Project Progress Report 1

February 14, 2023

1 Project Progress Report 1

The goal of this project is to develop a deeper understanding of generative models, specifically GAN (Generative Adversarial Network) and VAE (Variational Autoencoder). Generative models are classified as unsupervised machine learning algorithms.

Generative Models learn to produce new data that is comparable to the data they were trained on. These models are trained on a dataset in order to discover the underlying patterns and distributions of the data, and they may be utilized for tasks like creating images or texts. After being trained, generative models may be used to create fresh, artificial data that is comparable to the starting set.

1.1 Generative Adversarial Network

A generator network and a discriminator network are the two neural networks that make up GANs.

The discriminator network learns to discriminate between actual data and artificial data produced by the generator network while the generator network learns to create new data. The two networks are trained in tandem in a way that resembles a game, with the discriminator trying to accurately distinguish between genuine and false data while the generator seeks to produce data that can trick the discriminator into believing it is real.

1.2 Variational Autoencoder

They are a type of generative model that discovers how to create new data by encoding it into a lower-dimensional space and then decoding it to restore it to its original form. To understand the underlying distribution of the input data and produce new samples from it, VAEs employ probabilistic latent variable models.

1.3 Comparing GAN and VAE

It is well understood that the input data's latent representation is not precisely specified in GANs. The generator uses a noise distribution to generate fresh samples. On the other hand, VAEs discover a probabilistic model of the input data's latent space. VAEs learn a latent representation of the data that can be used to generate new data, while GANs learn to generate new data by adversarially training a generator and discriminator network. The general consensus also is that compared to GANs, VAEs are often more stable and simpler to tune. GANs can be challenging to train and may have issues like mode collapse, in which the generator only generates a small number of output samples but this is something that will be experimentally explored in the upcoming weeks.

2 Exploring MNIST Dataset

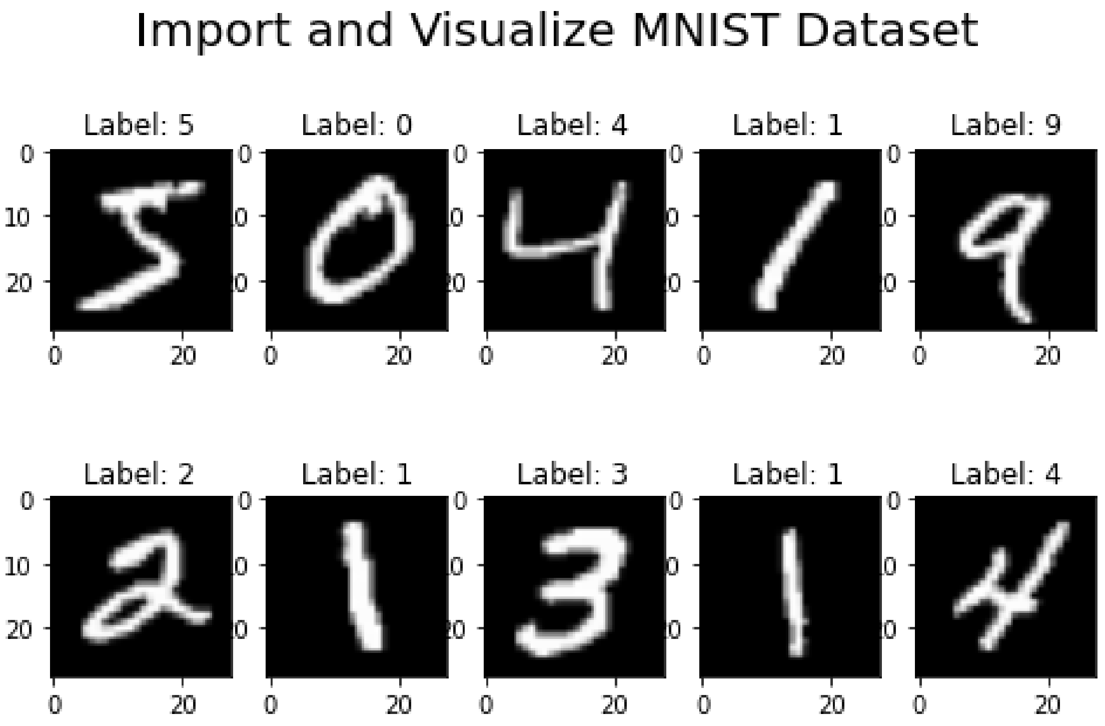
Before developing complex models using GAN and VAE, it is more prudent to apply these models to the handwritten digits dataset to develop a basic understanding of how they function. We first explore the MNIST dataset, and develop a simple convolutional neural network to classify the digits. This gives us a better understanding of both CNNs and their suitability for image data.

Eventually the goal is to generate the MNIST data using the generative model framework. Consequently this will allow us to compare the internal architectures of neural networks used with GAN and VAE to gain a better understanding of the differences between the two models. This exercise is planned for the next project report.

2.1 Future Plans

The upcoming week will be dedicated to developing simple convolutional networks to satisfy the internal requirements for GAN (generator and discriminator) and also for VAEs requirement of encoding and decoding data to-and-from higher to lower dimensions. The plan is to tweak the hyperparameters of these neural networks and compare the performance differences between the two models.

```
[13]:
```



Project Progress Report 2

This week was focussed mostly on developing a good understanding of GANs and their implementation in Keras. At a high level view, there are several types of GANs:

1. Vanilla GAN: Generator generates the images which the discriminator classifies (typically using neural networks)

2. Conditional Gan (CGAN): Both generator and discriminator are provided some extra information to converge faster and improve accuracy.

3. Deep Convolutional GAN (DCGAN): A deep convolutional network is used, which is well suited for images. ReLU or LeakyReLU activation functions are used along with adam optimizer.

Initial results from Vanilla GAN

➔ We initialize two neural network models, the generator, and discriminator with a simple architecture. Both models are 4 layers deep.

➔ The generator's last layer produces the pixel values and uses the 'tanh' activation function.

➔ The discriminator's last layer produces 0/1 to classify as either fake or real, and uses the 'sigmoid' activation function.

➔ Both architectures use the "binary_crossentropy" loss function and the "adam" optimizer. Also Leaky ReLU activation function is used for all the layers except the last.

➔ The model was trained for 5000 epochs, with batch size of 128 which took about 15 minutes on google colab using GPU.

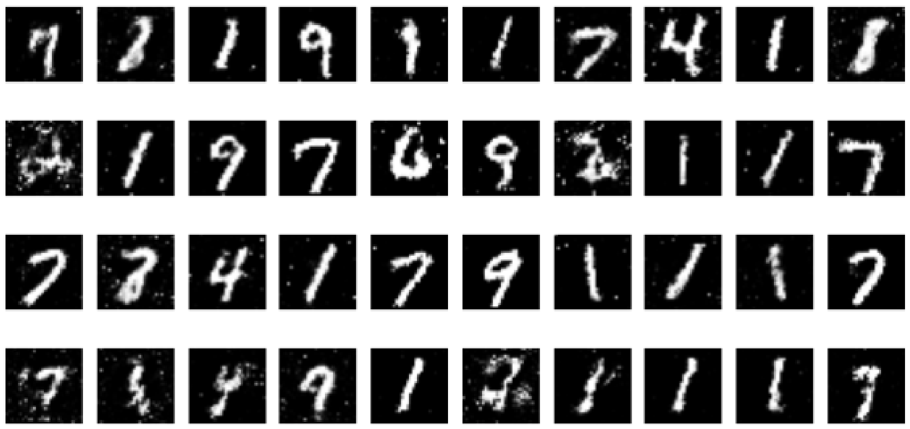


Fig 1. Vanilla GAN trained for about 15 minutes on google colab

Initial results from Deep Convolutional GAN

➔ Similar to Vanilla GAN, we initialize two neural network models. This time we introduce convolution layers into the network.

➔ To improve training speed "dropout" technique was also utilized for both neural network models.

➔ "Binary_crossentropy" loos function, "adam" optimizer and Leaky ReLU activation functions are used similar to vanilla GAN.

➔ The model was trained for 10000 epochs, with batch size of 128 which took about 30 minutes on google colab using GPU.

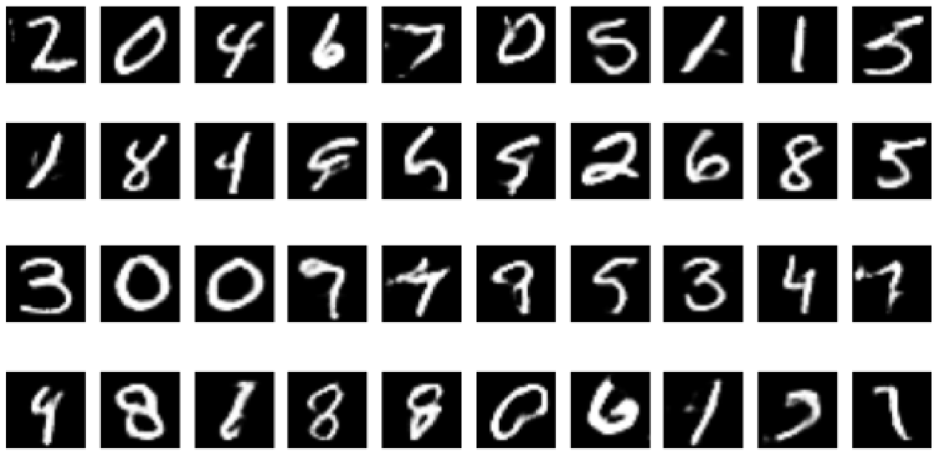


Fig 2. Deep Convolutional GAN trained for about 30 minutes on google colab

Future Work (for next progress report)

➔ Will explore VAE in more detail and understand it's architecture

➔ Will develop a prototype for VAE on the MNIST Dataset

➔ Explore the properties of MNIST to further understand the data and how the architecture can be modified to adapt to it.

Project Progress Report 3

This week was utilized to understand autoencoders and their architecture, variational autoencoders and a model was built to generate the MNIST dataset.

Autoecnoders and Variational Autoencoders (VAE)

An autoencoder is an unsupervised neural network that learns a compressed representation of the input data, called the latent representation or code. The objective of an autoencoder is to minimize the reconstruction error, which measures the difference between the input data and the reconstructed data (which is typically mean squared error (MSE) or binary cross-entropy (BCE)). Autoencoders cannot generate data without knowing the original data.

In a VAE, the encoder network maps the input data to a distribution over the latent space, and the decoder network maps the latent distribution to the output data. The objective of a VAE is to minimize the reconstruction error and the KL divergence between the latent distribution and a prior distribution, usually a standard normal distribution. VAE can generate new data by sampling data from the learned latent space which is fed as input to the decoder network.

Implementation of VAE for MNIST Dataset

Architecture

- **Encoder Network:** consists of the input layer, two convolutional layers of size 32 and 64 which are then flattened into a dense layer towards the end of the network

- **Latent Distribution & Sampling Layer:** the mean and variance values computed by the encoder network based on size of latent dimensions (3 in this case) which are then sampled to generate a latent distribution which allows backpropagation through the sampling layer (reparameterization trick)

- **Decoder Network:** inputs data from the latent distribution is a mirror image of the encoder network with transposed convolutional layers for upscaling the data from latent distribution to produce output

- **Loss Function**: a combination of *reconstruction error* and the *KL divergence* between the latent distribution and the normal distribution.

Results

The training took around 30 minutes for 30 epochs with a batch size of 128 using Google Colab with GPU.
We visualize some data generated by the model and also visualize the latent space learned by the model.

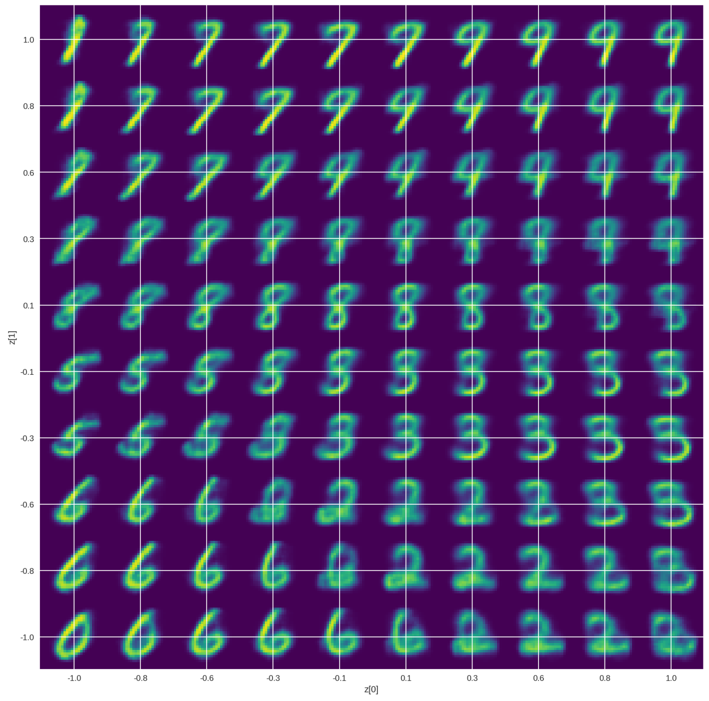


Fig 1. Results of generating data (latent space is the normal distribution)

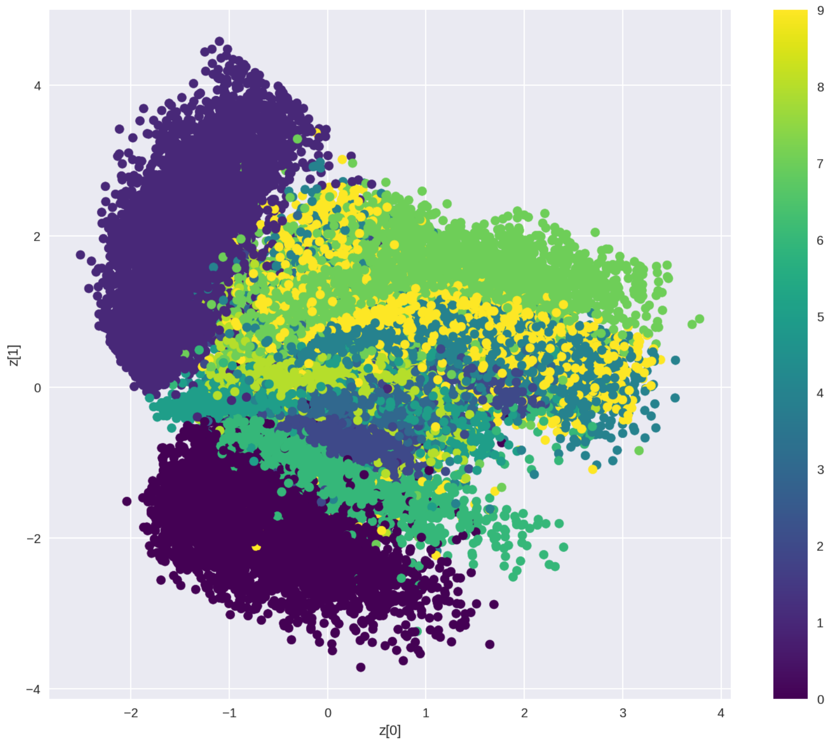


Fig 2. Latent space clusters for different digit classes from MNIST

Future Work (for next progress report)

- ➔ Will work on developing a better understanding of the both GANs and VAEs to improve the current models
- ➔ Will work on understanding the differences in training the models
- ➔ Will look into implementing these models using other datasets

Project Progress Report 4

Overview

This week was utilized to expand the initial models of GAN and VAE to test the fashion MNIST dataset. As we observe decent results for digits MNIST, testing these initial models on fashion MNIST would allow for deeper understanding of how the network architecture scales to more complex datasets. Furthermore, research for appropriate qualitative and quantitative measures for comparing GAN and VAE was conducted and these measures are briefly discussed in this report.

Observed results for fashion MNIST dataset

These are preliminary results that will be improved in future work by tweaking the model parameters. Similar architectures of GAN and VAE are used to train and test on the fashion MNIST dataset with a few minor changes. The most notable change is that 8 parameters are utilized to represent the latent dimension for GAN and 2 for VAE. Other changes are minor, and involve things like data set size, training epochs, etc.

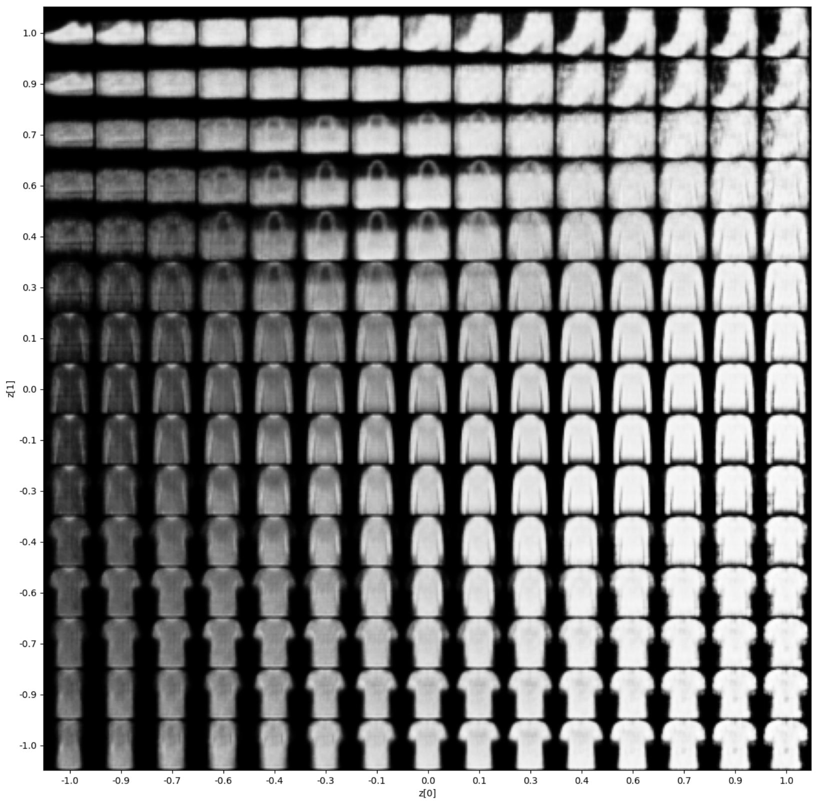


Fig 1. Result of generating data on VAE (2 parameters for latent dimensions)
Trained for 30 minutes on google colab GPU

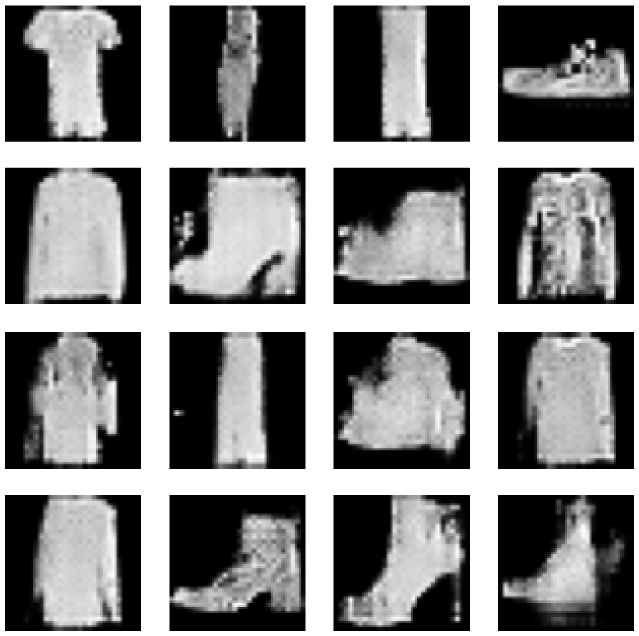


Fig 2. Result of generating data on GAN (visually not better than VAE, will improve in future work)
Trained for 15 minutes on google colab GPU

Potential Parameters for Comparison of GAN and VAE

1. **Network Size**: a measure of the number of trainable parameters in the architecture along with a few other components is a good estimate for the network size

2. **Computational Resources**: the hardware used to train the networks along with the training time, memory utilization etc. can be compared

3. **Training and Evaluation**: another measure to evaluate the performance using metrics such as *FID, Inception Score* etc. to measure the quality and diversity of the generated samples, as well as the similarity to the original dataset.

4. **Overall Application Specific Performance**: it is integral to evaluate the performance of each for the specific task of image generation on the MNIST and fashion MNIST datasets. The goal is to document a qualitative measure by observing the final results (sharpness of image, how distinct are the results compared to original dataset, usability of results for practical applications etc.)

Future Work

- Will work on narrowing down the specific parameters based on which both GAN and VAE models will be compared with each other.

- Will tweak the model's architecture to potentially improve the final results and document these changes.

- Will research the idea of training the GAN architecture using the latent space learned by the VAE architecture which will lead to novel results.

Project Progress Report 5

Overview

This week was focussed on understanding the difference in:

- Network Size
- Computational Difference
- Qualitative Nature of Output

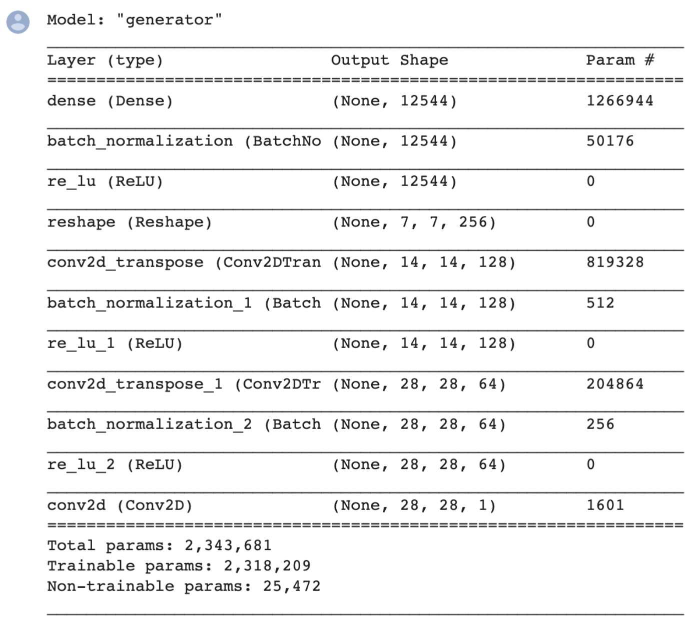
For both the GAN and VAE architectures.

Furthermore, research was done to understand why there is a difference in the architecture, especially based on these parameters for both GAN and VAE that produce similar levels of output in terms of quality.

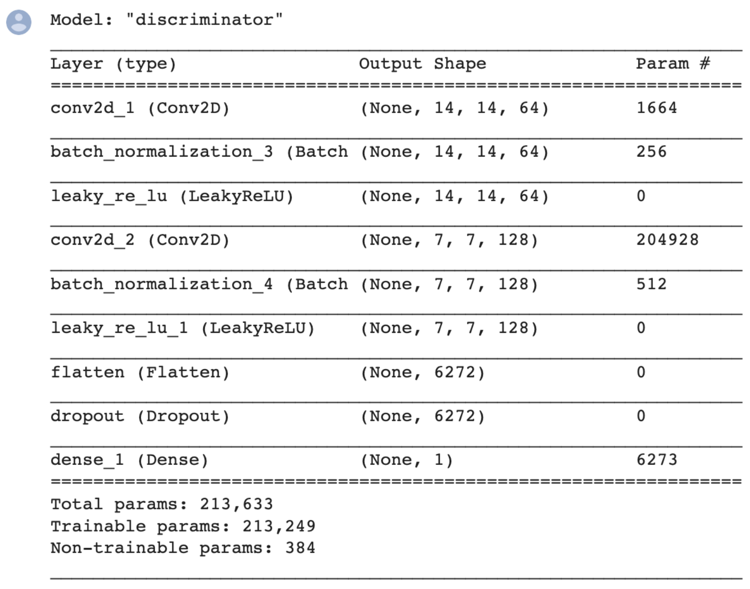
Network Size

The network sizes mentioned in this report are for very basic, but functional architectures of GAN and VAE which were tested on the Fashion MNIST dataset.

GAN



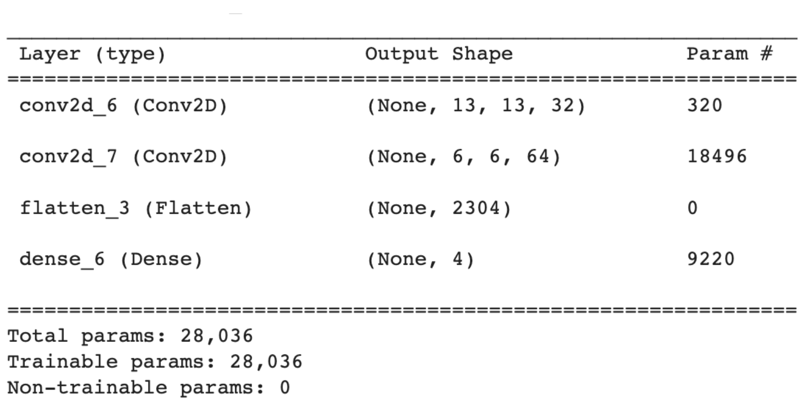
```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 12544)             1266944
_________________________________________________________________
batch_normalization (BatchNo (None, 12544)             50176
_________________________________________________________________
re_lu (ReLU)                 (None, 12544)             0
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 128)       819328
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
re_lu_1 (ReLU)               (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 64)        204864
_________________________________________________________________
batch_normalization_2 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
re_lu_2 (ReLU)               (None, 28, 28, 64)        0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 1)         1601
=================================================================
Total params: 2,343,681
Trainable params: 2,318,209
Non-trainable params: 25,472
_________________________________________________________________
```



```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 14, 14, 64)        1664
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 128)         204928
_________________________________________________________________
batch_normalization_4 (Batch (None, 7, 7, 128)         512
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dropout (Dropout)            (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 6273
=================================================================
Total params: 213,633
Trainable params: 213,249
Non-trainable params: 384
_________________________________________________________________
```

Fig 1. Size of GAN's discriminator and generator (number of trainable parameters)

VAE



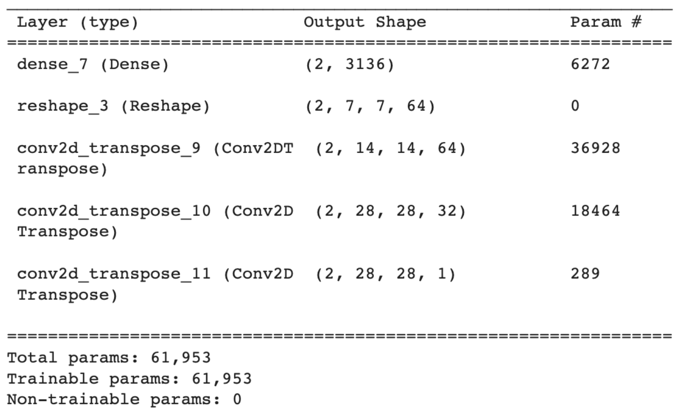
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 13, 13, 32)        320

 conv2d_7 (Conv2D)           (None, 6, 6, 64)          18496

 flatten_3 (Flatten)         (None, 2304)              0

 dense_6 (Dense)             (None, 4)                 9220

=================================================================
Total params: 28,036
Trainable params: 28,036
Non-trainable params: 0
```



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (2, 3136)                 6272

 reshape_3 (Reshape)         (2, 7, 7, 64)             0

 conv2d_transpose_9 (Conv2DT  (2, 14, 14, 64)          36928
 ranspose)

 conv2d_transpose_10 (Conv2D  (2, 28, 28, 32)          18464
 Transpose)

 conv2d_transpose_11 (Conv2D  (2, 28, 28, 1)           289
 Transpose)

=================================================================
Total params: 61,953
Trainable params: 61,953
Non-trainable params: 0
```

Fig 2. Size of VAE based on size of encode and decoder network (number of trainable parameters)

Summary of Observations

- It is observed that the number of trainable parameters in GAN is much higher than VAE which indicates that larger GAN models are needed to produce output of similar quality to a VAE

- The training for GAN took around 45 minutes, compared to only 20 minutes for VAE on Google Colab GPU computers to get similar quality output. This is due the difference in model size and the fact that GANs require multiple forward and backward passes through both the generator and discriminator networks at each training iteration.

- Based on the previous week's testing and observations, it can be noted that GANs tend to produce more realistic and visually pleasing output compared to VAE. This could be due to the fact that GANs train to recreate the input data whereas VAEs train to produce samples that resemble the mean of the input data.

- However it is also observed that the output from GAN, although visually pleasing, is often inconsistent or lacks diversity. In contrast, output from VAE is much more diverse because it captures the underlying latent distribution of the data.

Concluding Remarks

Over the course of the semester, both VAE and GAN architectures were studied in detail and implemented on both MNIST and Fashion MNIST datasets. The models were tweaked to produce visually pleasing outputs and these tweaks were documented. The final report will now be prepared documenting this process and providing insight on the differences between the GAN and VAE architectures and also the nature of the output generated by both these models.