number of instances	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949

Table 1: Table indicates the number of instances for each class in the MNIST dataset

Another advantage of this dataset is that it is readily available when importing *torchvision's datasets* module. This is particularly useful as machine learning projects are always data-heavy by nature, therefore being able to easily import and load data is extremely time efficient.

Pre-processing the data before passing it into the network, in fact, was not very time consuming. We split the data into two sets: a training set of 40,000 images, and a testing set of 20,000. The images we normalized to 0.2, as the DCGAN implementation dictates, and subsequently transformed to tensors.

Table 2: examples of the images from the MNIST datasets training images

1.4 Literature Review

Through the introduction of deep learning approaches, the field of artificial intelligence has reclaimed momentum. One of the most influential algorithms responsible for this phenomenon are GANs. Since their introduction in 2014, many have altered this model to yield solutions to many different image analysis problems.

One of the most successful adaptations are DCGANs, implemented by Radford et al. in 2016. These networks keep the objective and intuition of original adversarial networks. However, use typical convolutional neural network (CNN) architecture within both networks.
Additionally, other variations on the original GAN architecture have been proposed, such as CycleGANs, Deformable GANs, Conditional GANs, Super-Resolution GANs, StackGANs, and many others. These networks employ the GAN structure, yet the scope of these networks is varied. For instance, CycleGANs are used to implement image-to-image translation, whereas Deformable GANs are used for human-pose generation. Clearly, the scope of these systems is very different. However, it is remarkable that the same architecture can yield such distinct and successful results.

This spurt of image generation software has been steadily and increasingly growing within the last decade. In fact, it has inspired many to create new and original solutions using computer vision frameworks. An example of this being AI Artists, which use generative networks to creative fascinating novel artwork. These models are trained on a distribution of thousands of artworks, and the images yielded are astounding.

Image generation has also been employed to produce images of human faces. These systems are able to select specific traits and features from various faces and seamlessly fuse them

together. Similarly, data generation systems have gained attention, as GAN models are able to produce extremely realistic and precise images. Generative networks are now used to augmented data for many different purposes, such as medical imaging.

The field of computer vision is expected to continue growing, as AI systems make their way into people's everyday life. In fact, through the influx of data produced worldwide daily, AI systems will continue to test the boundaries of creativity and reality.

1.5 Related Work

There has been a lot of activity and progress in the field on computer vision in recent years. An interesting and insightful experiment was carried out by a team of researchers from the University of Groningen, in the Netherlands. A pattern recognition AI system was used to perform palaeography of the Isaiah scroll. The Isaiah scroll is one of the seven dead seas scrolls discovered in the 1940s that contains the oldest known versions of the bible, dating back to around the 3rd century BCE.

Figure 2: the Great Isaiah Scroll, the most well preserved of the biblical scrolls

They chose to analyse the Hebrew letter aleph as it appears more than 5000 times within the scroll. They used 3 types of feature extractors to analyses the letters. The first was a feature extractor at the textual level. The second was a feature extraction at the allograph level to look at variants of the letter with neural networks. Then finally a combined feature which combined both the previous 2 feature extractors.

From this they were able to extract ancient ink traces seen within the digital images of the letters taken from the scroll. These gave them information on the ink traces which can be attributed to a specific person's muscle movement. Up until now, the scroll was believed to have been written by one scribe. The scroll maintains strong consistency throughout which is why this was believed. However, the results suggested those 2 different scribes had work together to write the scroll but worked closely to keep the style of writing throughout. Due to this consistency the researchers concluded furthermore that they had likely undergone the same training via either the same school or family.

This case study is relevant to our project's scope as it demonstrates the advances that AI systems have made in the field of computer vision. Alternatively, GANs could have been used in to analyse the scroll and may have been able to detect the two handwritings, or even reproduce computer-generated data similar to that of the scrolls.

2. Methodology

2.1 Overview

The following section will outline the high-level theory and implementation of this project. Firstly, the intuition behind GANs will be described. Subsequently, a critical comparison between these and DCGANs will be exposed, describing the key differences between these two models. The value function used to train these networks will be described. Finally, the architecture of both Discriminator and Generator networks will be discussed.

2.2 Generative Adversarial Networks

GANs, first introduced by Goodfellow et al. in 2014, are semi-supervised deep learning models. They have proven to be highly successful machine learning models for a variety of deep learning problems in the fields of natural language processing, data augmentation and computer vision, alike.

GANs are comprised of two distinct networks, commonly referred to as a Discriminator *D* and a Generator *G*, which work in opposition to one another. The first of these networks is trained on a distribution of image and is responsible for detecting whether a new unseen input is “real” or “fake”. In other words, the network must retain an accurate and generalizable knowledge of the training data to classify and distinguish between unobserved data, which belongs to the training set, or data that is computer-generated. On the other hand, the Generator network attempts to “fool” the Discriminator network into categorizing its generated images as “real” by resembling the data’s distribution. It can be said that the more successful the Generator becomes at this the better the model has become.

2.3 Deep Convolutional Generative Adversarial Network

The introduction of GANs in the field of machine learning was highly influential, inspiring many to create their own variations. Some of the most successful being CycleGANs, Conditional GANs, Deformable GANs, and DCGANs.

DCGANs are a sub-type of the GANs, introduced by Radford et al. in 2016 and the one which we are particularly interested in for this project. The key distinction being that the Discriminator and Generator networks of the DCGAN model only employ layers of convolution, not fully connected layers. This enables both networks to retain important features of the distribution in a hierarchical manner across each network’s layers. In addition, as Radford et al. discuss in their paper, the CNN-inspired architecture ensures stability when training.

Another distinction is that DCGANs do not use any pooling layers, which enables the Discriminator and Generator networks to learn their own spatial downsampling and upsampling, respectively. In addition, removing pooling layers allows for the model to reach optimal convergence at a faster speed.

In the original GAN implementation, batch normalization is employed to avoid the Generator network from collapsing all the distribution into one image and retain a balanced oscillation of the distribution’s datapoints. However, Radford et al. argues that by removing the last batch normalization layer from the Discriminator network input layer and the Generator’s output layer, the model’s stability is improved.

Lastly, the activation functions employed in the DCGAN implementation are the LeakyReLU and ReLU functions for the Discriminator and Generator networks’ hidden layers, respectively. This contrasts with the original GAN implementation which makes use of the Maxout activation. Radford et al. elucidate that the use of these two functions ameliorates the network’s ability to learn the data’s colour space in less time.

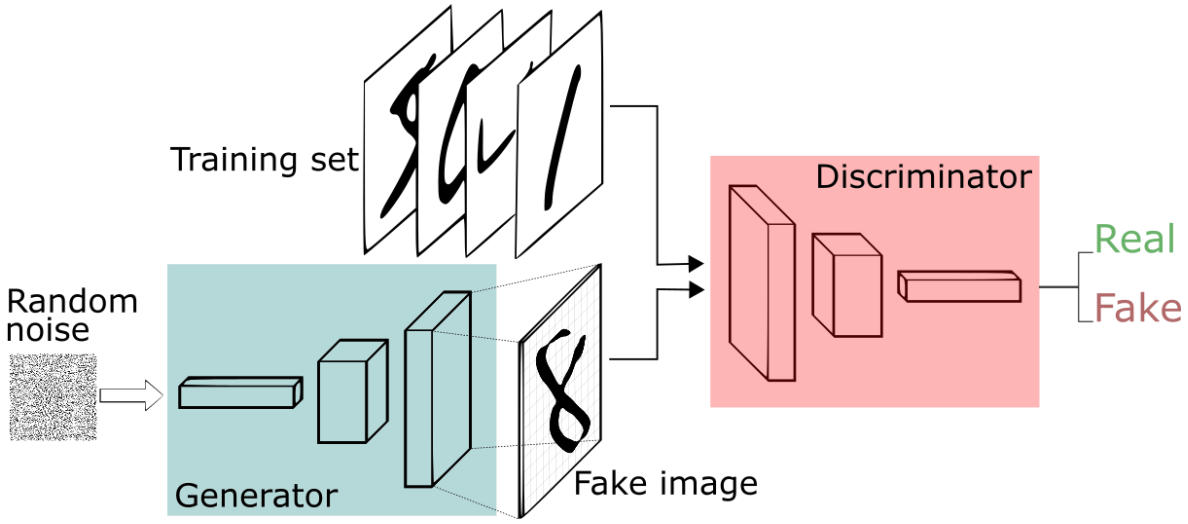

Figure 3: Generative Adversarial Network framework

2.4 Value Function

The value function which is used to train the adversarial networks as presented by Goodfellow can be seen in the figure below.

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

Figure 4: DCGAN value function equation

This function consists of a minmax game where the Generator and Discriminator models compete with one another to maximize their respective loss. The Generator is fed with a random noise vector *z,* which morphs into a fake sample of the distribution, *G(z)*. This output reflects the real dimensions (64x64) of the sample distribution, *x*. Both *x* and *z* are fed to the Discriminator network and are classified as either real or fake, denoted by a probability between 0 and 1.

The value function can be divided into two terms. The first observes the expectation of the log of *D(x)*, which signifies the probability of the input to the discriminator network being a real sample of the distribution. The Discriminator network aims to maximize this value. The second part of the equation above describes the Discriminator's expected log likelihood of the datapoint being "fake", produced by the Generator network, *D(G(z))*. The Discriminator strives to minimize this second value. However, as this second term is presented as *(1-(D(G(z)))*, the network is forced to maximize both. On the contrary, the Generator network aims to minimize both terms of the value function, as its true goal is to minimize the second term of the equation.

2.5 Discriminator Network

The Discriminator network of the DCGAN is responsible for classifying datapoints belonging to a given distribution, in our case the MNIST dataset. The model implemented can be viewed as a CNN, where the input image is fed through various layers of convolution operations. However, this model differs to traditional CNNs, as it lacks any pooling or fully connected layers.

The activation functions used in this network are LeakyReLU and Sigmoid. The LeakyReLU function is a bounded one, which is considered more accurate and time efficient in comparison to other non-linearity functions. Moreover, it actively counters the vanishing gradient problem, a

common issue in machine learning where the neurons of a network become increasingly saturated as an input passes through the layers.

The Sigmoid activation function is used in the Discriminator network's last layer. This is because the model, which in essence works as a classifier, must output a probability of the image being real or fake. In other words, must output a probability of the image belonging to the data distribution. Therefore, by employing the sigmoid function which returns a value between 0 and 1, the model is able to achieve its objective.

The input that is fed to the Discriminator network is a 64x64 image, which passes through the first convolutional layer and is translated into a 128x128 image. Morphing into a 256x256, 512x512, 1024x1024 image, at each subsequent convolutional operation.

2.6 Generator Network

The Generator Network can be described as the complement of the Discriminator network. In fact, their architectures are opposite, yet symmetrical. The input to the Generator is a random noise vector which gets shaped into a 1024x1024 image. As the network deepens, this vector is resized to 512x512, 256x256, 128x128 and finally, 64x64. This pattern mimics the layers of convolution of the Discriminator network, and similarly no pooling or fully connected layers are employed.

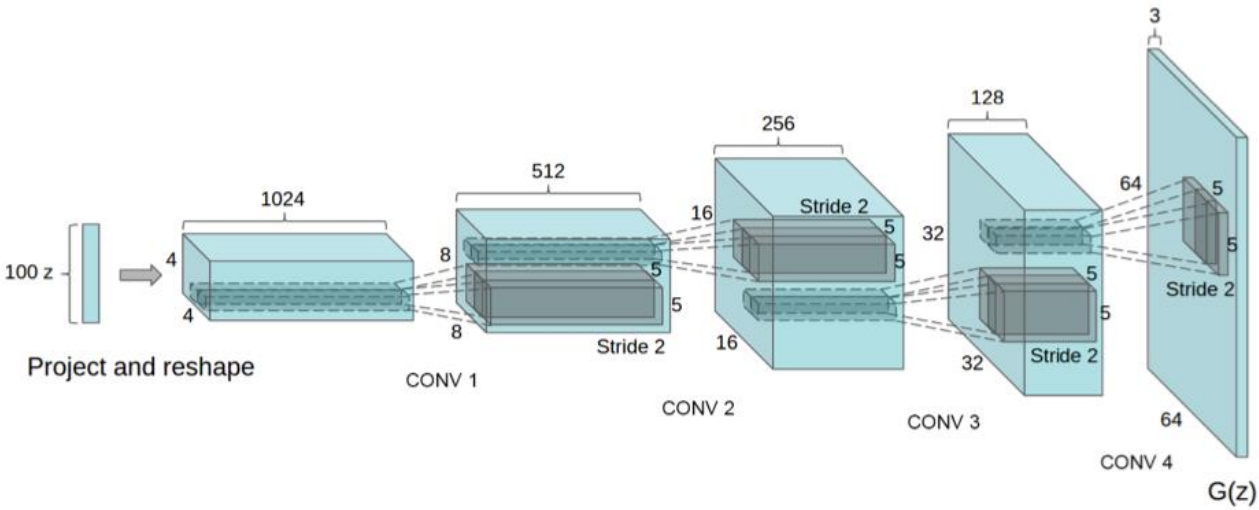

Figure 5: Generator Network architecture, note absence of pooling or FC layers

The Generator network aims to "fool" the Discriminator network into misclassifying its generated image as part of the distribution. In order to achieve this objective, this network mimics the samples of the data and receives feedback on the degree of success of its generated work.

The activation functions used for this network are ReLU and Tanh. The ReLU function, found in all layers of the model except for the last, is a commonly used one in the scope of machine learning. This function activates on positive inputs only, thereby saturating more quickly neurons which negative gradients. Though this may increase the vanishing gradient problem, it also allows for the network to learn more quickly.

In the final layer of the Generator the Tanh function is employed. This activation is similar to the previously described Sigmoid function, however outputs values in the range of -1 and 1.

2.7 Our Implementation

Our implementation of the DCGAN follows closely that built by Radford et al. in 2016. The following sections will outline the details of our project's implementation. The hyperparameters will be described first. Subsequently, the training of the networks will be discussed.
We will be using the tqdm library to help track our training progress and estimate how long our model will take to train. It allows to see a progress bar for each epoch as well as the time taken for said epoch.

2.8 Hyperparameters

The results GANs yield have a strong correlation to the hyperparameters set for the model. In fact, these parameters are so sensitive, that a small difference in any of the values can garner a significant difference in results, and in some cases may lead to the networks' collapsing. Radford et al.'s paper clearly outlines the hyperparameters used for the DCGAN's implementation, which we replicated for our own implementation.

Hyperparameters	Values
Mini batch size	128
Leaky ReLU Activation	0.2
Learning Rate for Adam Optimizer	0.0002
Betas for Momentum	0.9, with reduction of 0.5
Batch Normalization	0.02

Table 3: the hyperparameters we used for training

The paper elucidates that the model obtained better with a learning rate of 0.0002, instead of a commonly used value of 0.001. Moreover, the beta values above helped stabilize the training phase. Lastly, the Adam optimizer is chosen as it yields the most successful results, according to Radford et al. In fact, they state that the loss of the Discriminator is highest employing this optimizer: meaning the network tends to misclassify the Generator's images are samples of the real distribution.

2.9 Training Phase

The training of the DCGAN sees the implementation of the minmax value function. The training loop is divided into two parts: training the Discriminator network and the Generator.

To train the Discriminator we start by creating a batch of real samples of the distribution, feeding them to the network. We calculate the loss *log (D(x))*, and backpropagate this value to calculate the gradients of the model. Subsequently, we create another back for the fake samples using the Generator network. Once again, the loss of *(1-(D(G(z)))* is computed and backpropagated. Lastly, we calculate the accumulating sum of these two losses, and step using the Adam optimizer.

To train the Generator network, instead, we feed *G(z*) to the Discriminator network. The losses of the Generator model are calculated using the real ground truths of the images. This may appear senseless, however Goodfellow et al. cite this as a crucial factor in the Generator network's learning of the distribution *x*. Next, the gradients are calculated by backpropagating this loss, and lastly the optimizer updates the Generator network's parameters.

3. Results

3.1 Overview

This section will outline the results obtained in this project's implementation. Our primary DCGAN model was trained for 600 epochs taking a total of around 10 hours. The results produced can be seen in the figures and tables below.

3.2 Primary Experiment

As our models hyperparameters were chosen from the original paper there was very little or no need for tuning them as this had already been done. The figures below show average loss values at each epoch, and *D(x)* values so we can easily see how the model is learning.

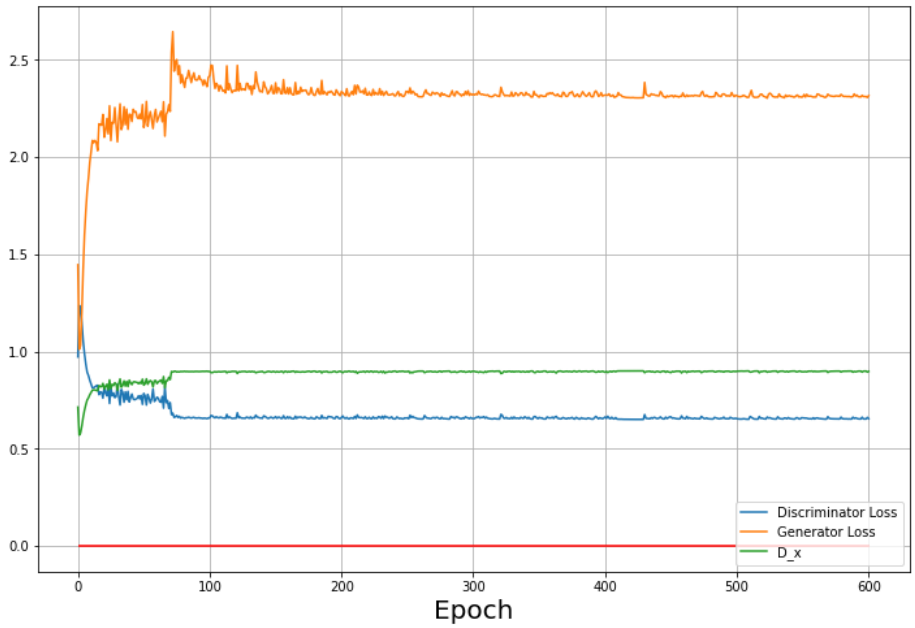

Figure 6: Generator and Discriminator loss over 600 epochs as well as D(x)

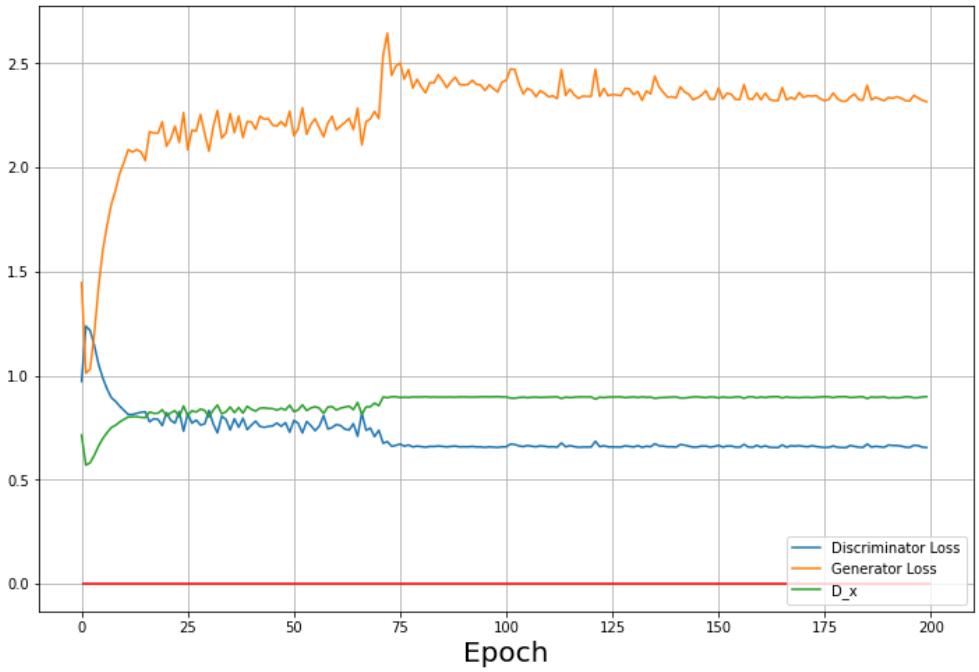

Figure 7: same results as figure 6 but only first 200 epochs are displayed

Figures 6 and 7, which show the losses of the two networks, give insight to the manner in which the models are learning. As GAN literature discusses, most successful results for the Discriminator's loss should stabilise around 0.5; instead, the Generator's loss should be between 0.5 and 2.0. As can be observed in the graphs, the Generator network becomes very good at "fooling" its counterpart network into classifying its generated images as part of the original distribution. This in turn, affects the Discriminator's loss value, which after less than 100 epochs stabilises at around 0.6. The *D(x)* value shows that the Discriminator network is still able to accurately identify real samples of the distribution, demonstrating that its classification is quite well-tuned.

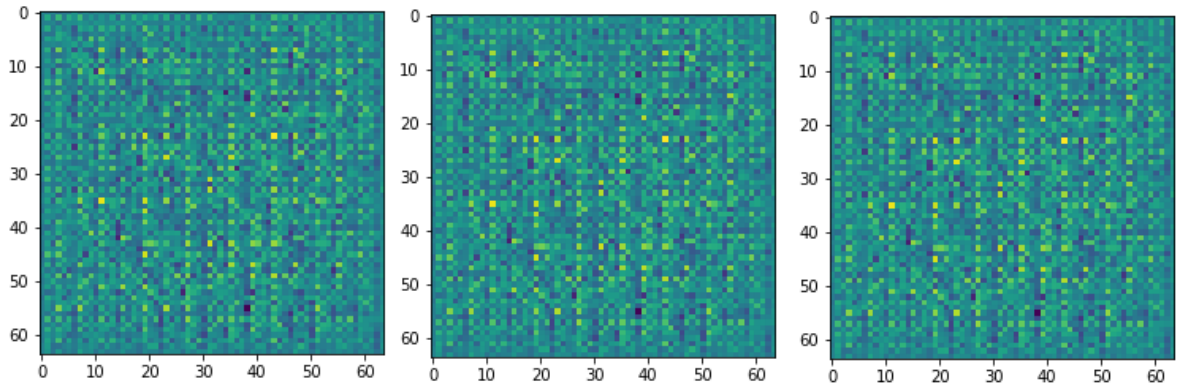

Table 4: Images produced by the Generator network after no training

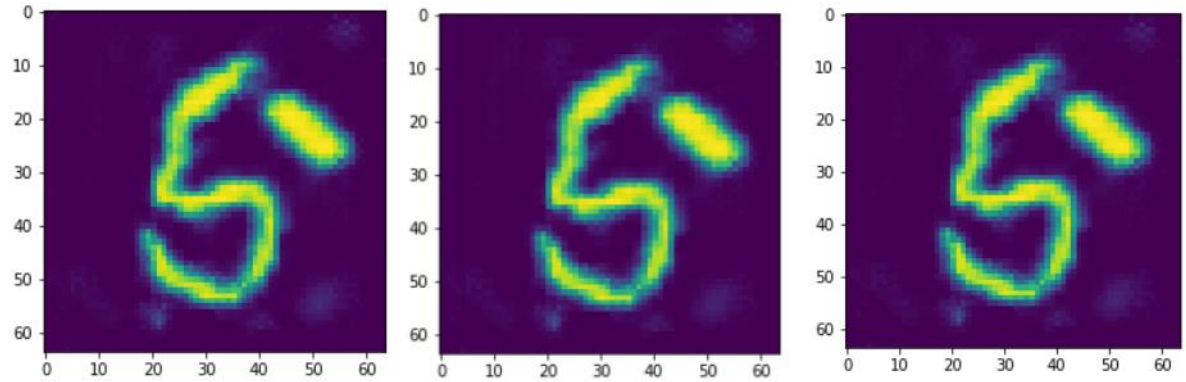

Table 5: Images produced by the Generator network after 100 epochs of training

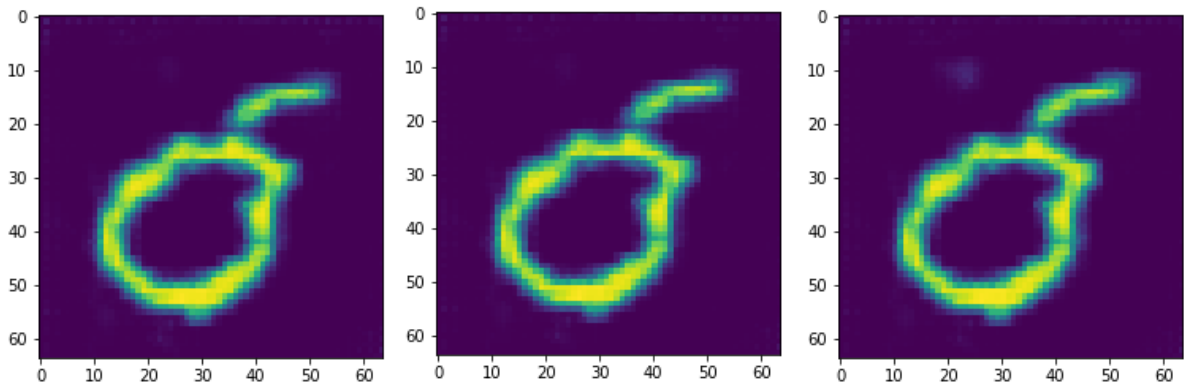

Table 6: Images produced by the Generator network after 600 epochs of training

Table 4 shows the images generated at the start of training, at the first epoch. It is evident that the images initially produced by the Generator network are mostly random noise, as the network has not learned any features to reproduce from the distribution. This is expected as the network's input before training is a random noise vector. The next one, table 5, shows the output after 100 epochs since by this point the model had already shown signs of reaching its equilibrium. The final table below it, table 6, shows images the Generator network produced after 600 epochs. We can see that the network is able to output images that somewhat resemble handwritten digits found in the MNIST dataset distribution. The images however seem to slowly morph between different numbers once equilibrium has been reached. This can be seen by looking at table 7 below, which shows one image form at 100 epoch intervals (excluding epoch 200). Notably, all are remarkable computer-generated images. However, the ones that fall short appear to be morphing from one to another. Only one image is shown, since when we checked at each epoch all images at that time step were nearly identical. This is shown clearly in the epoch specific tables 5 and 6 above.

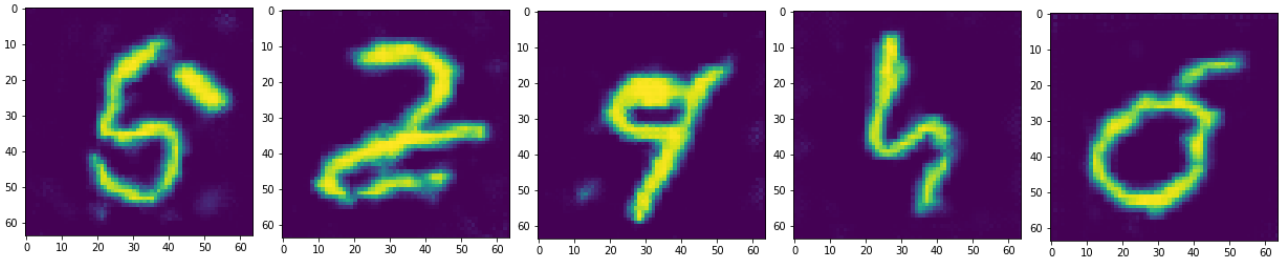

Table 7: Images produced by the Generator network at 100, 300, 400, 500 and 600 epochs

Research on GANs have elucidated that the best generated images are produced when the model stabilises. However, long periods of training stability may also generate high-variance loss, thereby producing low results. This is reflected in our results as we can see that images generated at the time step when the model stabilises seem to better represent the MNIST dataset. In fact, our model stabilizes at around 75 epochs, and images outputted at the 100th epoch are quite successful.

The losses of the Generator and Discriminator reach equilibrium between 75 and 200 epochs of training. In fact, we can observe that there is no oscillation after they have reached convergence. This is positive and provides us with strong reassurances of our model's success. This is because it proves that the DCGAN has avoided any form of mode collapse. Moreover, as Radford et al. mention in their findings, the DCGAN's architecture is less susceptible to mode collapse in comparison to traditionally implemented GANs.

3.3 Secondary Experiment

We ran another experiment for a shorter amount of time using slight variations in hyperparameters, to make sure that the results obtained were not casual, and that they maintained consistency. The changes made were to the learning rate, batch size and momentum. These were changed to 1e-6, 100 and 0.4, respectively. This model was trained for 200 epochs, the results of which can be seen below in figure 8. It follows a very similar pattern to the primary experiment as expected and appears to show signs of even further improvement right at the end. This would make sense as the output images are still a little bulky in form. Moreover, using this parameter for the learning rate, as Radford et al. mentioned, the model will need more time to converge.

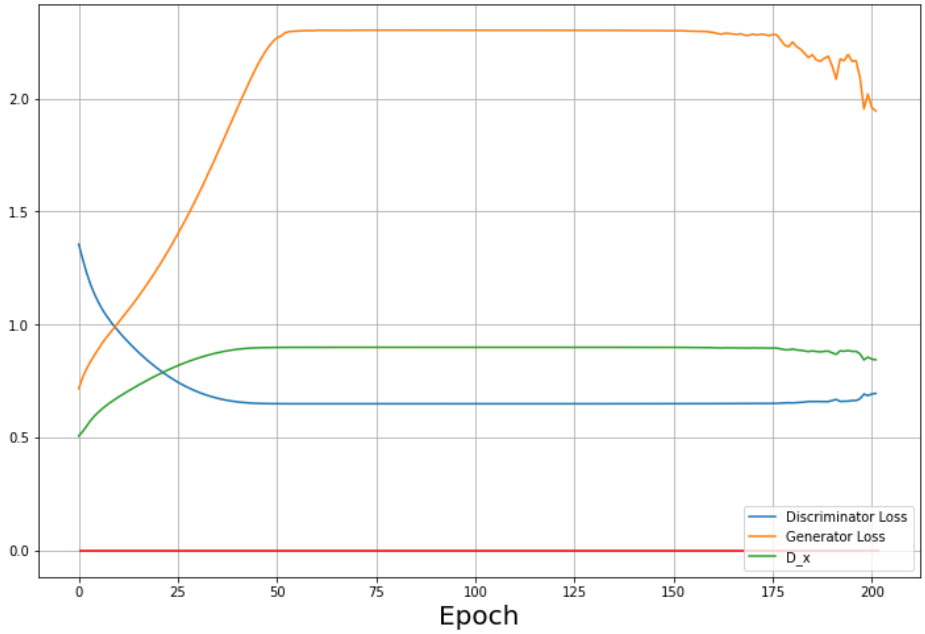

Figure 8: Generator and Discriminator loss over 600 epochs as well as D(x) training on the secondary experiment

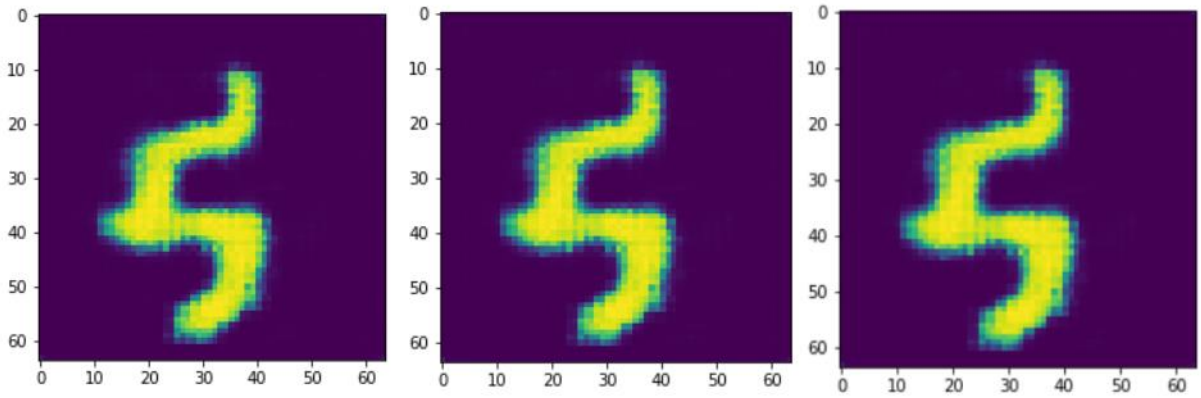

Table 8: Images produced by the Generator network after 200 epochs of training on the secondary network or experiment

3.4 Performance Metrics

In traditional black-box machine learning models, common performance metrics used to evaluate systems are accuracy, precision, recall and f1-score. However, analysing the quality of results obtained by generative models is more complex. This is because the nature of the problem is to yield novel results, which may at times detect and reproduce obscure patterns of the data distribution used in training. Therefore, it is especially important that the training set used be balanced and represent realistic samples.

There are a few performance metrics used to evaluate generative models, with varying degrees of accuracy and significance. The Inception Score metric appears to be the most significant one used for evaluating computer-generated images. In fact, this metric is cited to garner evaluation results which mimic human-level quality. This metric consists of passing the generated images in a pre-trained neural network, which evaluates, at each layer of the network, the features of the image. However, these networks can often produce false positive results by algorithms which retain too much information from their training data.

For this project's implementation, we attempted to calculate the Inception Score, by employing the pretrained InceptionV3 model, found on the *torchvision* module. However, our efforts were unsuccessful because this network is trained on coloured images, which contain 3 RGB channels. As this network is pre-packaged and pretrained, it was not possible to change this network's requirement. Therefore, because the MNIST dataset contains grayscale images, we could not calculate our model's inception score.

4. Conclusions

Our project's aim was to learn about image generation. To achieve this goal, we implemented a DCGAN which produced images resembling the MNIST data distribution. Two convolutional networks were built: a Discriminator and a Generator. The Discriminator network aimed to output a probability of an image belonging to the distribution; whereas the Generator was responsible for generating images that mimicked the MNIST dataset. We followed closely the state-of-the-art implementation presented by Radford et al., learning about the different hyperparameters they used as well as the architecture they introduced. To train our model we calculated the respective losses of the two networks, using the minmax value function presented by Goodfellow et al.. The DCGAN produced good results, however critically evaluating the model proved difficult as the Inception Score could not be calculated on our grayscale images.

The results achieved were very promising as the network reached equilibrium with loss values similar to those stated in the original paper. The output images also look very good and closely resemble the number six. They also are very similar to the training images which further reinforces our positive conclusion.

We learnt a lot about analysing images, specifically how GANs work and some of their real-world applications which was particularly interesting. We also learnt a lot about other image analysis models as we researched the way to implement this project, such as CNNs.

5. Reflections

In this project we were able to learn about generative machine learning models, by implementing a DCGAN. Learning about this topic was extremely fascinating. Overall, this project presented a great learning experience for us.

The most challenging part of this project was implementing the training of the two networks. This is because we had to do extensive research on the most optimal hyperparameters and optimizers. GANs are very sensitive to these values and can result in a mode collapse if not well parametrized. Our implementation does not yield strikingly successful results. This is unexpected because the model we created reflects the characteristics described by Radford et al.

Future improvements to this project may include using different datasets, like the CIFAR100, to see if our model is able to generalize and reproduce other distributions. Moreover, by doing so with an RGB dataset, we would be able to calculate the inception score to evaluate our model's performance and quality of generated images.

Overall, for this project's implementation, the percentage of code borrowed is around 10%. We aimed to recreate the original DCGAN model, and therefore used their network's parameter specifications and values. We mostly referenced Radford et al.'s tech report, as well as Goodfellow et al.'s paper. Moreover, the layout of our code takes inspiration from the structure presented in Lab 6.

We worked very well as a team and contributed equally to all aspects of the project. We scheduled zoom meetings frequently to discuss our project and implement it together. We used the university's OneDrive to send each other useful resources, and to keep track of different updates we made to both our code and our report. We very much enjoyed this project as we learnt a lot and notably also managed to generate strong results.

We would have liked to implement YOLOv3 as it was a very interesting algorithm that has a wide range of uses. However, ultimately, we decided to revert to our original idea to implement a DCGAN. We encountered difficulties when computing the loss calculations needed for backpropagation. The lack of documentation and information on YOLOv3 was simply not sufficient to allow us to implement it within the time frame. In addition, combined with the size and complexity of the MS COCO dataset we decided to implement a DCGAN.

6. References

Alec Radford & Luke Metz, 2016. *DCGAN research paper.* [Online]
Available at: https://arxiv.org/pdf/1511.06434v1.pdf
[Accessed April 2021].

Ian J. Goodfellow, 2014. *GAN research paper.* [Online]
Available at: https://arxiv.org/pdf/1406.2661.pdf
[Accessed April 2021].

MachineLearningMastery, 2019. *Developing a GAN for MNIST.* [Online]
Available at: https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-an-mnist-handwritten-digits-from-scratch-in-keras/
[Accessed April 2021].

Wikipedia, 2021. *Explanation and info on MNIST dataset.* [Online]
Available at: https://en.wikipedia.org/wiki/MNIST_database
[Accessed April 2021].

BBC news, 2021. *Unlocking the mystery of the Dead sea scrolls.* [Online]
Available at: https://www.bbc.co.uk/news/world-middle-east-56842712
[Accessed April 2021].

Plos One, 2021. *Unlocking the mystery of the Dead sea scrolls.* [Online]
Available at: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0249769
[Accessed April 2021].

Wikipedia, 2021. *Isaiah Dead sea scroll inforamtion.* [Online]
Available at: https://en.wikipedia.org/wiki/Isaiah_Scroll
[Accessed April 2021].

FreeCodeCamp, 2018. *Introduction to GANs.* [Online]
Available at: https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394/
[Accessed April 2021].

MachineLearningMastery, 2019. *Inception score implementation.* [Online]
Available at: https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/
[Accessed April 2021].

Arxiv.org, 2018. *A note on the Inception Score.* [Online]
Available at: https://arxiv.org/abs/1801.01973
[Accessed April 2021].

6.1 Image Reference

Figure 1: MachineLearningMastery, 2019. *Developing a GAN for MNIST.* [Online]
Available at: https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-an-mnist-handwritten-digits-from-scratch-in-keras/
[Accessed April 2021].

Figure 2: Wikipedia, 2021. *Isaiah Dead sea scroll inforamtion.* [Online]
Available at: https://en.wikipedia.org/wiki/Isaiah_Scroll

Figure 3: FreeCodeCamp, 2018. *Introduction to GANs.* [Online]
Available at: https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394/
[Accessed April 2021].

Figure 4: Ian J. Goodfellow, 2014. *GAN research paper.* [Online]
Available at: https://arxiv.org/pdf/1406.2661.pdf
[Accessed April 2021].

Figure 5: Alec Radford & Luke Metz, 2016. *DCGAN research paper.* [Online]
Available at: https://arxiv.org/pdf/1511.06434v1.pdf
[Accessed April 2021].