GANs for synthetic image generation

Martin Ferianc, Divyansh Manocha
{mf2915, dm2515}@ic.ac.uk, CIDs:{00984924, 01053537}

1. Introduction

This report looks at training and testing of generative adversarial networks (GANs) [1] to generate synthetic handwritten digit images from the MNIST dataset [2]. Their performance has been evaluated visually through inspection and an inception score that was calculated using a LeNet [3].

The MNIST dataset consists of 60,000 examples of handwritten digits for training and 10,000 samples for testing. Each example is a gray-scale 28×28 image, which was zero centred to enable backpropagation of both positive and negative gradients during training.

2. Deep-Convolutional Generative Adversarial Network

GANs are an example of generative models. They attempt to learn a distribution of the data p_{data} through maximum likelihood estimation to arrive at an approximation of the model p_{model}. Deep-convolutional GAN (DCGAN) [1], similarly to class-conditional GAN (CCGAN) [4], does it through the use of convolution and deconvolution layers.

2.1. Architecture

The design of the networks consists of two separate parts: a *Generator* and a *Discriminator*. The task of the generator is to generate new examples from random noise and the task of the discriminator is to distinguish these generated examples from real ones.

2.1.1 Generator

Our architecture is based on the original DCGAN, seen in Figure 1. First, the 1D noise input of length 100 is processed through a fully connected (Dense) layer. Afterwards, it is processed by batch normalisation (BN) [5], followed by rectified linear unit (ReLU) [6] activation. We use BN after each Dense or convolution to provide regularisation to avoid overfitting and smooth normalised propagation of signals. The nonlinear ReLU is used to address saturation problems and vanishing gradients as a result of it. BN and ReLU are again followed by a Dense layer and a BN and ReLU, followed by reshaping function to reshape the output into 3 dimensions. Then, it uses an upsampling layer, which performs nonlinear deconvolution and thus generates new data that is then later refined by convolution, which is again followed by BN and ReLU. Afterwards, the features are again upsampled to the original dimensions, followed again by convolution.

2.1.2 Discriminator

A generator alone just creates random noise and it requires guidance onto what images it should produce. The discriminator learns the guidance through learning the decision boundary between input images being real or fake.

The discriminator begins with convolution to extract features from the original image followed by BN and an ELU activation which does not saturate its output at zero as ReLU does, making training more efficient. Afterwards, the features are refined by yet another convolution and the weights are regularised through BN and outputted through ELU into a pooling layer which selects the most significant features in terms of magnitude. Lastly, it flattens the features which are refined by Dense layers into a probability. The last stage of the discriminator outputs a value through a sigmoid, as the activation layer, to predict whether the input is real or a fake.

This work explores the capacity (number of channels) of both networks by changing the parameter F. By changing the networks' capacity both networks will be able to extract/generate more distinct features which can result in a more detailed image and better prediction. Another hyperparameter which can be evaluated upon is the balancing of the discriminator and generator, by making the discriminator deeper. While the architecture of the generator is fixed, we propose to observe the effect of empowering the discriminator by including more convolutions - more powerful feature extractors by changing D, as depicted in Figure 1.

2.2. Training

2.2.1 Loss-Function

The training of this joint network is achieved by alternating gradient descent on the discriminator network, while the generator's weights are frozen and vice versa. In general, GAN is optimising against a minmax game:

$$\min_G \max_D V(G,D) = E_{\mathbf{x}\sim p_{data(\mathbf{x})}}(log\ D(\mathbf{x})) + E_{\mathbf{z}\sim p_{z(\mathbf{z})}}(log\ (1 - D(G(\mathbf{z}))))$$

where the G corresponds to the generator and D corresponds to the discriminator and $\mathbf{x}$ corresponds to the real image data and $\mathbf{z}$ corresponds to the input noise, in this case a 100 dimensional Gaussian with 0 mean and 1 variance. This is a similar formula for the binary cross-entropy error which is used as the loss function for training both the generator and the discriminator.

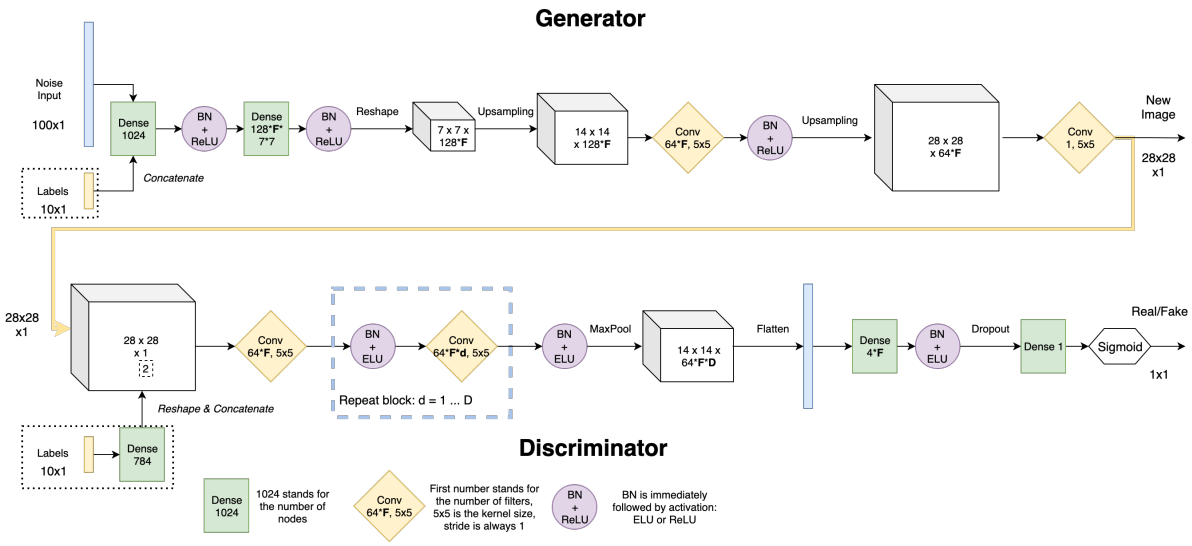

Figure 1: Proposed generative adversarial network architecture.

The objective of the discriminator is to maximise the recognition rate, which corresponds to $\max_D$ The generator, wants to generate images with the highest possible value of $D(G(\mathbf{z}))$ to fool the discriminator, which corresponds to minimising in $\min_G$. The equilibrium is achieved when $p_{model}(\mathbf{x}) = p_{data}(\mathbf{x})$ and the discriminator is unsure about images being real or fake and giving $D(\mathbf{x}) = \frac{1}{2}$.

A significant hyper-parameter is the batch size B, which refers to the number of samples each network sees during training at once, per iteration. The batch size can delay or prevent mode collapse by encouraging diversity during training and later during inference.

Overall, this work assumes three hyper-parameters for experimentation: the capacity F, depth of discriminator D and the batch size B. The multipliers B, F ranged from 1 to 3 while B was chosen to be 32, 64, 128. In total giving 27 combinations[1].

We expect that the increasing batch size will accelerate the training, while increasing D will delay mode collapse. Increasing F can result in more detailed samples and better prediction by discriminator.

2.3. Evaluation

The performance is evaluated by using the training accuracy of the discriminator, visual inspection and losses of the generator and discriminator to inform about overfitting or underfitting.

Each combination was trained over 25 epochs with Adam [7] being the optimiser with an initial learning rate 0.002 both for discriminator and generator. The loss functions of both generator and discriminator as well as the training accuracy of the discriminator can be seen in Figures 2, 3, 4.

We have observed that the simplest combination with F and D both being 1 and with the batch size 32 achieved the best overall results in terms of accuracy of the discriminator and loss convergence. Samples taken from this network can

[1]For $F = 3, D = 3$ the GPUs run out of memory.

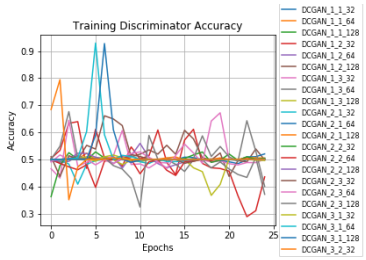

Figure 2: Training discriminator accuracy for DCGANs, the first number in the name means the size of F, the second number the depth D and the last number batch size B.

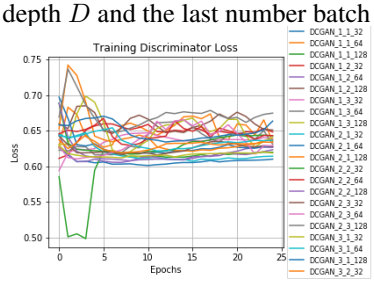

Figure 3: Descriptor loss for DCGANs, the first number in the name means the size of F, the second number the depth D and the last number batch size B.

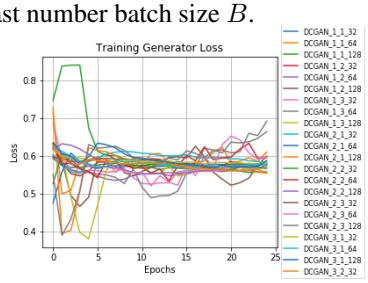

Figure 4: Generator loss for DCGANs, the first number in the name means the size of F, the second number the depth D and the last number batch size B.

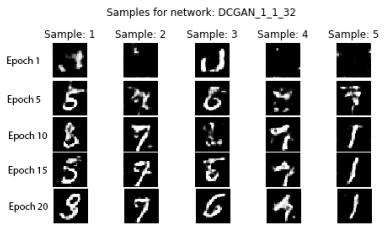

Figure 5: Sample images for DCGAN with $F = 1$, $D = 1$ & $B = 32$.

be seen in Figure 5. The discriminator accuracy stabilised at the ideal 0.5.

It was interesting to note that the generator has 6,768,129 parameters while the discriminator has 307,737 parameters. This network contradicts our initial hypothesis that a more complex and balanced network are going to be better fit. The simpler network outperforms the more complex ones, as predicted through Occam's razor, and it extracts enough features given the complexity of the data. Computation-wise this network requires the least resources in comparison to other options.

We applied virtual batch normalisation to this network, by randomly choosing half of the batch size in each epoch and normalising the other half with respect to the chosen one. The second half was different per each iteration. The loss functions as well as discriminator accuracy can be seen in Figures 13, 14 & 15.

However, we have not seen a significant improvement and visually it produced worse results than unnormalised network. That might be caused by too overextending normalisation, given that it is already present after each feature extractor both in generator and discriminator networks.

We have not observed the mode collapse in general with the given combinations. The most common issue, especially seen in the more complex architectures, was that even though the losses stopped decreasing the generator stopped improving and visually judging, it produced unrecognisable numbers.

3. Class-Conditional Generative Adversarial Network

In comparison to the DCGAN, CCGAN accepts labels both in discriminator and generator, which enables a generation of samples for a particular class which was previously impossible.

3.1. Architecture

A subtle change in the network from the one seen in Figure 1, the label is simply concatenated into the Dense layer in the generator with original number of nodes. In the discriminator, the labels are processed first by a Dense layer with 784 nodes and then reshaped into the image dimensions and concatenated to the original input.

3.2. Training

With the influence of the label as an added input the cost function now changes into:

$$\min_G \max_D V(G,D) = E_{\mathbf{x},l\sim p_{data(\mathbf{x},l)}}(log\ D(\mathbf{x},l)) + E_{\mathbf{z}\sim p_{z(\mathbf{z})},l\sim p_l(l)}(log\ (1 - D(G(\mathbf{z},l),l)))$$

where l represents the label and $p(l)$ represents the distribution from which the label was uniformly selected for generator training and $p_{data(\mathbf{x},l)}$ corresponds to the the original distribution of labels. The objective does not change from binary cross-entropy except the fact that the optimisers now accepts an additional input, the label l.

We again perform experiments over the capacity $F = 1, 2, 3$, depth of discriminator $D = 1, 2, 3$ and the batch size $B = 32, 64, 128$. In total giving 27 combinations, each combination is being trained over 25 epochs with Adam optimiser and an initial learning rate being 0.0002 for both discriminator and generator.

3.3. Evaluation

Given that the networks accept labels we can measure inception score which was measured against a LeNet [3], in Listing 1, which achieved 99.4% on the training set and 98.8 % accuracy on the test set respectively after 5 epochs of training with the Adam optimiser. By visual inspection and adding inception score into the observed metrics, depicted in Figures 7, 16, 17 & 18 we conclude that again the simpler networks outperform the more complex ones, but only by a small margin, that could be broken by longer training times. The best network was with $F = 1, B = 64, D = 1$, giving 307,385 and 7,572,865 parameters per discriminator and generator respectively and **94.8%** inception score. The batch-size increased suggesting that the the gradient descent should be smoother by averaging over more samples at once. This network required the least time to train and it had the smallest memory footprint for its parameters.

We observe that by including the label into the input, in comparison to the DCGAN, the quality of the sampled images improved as seen in Figure 6. The training of the generator is first biased towards more distinct numbers such as 0 or 2, which were overall learnt first and their distinctiveness can be demonstrated through the first two components through principal component analysis (PCA) in Figure 8. We also observe in Figure 16 that the discriminator accuracy stays rather constant since the beginning of the training in comparison to the DCGAN's varying discriminator accuracy observed in Figure 2. In general, the generator was more challenging to train in comparison to the discriminator whose accuracy achieved the ideal state of 0.5 already in the early epochs.

Virtual batch normalisation again did not provide any significant improvement, probably because of already including BNs. However, label smoothing, through converting the labels into decimal one-hot encoding boosted the inception score on average by 3% per epoch, which gave the best overall inception score **97.6%** for the best network. The confusion matrix can be seen in Figure 8. The most

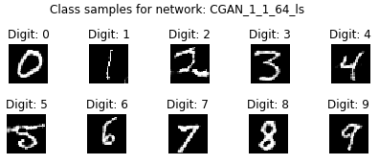

Figure 6: Sample images for CCGAN $F = 1$, $D = 1$ & $B = 64$ with label smoothing.

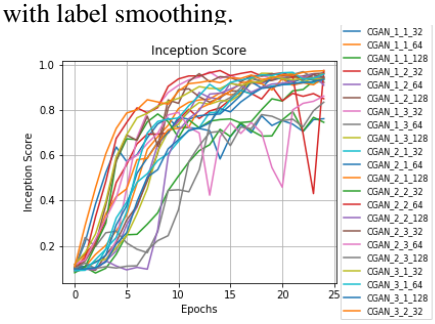

Figure 7: Inception scores for CCGANs.

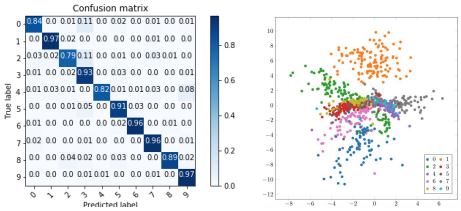

Figure 8: Left: Confusion Matrix for CCGAN $F = 1$, $D = 1$ & $B = 64$ with label smoothing Right: MNIST test data projection through PCA.

mis-classified numbers were 2 and 3 which share certain details in their shaping. It should also be noted here that the MNIST dataset does not have equal proportions of samples for each class (number), which can affect the accuracy observed. By inspecting the losses as well as the inception score, we have not observed the mode collapse and our network generates a rich variety of different samples for each number.

4. Training with Synthetic Data

4.1. Generation

The samples obtained from the optimal CCGAN found in Section 3 can be used to retrain the inception network. The same number of images were kept for all training data, also with respect to individual classes, and shuffled with random seeds to ensure a fair comparison can be made.

4.2. Basic Approach

Considering the accuracy loss during testing to the one obtained using 100% real training data (98.8%) in Figure 9, it is clear that more real data and less synthetic data gives better performance. Interestingly the *training* accuracy loss seems to be smaller when either the synthetic or real dataset dominates the input.

4.3. Strategy Reconsideration

Clearly even 10% of real samples gives a high recognition rate, being the worst obtained performance. The two hyperparameters of the network that could be explored were

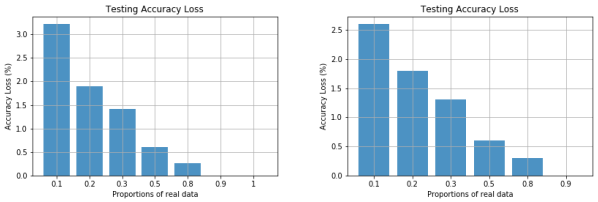

Figure 9: Testing accuracy loss with mixed datasets (Left: Unchanged, Right: Improvements).

the learning rate and batch size. To choose the optimal learning rate requires a trade off between the convergence time and reliability. Fine-tuning the learning rate of the network (around the previously found optimal value) had little effect (+/- 5%) on the recorded performance. The results of optimising the inception network parameters are shown in Figure 11. A learning rate of 0.001 was still the best for the network. It was also observed that tuning the batch size for learning rates that performed well also did not improve the accuracy by any significant amount (+/- 1%). A batch size of 32 was still the best.

Further analysis, however, showed that class 2 was often mis-predicted as class 0. As a result the percentage of real and synthetic data was altered to achieve better results. Previously all classes had the same percentage of synthetic data, and the same number of samples per class. Interestingly, performance did not always increase with a higher percentage of real data for these classes. The best configuration was found to be to increase the percentage of real data in class 4 to 40%, with all other classes remaining with 10%. The confusion matrices of with and without this adjustment are shown in Figure 10.

4.4. Comparison

The right graph of Figure 9 shows the improvement that is being achieved by altering the combination of synthetic and real data. Although, the overall results qualitatively remain the same, we were able to reduce the difference in the testing accuracy loss achieved by approximately **0.5%**.

From this experiment, the results achieved show that for datasets of smaller dimensions (such as the MNIST handwritten digits) it is possible to use GAN generated images for data augmentation. This has been studied thoroughly in [8]. However, for dataset such as this it may not be practically useful because the GAN itself needs a sparse dataset to learn a varied distribution. If it is not, then generating these images for training data on another model will not produce a varied one; therefore biasing the model.

5. Comparison with Principcal Compoment Analysis

With probabilistic PCA (PPCA) it is possible to generate samples from a learnt data distribution. However, this is beyond the scope of this project. The dataset is small enough to use standard PCA to generate images in a lower dimensional space. Data augmenetation using PCA has pre-

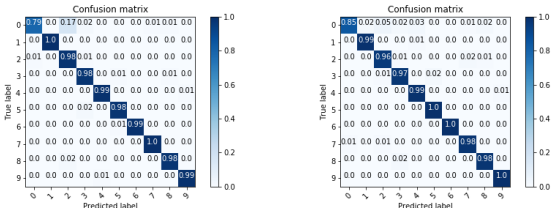

Figure 10: Confusion matrix with mixed datasets (Left: Unchanged, Right: Improvements).

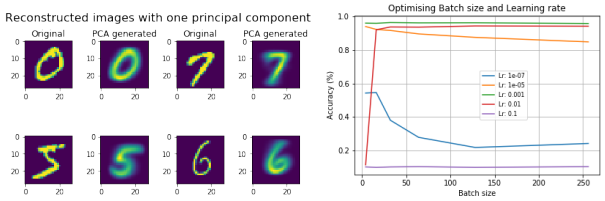

Figure 11: Left: Examples of PCA dimensionality reduction and reconstruction of the data. Right: Test accuracy for inception network.

viously been explored, for example by Krizhevsky *et al.* [9], who used multiples of the principle components to generate images of different intensity and colour.

With this particular dataset, one component is sufficient for a human to visually discriminate between the numbers, as illustrated in Figure 11.

5.1. Generation

From the MNIST dataset, all images belonging to a particular label were treated as the training data to PCA. n components, corresponding to n eigenvectors of the highest eigenvalues, were then extracted. Individual images were then reconstructed into its original size of 28×28 pixels. Hence, this can be considered a generative process.

In order to compare the two datasets, a common metric was required. The inception score was used, as had been used previously in this report to compare the performance of the CCGAN.

5.2. Results

In Section 3, the CCGAN achieved 97.6% accuracy against LeNet. This is what the PCA accuracy is expected to converge to. As Figure 12 demonstrates, there is a large difference in performance increasing the number of principal components initially. The improvement then saturated post $\sim$**64** principal components.

PCA with one component clearly performs considerably worse than than the images generated from the best performing CCGAN model. The inception score was almost 10% lower for PCA generated images with one principal component. The confusion matrices were also plotted, as shown in Figure 19.

5.3. Comparison

It should be noted that GANs are able to produce images by learning more complicated distributions than simple linear PCA. This, however, does mean that they require a much larger dataset than using the latter technique as a generative model. Unlike PCA, GANs have numerous hyperparameters to optimise and are also more difficult to implement. Their deep learning approach also make them a black box, with little explainability and ability to infer the model. PCA, which is a well-known technique, requires fewer computational resources as well and it is much simpler to implement.

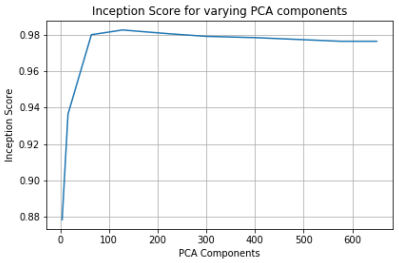

Figure 12: Inception score with varying number of principal components from PCA.

6. Conclusion

In this work we conducted experiments with DCGAN, CCGAN and PCA for generation of sample images based on the MNIST dataset. The best DCGAN and CCGAN observed were with $F = 1$, $D = 1$ and batch sizes $B = 32$ and $B = 64$ respectively. The best inception score that was measured with the best CCGAN architecture achieved **97.6%**. When augmenting the dataset with synthetic GAN generated images, the best performance was achieved when the **entire real** training data was used. When finding the best performance of PCA as a generative model, it was interesting to note that there were a points of significant sudden change in the gradient of Figure 12. Referring to it as a saddle point, found to be at $\sim$**64** components, it was chosen to be the best dimensional space to generate images. This is because higher dimensions introduced a significant time penalty for an accuracy difference of approximately *1.5%*.

We have observed that the more complex architectures do indeed give a better performance by a small margin, however, their use cannot be justified in terms of compute time during inference and training as well as memory consumption. It was found that GANs can be used for data augmentation for the MNIST dataset, as even a small percentage of real data gives high accuracy. Similarly the PCA analysis conducted shows that only 64 of the 784 potential features of the images are required for a high performance.

In the future, the training could be further accelerated by pre-training the discriminator before the training of the generator. We could have also run a Bayesian optimisation to find the best networks, instead of a grid search. Bayesian formulations of GANs [10] can also be used to give a better performance, which usually does not require interventions such as label smoothing or mini-batch discrimination during training. As was mentioned in the discussion of Section 5.3, PPCA is another technique that should be explored for a more thorough analysis of data augmentation.

References

[1] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, vol. abs/1511.06434, 2015.

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

[3] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[4] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014.

[5] S. Ioffe and C. Szegedy, "Batch normalisation: 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, pp. 448–456, JMLR.org, 2015.

[6] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, (USA), pp. 807–814, Omnipress, 2010.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[8] A. Antoniou, A. Storkey, and H. Edwards, "Data augmentation generative adversarial networks," *arXiv preprint arXiv:1711.04340*, 2017.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[10] Y. Saatci and A. G. Wilson, "Bayesian GAN," in *Advances in neural information processing systems*, pp. 3622–3631, 2017.

Listing 1: LeNet Architecture implemented with Keras.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                        activation='relu',
                        input_shape=(28,28,1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(class_dim,
                activation='softmax'))
```

Figure 13: Generator loss for DCGAN with F being 1, D being 1 and trained with batch size B being 32 together with virtual batch normalisation.

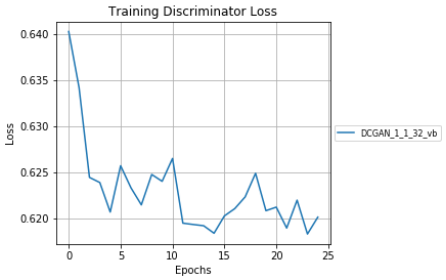

Figure 14: Discriminator loss for DCGAN with F being 1, D being 1 and trained with batch size B being 32 together with virtual batch normalisation.

Figure 15: Discriminator accuracy for DCGAN with F being 1, D being 1 and trained with batch size B being 32 together with virtual batch normalisation.

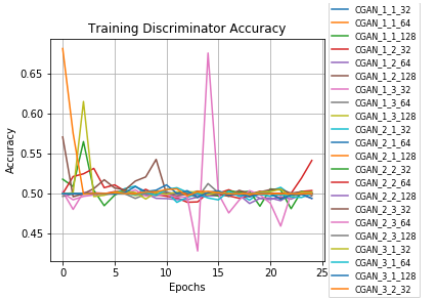

Figure 16: Training accuracy for CCGANs, the first number in the name means the size of F, the second number the depth D and the last number batch size B.

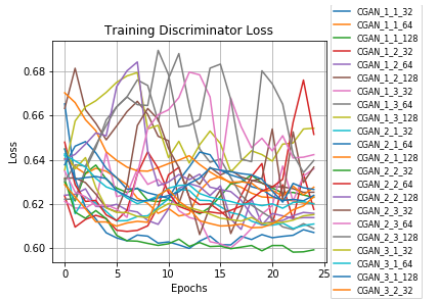

Figure 17: Descriptor loss for CCGANs, the first number in the name means the size of F, the second number the depth D and the last number batch size B.

Figure 18: Generator loss for CCGANs, the first number in the name means the size of F, the second number the depth D and the last number batch size B.

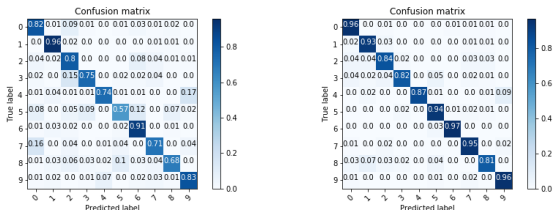

Figure 19: Confusion matrices for PCA generated images of n components being 1 (Left) and 64 (Right).