Generative Adversarial Networks for Synthetic Image Generation

Andre Bharath
CID: 01054089
agb215@imperial.ac.uk

Rajan Patel
CID: 01062228
rnp15@imperial.ac.uk

Abstract

We present an exploration into the generation of synthetic handwritten digit images using Generative Adversarial Networks (GANs). We endeavour to generate synthetic images that are indistinguishable from real images, thus qualitative, and quantitative measures such as the Inception Score are endorsed to assess the quality and diversity of generated images. We endeavour to find an optimal approach that maximises the Inception Score, by taking into consideration preprocessing steps, training procedures and the configuration of hyperparameters for various GAN architectures.

1. Dataset, Definitions & Preprocessing

All experiments in this report have been conducted on the popular MNIST dataset [11], which contains $N = 70,000$ images representing $L = 10$ unique classes of handwritten digits ranging from $0 - 9$. The dataset is split into a training set X_{train} and test set X_{test}, respectively. Thus, the cardinalities of X_{train} and X_{test} are denoted as $N_{train} = 60,000$ amd $N_{test} = 10,000$ respectively. The original bi-level black and white images from NIST[1] [6, 4] are first normalized to fit within a 20x20 pixel^2 box, while preserving their aspect ratio[2]. The images are centered by computing the center of mass of the pixels [14], and translating the image so as to position this point at the center of a 28x28 pixel^2 field[3]. Each single channel greyscale 28x28 pixel^2 image can be flattened and represented mathematically in the form of a column-vector denoted as X_i, where $X_i \in \mathbb{R}^D$, $D = 784$ and $i \in [1, N]$. X_i is then normalized, $X_i \Rightarrow \overline{X_i}$ such that the range of values its constituent pixels P_j can be attributed to undergo the following transformation: $P_j \in [0, 255] \Rightarrow P_j \in [-1, 1]$.

$$\overline{X_i} = \sum_{j=1}^{D} 2(\frac{P_j}{255} - 0.5) \tag{1}$$

The normalization procedure summarised in Equation 1 has been known to facilitate faster convergence to the Nash equilibrium and contribute to the stable training of GAN architectures [15, 18], where the generator and discriminator use tanh and sigmoid output activation functions respectively.

2. Deep Convolutional GAN (DCGAN)

The Deep Convolutional Generative Adversarial Network (DCGAN) [19] is a modified expansion of the original vanilla GAN architecture proposed by Ian Goodfellow [5]. The DCGAN architectures considered in this report adhere to a guideline general architecture, in order to avoid common pitfalls [1].

For the discriminators, strided convolutional layers are used instead of deterministic spacial pooling layers to learn its own spacial downsampling pooling function. The same approach is endorsed for the generators, where fractionally strided convolutional layers facilitate learned spacial upsampling [23]. In case of poor initialisation, batch normalization layers are placed after convolutional layers, in an attempt to avoid diminishing gradients and to aid the flow of gradients between layers during training [8]. Thus, Batch Normalization (BN) (Figure 12) aims to stabilize learning by normalizing the input to each layer to have zero mean $\mu = 0$ and unit variance $\rho = 1$. BN is not applied to the generator output layer nor the discriminator input layer as it is known to cause oscillation and model instability during training. The simple bounded ReLU activation function [17] enables faster learning of the training distribution. The leaky ReLU activation function [13, 25] is similar, but enables a small negative gradient to pass instead of passing a gradient of 0 during back propagation. A modified version of Binary Cross Entropy (BCE) is used as the loss function during training as shown in Equation 3.

The Generator (G) of the first DCGAN architecture considered, denoted as DCGAN-v1 where the architecture summarised in Figure 13, takes in input a 100 dimensional latent vector z randomly drawn from a standard normal distribution.[4] The first layer accepts 784 data points and reshapes the data into 7x7 shape with 16 filters, then two transposed convolution layers up-sample the data into a 28x28 image with 1 channel. A tanh activation is used on the last layer to make sure that the output for the generator function is confined to the interval $[-1, 1]$. The Discriminator (D) takes as input the 28x28 synthetic image output of the generator and outputs the estimated probability that the input image is a real MNIST image. The network is modeled using strided convolutions with Leaky ReLU activations except for the last layer. A sigmoid activation is used on the last layer to ensure the discriminator output lies in the interval of $[0, 1]$.

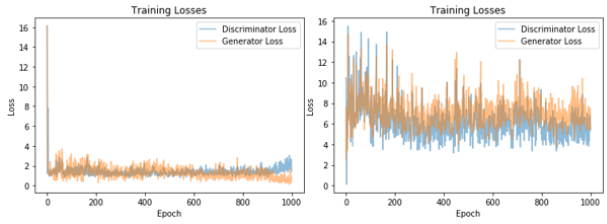

Figure 1. DCGAN-v1 Training Losses for hyperparameter configurations DC 1.1 (left) and DC 1.2 (right)

[1] Refer to Appendix I for more information.

[2] The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm.

[3] For classification with SVM and K-nearest neighbors, the error rate improves when the digits are centered by bounding box [9] rather than center of mass.

[4] DCGAN-v1 was trained by sampling z from a uniform distribution, where the output images seemed similar from a qualitative perspective as well as the training loss history but the losses did not converge as fast.

DCGAN-v1 was trained for 1000 epochs with batch sizes of 128 and an Adam optimizer [10, 20] with a baseline and improved hyperparameter configuration outlined in Table 1. As can be observed from Figure 1 (left image), the magnitude of the losses of G and D for configuration DC 1.1 are relatively small, this is primarily due to their respective high learning rates, used to address the problem of slow learning and imbalanced update steps. Furthermore, the two-timescale update rule is endorsed [7], such that $lr_d = 2 * lr_g$, allowing different learning rates for optimizing G and D. A larger learning rate means that the discriminator will absorb a larger part of the gradient signal. Hence, a higher learning rate eases the problem of slow learning of the regularized discriminator. This approach makes it possible to use the same rate of updates for G and D, such that a 1:1 update interval per iteration is used. After around 800 epochs, G and D losses begin to diverge, this may at first seem to be promising as the D loss overtakes the G loss, indicating that D is finding it more difficult to distinguish between "real" and "fake" images. However, as illustrated in Figure 15 there are early signs of mode collapse, as generation of 1s and 9s seem more favoured and 2s, 3s and 4s are not generated at all.

The changes made in Configuration DC 1.2 addresses some of the training performance issues encountered when using other configurations. As illustrated in Figure 1 (right image), the respective losses no longer diverge and mode collapse no longer takes place, as can be seen from Figure 17, in Appendix II, where a more diverse range of images are generated at a similar but slightly worse quality to those in Figure 15. This is primarily due to decreasing $lr_g = 0.001$, such that $lr_d = 10 * lr_g$ and updating D twice as often as G at each iteration, resulting in larger losses in terms of magnitude, as well as slower learning. This suggests that training for longer may indeed improve the quality of generated images from a qualitative perspective. The training instability, in terms of the high oscillation and magnitude of the respective losses, when using configuration DC 1.2 with $\beta_1 = 0.9$ is highlighted in Figure 18. This was greatly reduced when the exponential decay rate for the first moment estimates for the Adam optimizers of G and D was decreased from $\beta_1 = 0.9$ as suggested in [10] to $\beta_1 = 0.5$ subsequently recommended in [19] specifically for DCGANs.

DCGAN-v2, a deeper architecture (Figure 19) was trained for 1000 epochs with batch sizes of 128 and a Adam optimizer on different hyperparameter configurations, with a baseline and improved configuration outlined in Table 2.

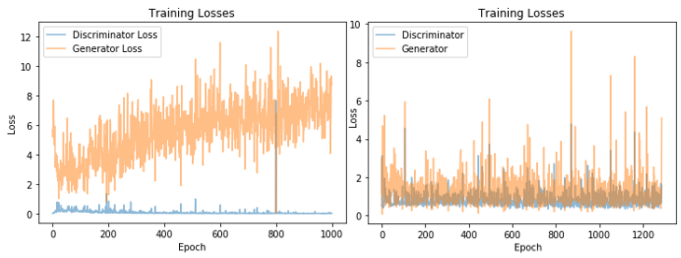

Figure 2. DCGAN-v2 Training Losses for hyperparameter configurations DC 2.1 (left) and DC 2.2 (right)

As can be observed from Figure 2 (left image), the losses of G and D diverge significantly and relatively early when training DCGAN-v2 with configuration DC 2.1. Furthermore, the magnitude of G losses are significantly larger and oscillate more than D losses, where the D loss seems to collapse almost completely, indicating that D is too strong. This highlights that the learning gradient of G is diminishing as D gets too successful up to the point that the G gradient vanishes and learns nothing. Virtual Batch Normalization (VBN) [22], a steeper negative slope of the leaky alpha activation, $\alpha = 0.2$ and a greater learning rate, $lr_g = lr_d = 0.0002$ are endorsed in configuration DC 2.2 to address the vanishing gradient problem.

Batch normalization causes the output of each layer for a given input to be highly dependent on several other inputs in the same minibatch. To avoid this problem, VBN is used such that each example is normalized based on the statistics collected on a reference batch of examples that are chosen once and fixed at the start of training, and on the example itself. The reference batch is normalized using only its own statistics. VBN is computationally expensive because it requires running forward propagation on two minibatches of data, so it is used only in G. We set $lr_d = lr_g = 0.0002$ with both G and D being updated only once per iteration in an attempt to speed up learning, but also try to weaken D with respect to G. A larger $\alpha = 0.2$ is used, to allow very small negative gradients close to zero to be greater during back-propagation to aid more effective learning. Figure 2 (right image) shows that these changes have addressed the vanishing gradient issue in this case, however the losses are very volatile and decreasing to $\beta_1 = 0.5$ did not help reduce this volatility. The output images of DCGAN-v2 for DC 2.2 (Figure 23) seem only slightly more coherent than for DC 2.1 (Figure 21), but are both visually clearer than the output images of the shallower DCGAN-v1 (Figures 17, 15).

3. Conditional GAN (CGAN)

We aim to improve the quality of generated images by adopting a supervised approach and using X_{train} to train shallow fully connected (CGAN-v1) and deep convolutional (CGAN-v2) implementations of a Conditional Generative Adversarial Network (CGAN) [16]. In a DCGAN there is no control on the exact modes of data being generated. However, by conditioning a GAN with auxiliary information and endorsing a modified loss function (Equation 4) it is possible to direct the data generation process and directly combat the issue of mode collapse, as the output is instructed and incentivised to generate images of a given desired label. We utilize CGANs to generate MNIST digits conditioned on class labels that are encoded as 10 dimensional one-hot vectors, denoted as y. The Generators of CGAN-v1 and CGAN-v2 both take into input a 100 dimensional latent vector z drawn from a uniform distribution, along with y, and output a 28x28x1 image.

The architecture of CGAN-v1 is summarised in Figure 24. Both G and D are shallow fully connected networks. It is important to note that D recieves an input of size $784 + y = 794$, due to the label conditioning. CGAN-v1 was trained for 1000 epochs with batch sizes of 64 and an Adam optimizer on different hyperparameter configurations, with a baseline and improved con-

figuration outlined in Table 3. As can be observed from Figure 3 (left image) the losses when using configuration CC 1.1 are large and oscillate in a volatile fashion. Furthermore, the losses converge, but diverge again as time progresses with D loss being initially higher than G loss. This suggests that D is initially quite weak and does not penalize G enough during the early stages of training, such that in later stages as D gets stronger, G learns poorly and the quality of generated images do not improved significantly. This is illustrated in Figure 25, where after 200 epochs the quality of generated images do not seem to improve.

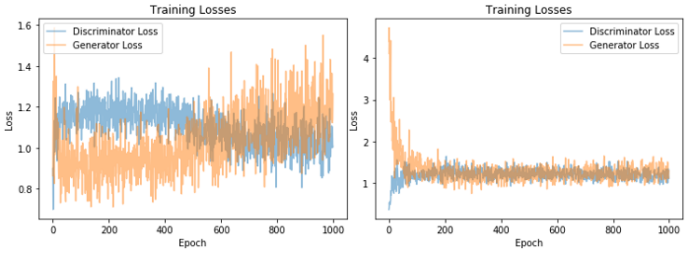

Figure 3. CGAN-v1 Training Losses for hyperparameter configurations CC 1.1 (left) and CC 1.2 (right)

The adjustments made in configuration CC 1.2 aim to address some of training issues encountered when using other configurations. As can be observed from Figure 3 (right image), the magnitude and volatility of the respective losses have been reduced and converge much faster. This is most likely due to doubling the initial learning rate to $lr_g = lr_d = 0.002$ as well as increasing to $\beta_1 = 0.9$ in an attempt to speed up learning and reduce oscillation. The D loss also starts below the G loss and doesn't overtake and diverge from it during training. This is probably due to the fact that D was updated twice as often as G per iteration in order to strengthen D. Furthermore, one sided label smoothing was adopted to ensure that D doesn't get too strong, as it prevents D from giving very large gradient signals to G, facilitating a more balanced learning approach. CGAN-v1's seemingly rudimentary structure was initially used as it is very quick to train, but surprisingly generates images (Figure 28) visually comparable to that of both DCGANs, demonstrating the power of class conditioning in image synthesis.

The architecture of CGAN-v2 is summarised in Figure 29. The main difference being that this structure behaves essentially as a class conditioned DCGAN, where it is important to note that the Discriminator input has dimensions 28x28x1 + 28x28x10 = 28x28x11, to ensure correct concatenation of the generated image and the corresponding label encoding. CGAN-v2 was trained for 1000 epochs with batch sizes of 64 and an Adam optimizer on different hyperparameter configurations, with a baseline and improved configuration outlined in Table 4. As can be observed from Figure 4 (left image), when training using configuration CC 2.1, G losses diverge from D losses after 200 epochs and G seems to be confronting a diminishing gradient as the magnitude of D loss slowly dwindles towards 0. A vanishing gradient seems to be a common challenge when training deep convolutional architectures, most likely due to having D too strong, G not learning enough in early training stages, and issues with Batch Normalization as previously discussed in Section 2.

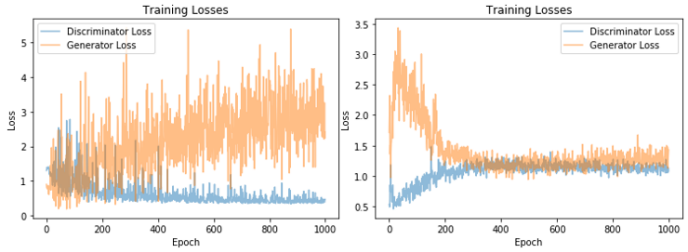

Figure 4. CGAN-v2 Training Losses for hyperparameter configurations CC 2.1 (left) and CC 2.2 (right)

In configuration CC 2.2, VBN was applied to G to address the problem of Internal Covariate Shift (ICS) [21], as the constant shifting of input distributions in subsequent layers makes it difficult for the optimization to converge. In [22] it was argued that empirically BN does not reduce ICS, but it does improve the "Lipschitzness" of the gradient, broadening the range of learning rates and generally stabilizing learning. Thus, a higher learning rate $lr = 0.002$ was used to speed up learning, where D and G were each updated once per iteration, in order to weaken D with respect to G. As can be seen in Figure 4 (right image), the implemented changes address the vanishing gradient issue and improve training stability as the losses converge, with G loss decreasing and D loss increasing, indicating that effective learning is taking place. The quality of generated images Figure 33 are significantly more coherent for a range of styles for all class labels compared to all previous approaches. In fact, the quality of generated images do not change significantly after 300 epochs as can be visually verified by Figure 32, since the losses converge and stabilize at around this point (Figure 4).

4. Inception Score

In the previous section, subjective qualitative analysis of generated images with respect to the CGAN architecture and the corresponding training processes were discussed. In this section, a well known quantitative measure known as the Inception Score (IS) [1, 2] is utilized to assess the quality and diversity of images generated by CGAN-v1 and CGAN-v2. Equation 2 highlights how IS is calculated, where $\mathbf{x} \sim p_g$, indicates that $\mathbf{x}$ is sampled from the distribution of generated images p_g. $D_{KL}(p(y|\mathbf{x})\|p(y))$ is the KL-divergence [24] between the conditional class distribution $p(y|\mathbf{x})$, and the marginal class distribution $p(y) = \int_{\mathbf{x}} p(y|\mathbf{x})p_g(\mathbf{x})$, of the generated images.

$$\mathrm{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_g} D_{KL}(p(y|\mathbf{x})\|p(y))\right) \tag{2}$$

Desirable qualities of generated images are encoded into Equation 2. Since $\ln(\mathrm{IS}(G)) = I(y;\mathbf{x}) = H(y) - H(y|\mathbf{x})$, which shows that the expected KL-divergence between $p(y|\mathbf{x})$ and $p(y)$ is equal to their Mutual Information $I(y;\mathbf{x})$, which is related to their respective entropies $H(y)$ and $H(y|\mathbf{x})$. Thus, in order to have a high IS score, the entropy of $p(y)$ should be high and the entropy of $p(y|\mathbf{x})$ should be low. A perfect IS score would be achieved on the ideal distributions shown in Figure 34. For $p(y|\mathbf{x})$ to have a low entropy, generated images should be sharp and contain objects that can be clearly distinguished. In order for $p(y)$ to have high entropy, p_g should be representative of a high diversity of images from all the classes.

In order to report on the Inception Scores, the popular LeNet-5 classifer [12] was pre-trained on X_{train} for 100 epochs. The LeNet-5 architecture (Figure 35) consists of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully-connected layers and finally a softmax classifier. The classification accuracy on X_{test} was high, as expected, at 97.5%, demonstrating that the classifier is quite good at recognizing images belonging to the MNIST dataset distribution.

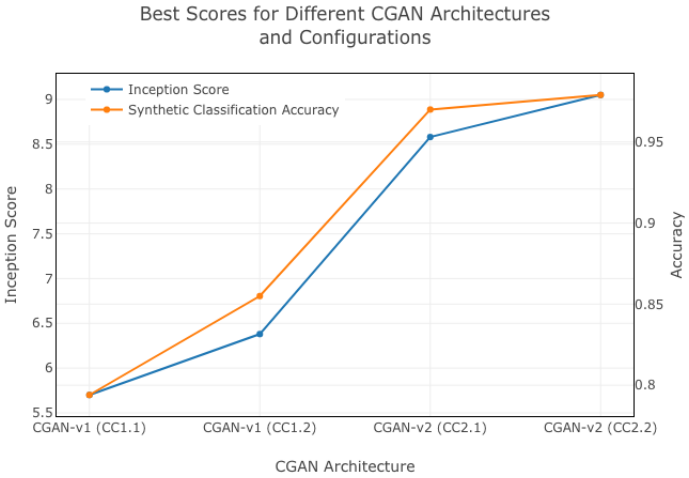

Figure 5. Inception Scores for Architectures and Configurations discussed in Section 3.

As can be observed from Figure 5, noticeable improvements in IS occur when CGAN-v1 uses CC 1.2 instead of CC 1.1 and when CGAN-v2 uses CC 2.2 instead of CC 2.1. Thus, focusing on the stability of training by tuning hyperparameters and procedures can produce tangible and measurable improvements in image generation quality. CC 1.2 gave improvements through increasing lr_g, lr_d and β_1, as well increasing the strength of D by making the G:D update ratio 1:2 per iteration. One-sided label smoothing was also adopted to ensure D doesn't get too strong and to balance training. The largest improvement in IS occurs by changing from CGAN-v1 to CGAN-v2, since you move from a shallow fully connected structure to a deep convolutional structure capable of learning in much higher dimensions. CC 2.2 enabled smaller improvements in IS, through the greater effect that VBN has over BN on reducing ICS, along with increasing the learning rate.

5. Re-trained Handwritten Digit Classifier

In the following section, various models are trained using random subsets of X_{train} and images generated by CGAN-v2 denoted as X_{synth} with cardinality 10,000. X_{train} is split into X'_{train} and X_{val}, with cardinalities of 55,000 and 5000 respectively for training and evaluation of models.

As seen in Section 4, the LeNet-5 classifier can achieve strong classification accuracies of 97.5% on X_{test}. This score was achieved by using a large number of training points, which in a real scenario can be costly to label and train on. There is therefore motivation to find ways of combining the easily obtained CGAN data with the expensive real data in a way that requires less real data but still achieves a high performance.

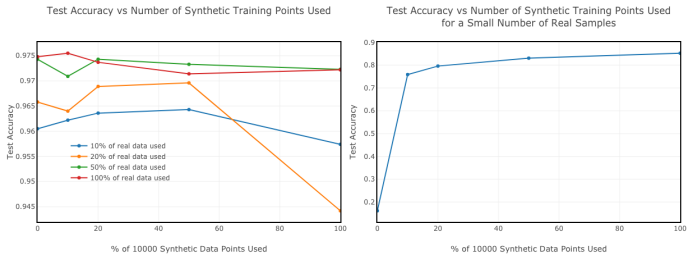

Figure 6. Test Scores using Varying Portions of the Real and Synthetic Sets.

In the initial approach, the LeNet-5 architecture is trained by randomly mixing different portions of X'_{train} and X_{synth}. The effect of varying the portions of the training sets used can be seen in Figure 6 (left Image). Models trained on more real data generally outperform models that see fewer examples. Models trained on more synthetic data get a better performance when trained on $2,000 - 5,000$ synthetic data points. Figure 6 (right image) shows the accuracy of the model trained using only 1% of X'_{train}, achieves significantly worse performance than any model trained with a larger subset of X'_{train}, even with the addition of many synthetic samples. Therefore a different training strategy is required in order to improve the test performance.

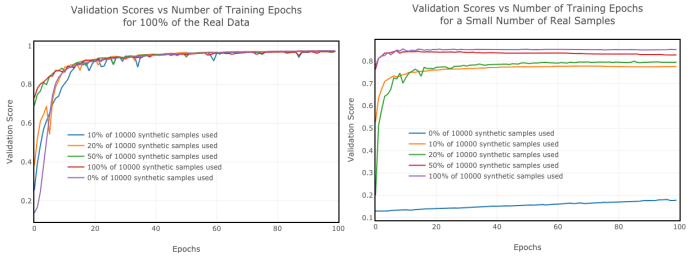

Figure 7. Validation Scores at Each Epoch When Training on 100% (left) and 1% (right) of X_{train}

Looking at the validation score curves on the left in Figure 7, it can be seen that models trained with a larger portion of X_{synth} perform better on X_{val} during training in earlier epochs. This is also true for Figure 7 (right) where 1% of X'_{train} is used. The tradeoff is that using more synthetic data causes random drops in the validation score between epochs. Thus, it can be expected that training the classifier with 20% of X_{synth} for approximately 20 epochs and then switching to the small set of X'_{train} for the remaining 80 epochs will lead to faster training in the first training phase, and more stable performance in the second phase.

Figure 8. Validation and Test Scores Using the New Training Strategy

Figure 8 shows that when the hypothesized strategy is used, a performance of 97.4% (similar to the original classifier) can be

achieved. The validation curve in Figure 8 shows significant improvements during the first training phase when a large portion of X_{synth} is used. After 20 epochs, when there is little improvement in the validation scores, the training is performed on the 1% subset of X'_{train} for fine-tuning (hence the temporary drop in validation score in Figure 8). From both graphs it can also be seen that the best performance occurs when the size of X_{synth} is 2000. Although the test accuracy was not improved upon, it has been shown that there are alternative training methods to achieve a good performance for scenarios where labelled data is scarce.

6. Bonus: t-SNE & PCA

The LeNet-5 classifier pre-trained on X_{train} is taken and used to extract the penultimate layer's activations (embeddings) of 500 randomly sampled real images from X_{test}, which are denoted as S_{real}, and 500 randomly sampled synthetic images, denoted as S_{fake} which were generated from CGAN-v2 (CC 2.2). Both S_{real} and S_{fake} contain 50 images from each class[5]. The quality and diversity of S_{real} is assessed against S_{fake}, by projecting the extracted embeddings to a 2D sub-space using embedding methods such as t-SNE (Figures 9 & 10) and PCA[6] (Figures 36 & 37).

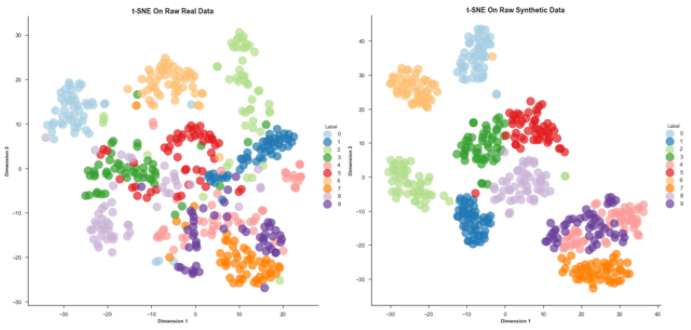

Figure 9. t-SNE projection of S_{real} (left) and S_{fake} (right) without LeNet-5 feature extraction

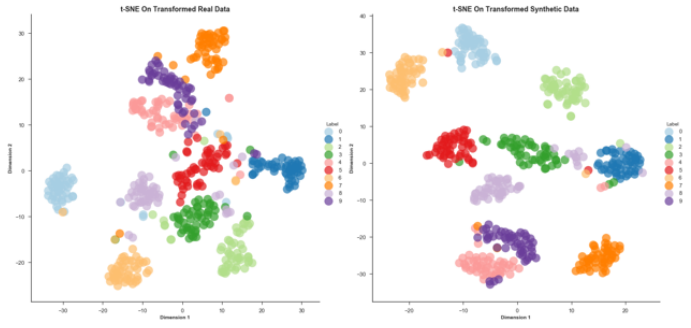

Figure 10. t-SNE projection of S_{real} (left) and S_{fake} (right) with LeNet-5 feature extraction

As can be seen from Figure 9, the between-class separation is greater, the within class separation is smaller and there are not as many outliers in S_{fake} compared to S_{real}, which illustrates that CGAN-v2 (CC 2.2) generates images that do not vary much within a class, but are quite distinct between classes. As can be seen from Figure 10 the projected embeddings are grouped much closer together, especially for S_{real}. It is important to note that both graphs show the same overlaps in clusters, especially with the pink (label 4) and purple clusters (label 9), due to the similarity of their handwritten shapes. The observation of similar cluster overlaps supports the claim that CGAN-v2 (CC 2.2) models the X_{train} distribution well. Furthermore, these similarities provide further insight as to why the classification accuracy on X_{test} was quite high at 75% even when trained on a small 10% of synthetic images (1000 images) from CGAN-v2 (CC 2.2) and a very small number of real samples (1% of X_{train} = 550 images).

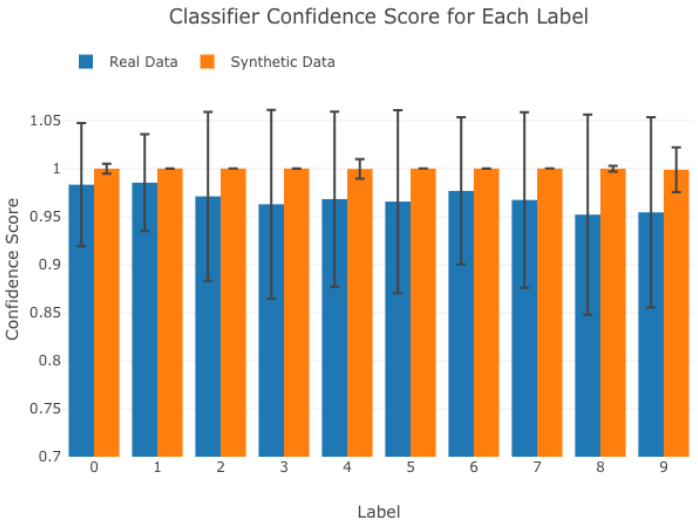

Figure 11. Classification confidence scores for each label of S_{real} (Blue Bar) and S_{fake} (Orange Bar) with Standard Deviation range

Figure 11 further shows that the classes of S_{fake} are easily separable as the mean confidence scores are higher, and the standard deviations are much smaller for S_{fake} compared to S_{real}. It is important to note, that labels 4 and 9 in both cases show the highest standard deviations, supporting the observations made regarding Figure 10.

7. Comparisons, Conclusions & Future Work

From the explorations and experimentations presented in this report it was evident that DCGANs, especially deep structures, require great effort to train in a stable manner and the quality of generated images at best just fall short of being aesthetically comparable to X_{train} as a whole. A significant advancement in the quality of generated images came about with the implementation of CGAN-v2, most likely due to its inherent deep convolutional structure and the incorporation of supervised class conditioning to aid the generation of all desired modes. Further improvements were enabled with the adoption of techniques such as VBN and one-sided label smoothing to aid stable and effective learning. Achieving a relatively high IS on CGAN-v2 (CC 2.2) corroborated visual assessments of its superior image generation. However, due to the limitations of IS [2], other measures such as the Fréchet Inception Distance (FID) [7] can be endorsed to try to further verify our assessments. Other architectures such as InfoGAN [3] can be considered to try to improve image generation quality.

[5] 50 instead of 10 images per class were taken as the larger sample sizes include more outliers and are more representative of their respective distributions.

[6] PCA did not provide useful visualisations or insights into how the classes are distributed, so it is not discussed in the main body of the report (Apendix VII).

References

[1] D. Bang and H. Shim. Improved training of generative adversarial networks using representative features. *CoRR*, abs/1801.09195, 2018.

[2] S. Barratt and R. Sharma. A Note on the Inception Score. *arXiv e-prints*, page arXiv:1801.01973, Jan 2018.

[3] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016.

[4] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. Emnist: an extension of mnist to handwritten letters. *CoRR*, abs/1702.05373, 2017.

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1406.2661, Jun 2014.

[6] P. J. Grother. Nist handprinted forms and characters, nist special database 19, 1995.

[7] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[9] Y. Karayaneva and D. Hintea. Object recognition in python and mnist dataset modification and recognition with five machine learning classifiers. *Journal of Image and Graphics*, 6:10–20, 01 2018.

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

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[13] A. L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[14] I. Mcmanus, K. Stöver, and D. Kim. Arnheim's gestalt theory of visual balance: Examining the compositional structure of art photographs and abstract images. *i-Perception*, 2:615–47, 10 2011.

[15] L. M. Mescheder. On the convergence properties of GAN training. *CoRR*, abs/1801.04406, 2018.

[16] M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. *arXiv e-prints*, page arXiv:1411.1784, Nov 2014.

[17] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. volume 27, pages 807–814, 06 2010.

[18] H. N. Pathak, X. Li, S. Minaee, and B. Cowan. Efficient super resolution for large-scale images using attentional GAN. *CoRR*, abs/1812.04821, 2018.

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

[20] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

[21] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[22] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How Does Batch Normalization Help Optimization? *arXiv e-prints*, page arXiv:1805.11604, May 2018.

[23] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[24] M. Vidyasagar. Kullback-leibler divergence rate between probability distributions on sets of different cardinalities. In *49th IEEE Conference on Decision and Control (CDC)*, pages 948–953, Dec 2010.

[25] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.

Appendix I - Formulations & Techniques

The MNIST dataset was constructed from a subset of NIST's Special Database 1 (SD-1) and Special Database 3 (SD-3) which contain binary images of handwritten digits [6, 4]. SD-3 samples were collected from Census Bureau employees and SD-1 samples were collected from high-school students, thus samples from SD-3 are generally cleaner and easier to recognize than samples from SD-1. X_{train} is composed of 30,000 patterns from SD-1 and 30,000 patterns from SD-3 from approximately 250 writers. X_{test} is composed of 5,000 patterns from SD-1 and 5,000 patterns from SD-3 approximately from 250 writers. It is important to note that the sets of writers pertaining to X_{train} and X_{test} respectively, are mutually exclusive

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 12. Batch Normalization Algorithm [8]

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{3}$$

$$\min_G \max_D V(D,G) = \mathop{\mathbb{E}}_{x\sim p_{\text{data}}(x)}[\log D(x,y)] + \mathop{\mathbb{E}}_{z\sim p_z(z)}[\log(1 - D(G(z,y),y))] \tag{4}$$

Appendix II - DCGAN-v1

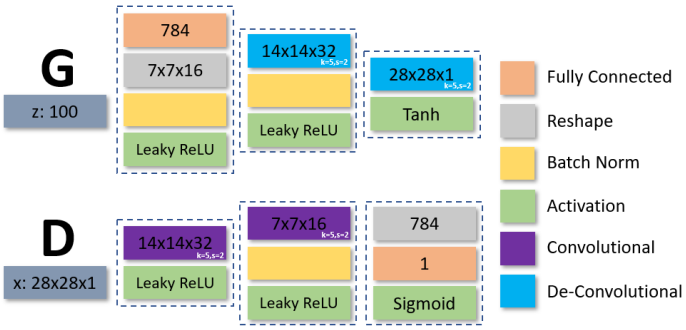

Figure 13. DCGAN-v1 Architecture Overview

Configuration	DC 1.1	DC 1.2
Epochs	1000	1000
Batch Size	128	128
lr_g	0.0001	0.0005
lr_d	0.001	0.001
β_1	0.9	0.5
α	0.01	0.2
Update Ratio (G:D)	1:1	1:1

Table 1. DCGAN-v1 Hyperparameter Configurations

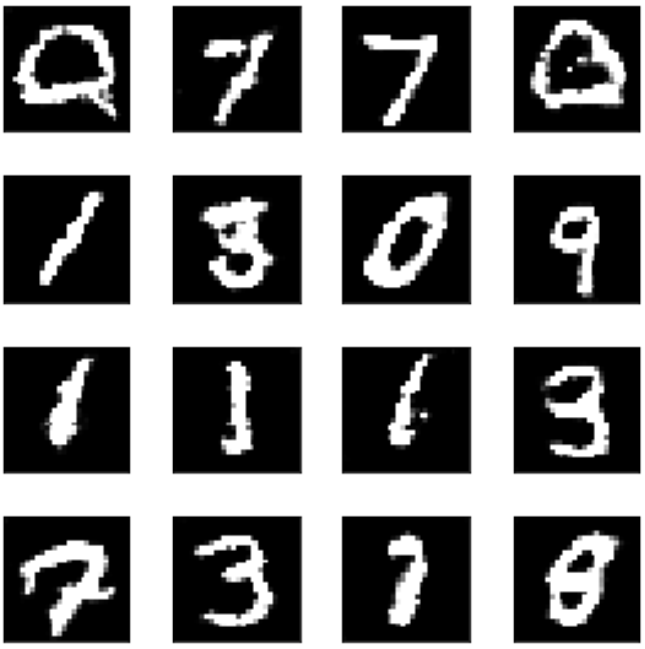

Figure 14. Generated images after 10 epochs using DC 1.1

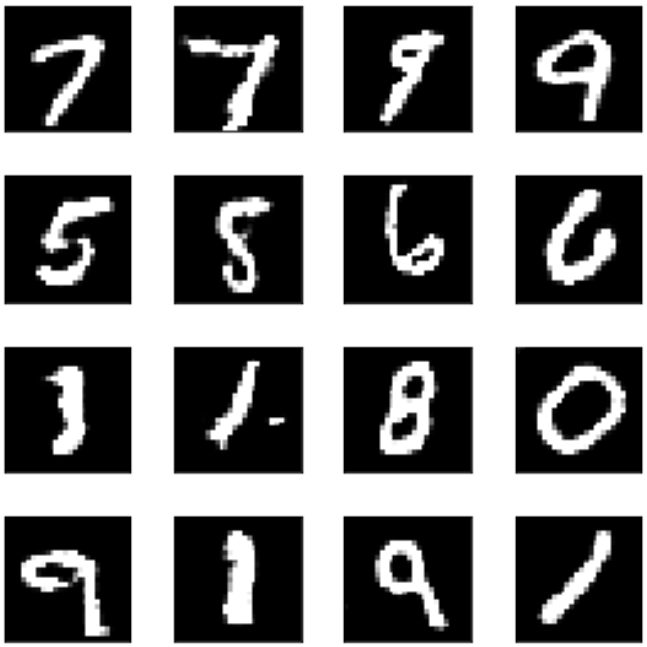

Figure 15. Generated images after 1000 epochs using DC 1.1

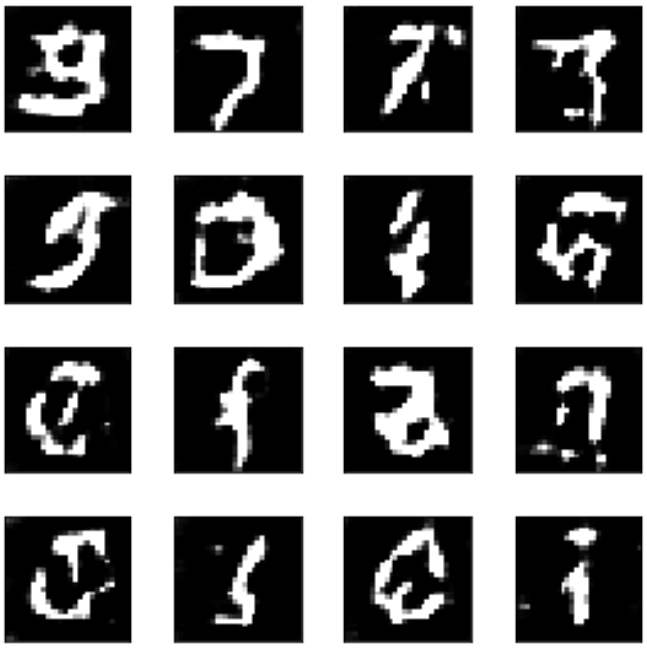

Figure 16. Generated images after 10 epochs using DC 1.2

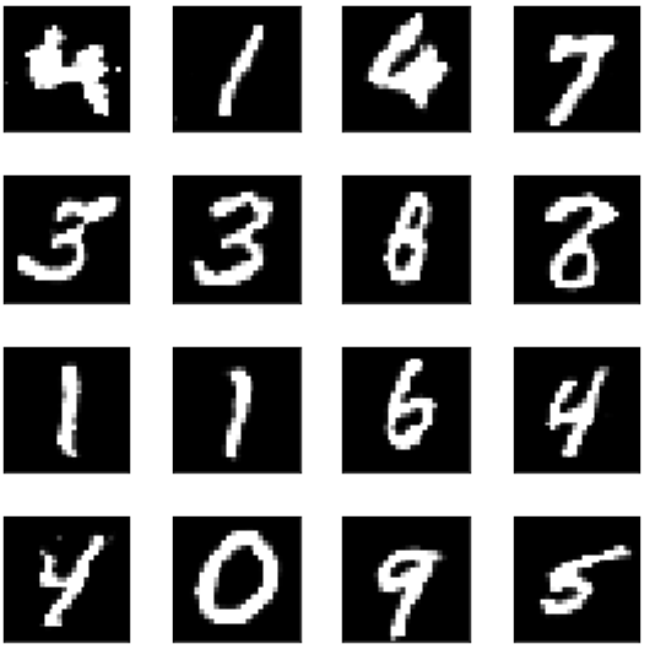

Figure 17. Generated images after 1000 epochs using DC 1.2

Configuration	DC 2.1	DC 2.2
Epochs	1000	1000
Batch Size	128	128
lr_g	0.0001	0.0002
lr_d	0.0001	0.0002
β_1	0.9	0.5
α	0.1	0.2
Update Ratio (G:D)	1:2	1:1
Batch Normalization (BN)	BN	VBN

Table 2. DCGAN-v2 Hyperparameter Configurations

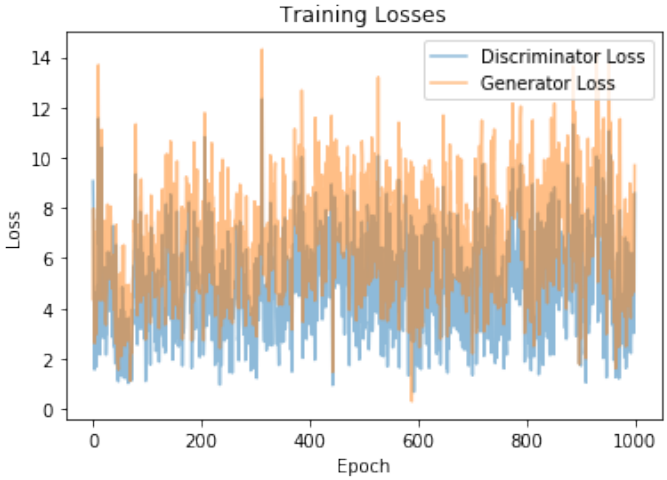

Figure 18. Training Losses using DC 1.2 with $\beta_1 = 0.9$

Appendix III - DCGAN-v2

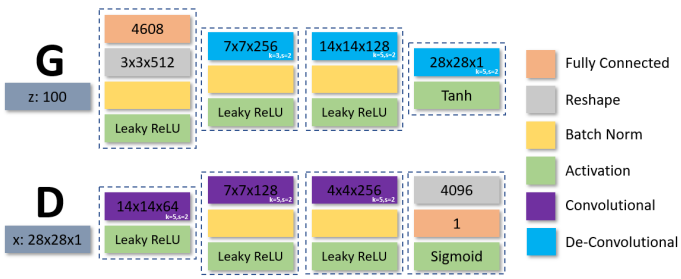

Figure 19. DCGAN-v2 Architecture Overview

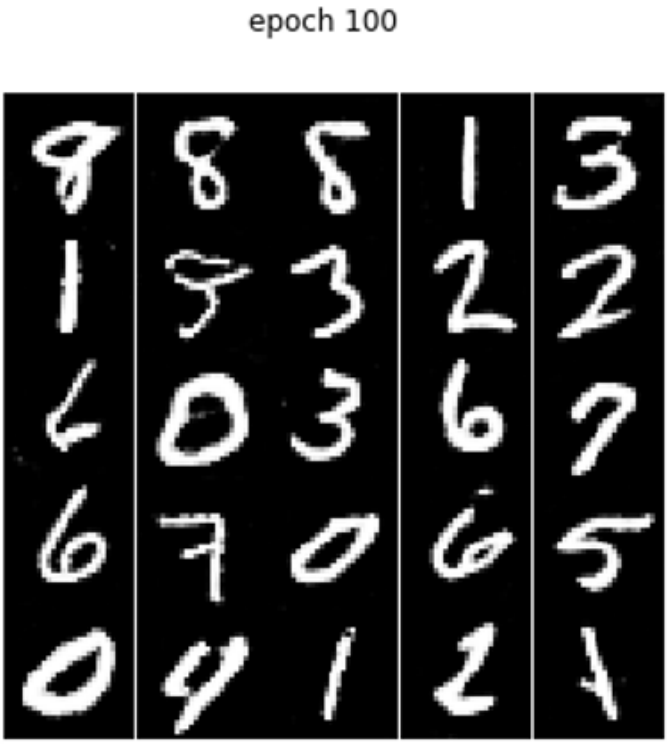

Figure 20. Generated images after 100 epochs using DC 2.1

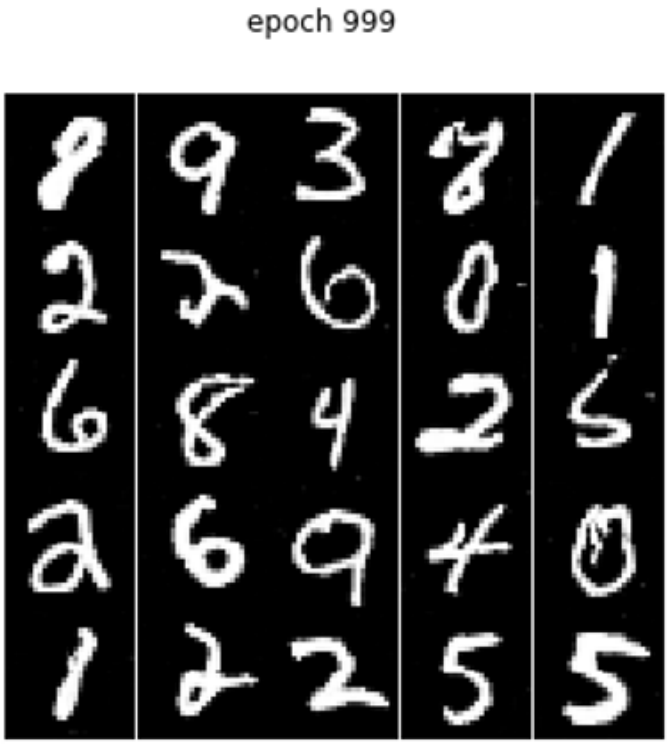

Figure 21. Generated images after 999 epochs using DC 2.1

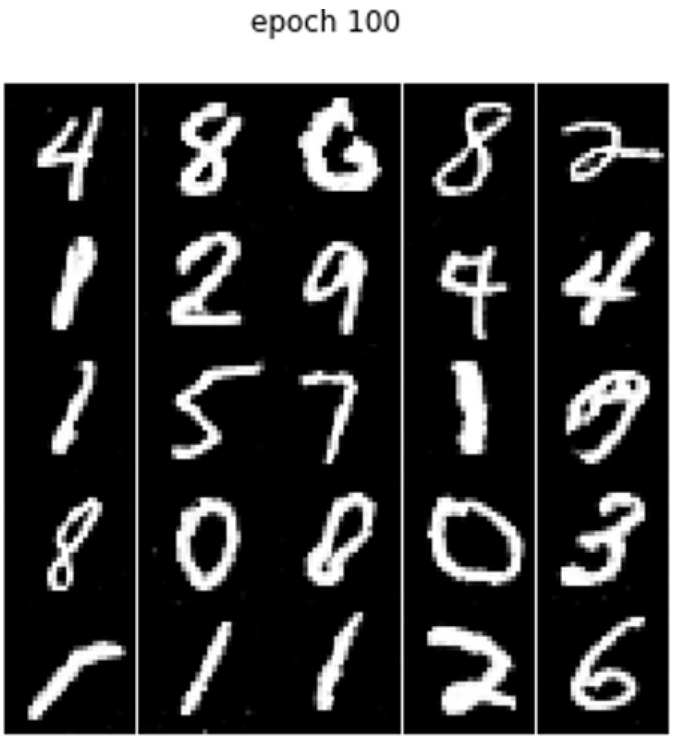

Figure 22. Generated images after 100 epochs using DC 2.2

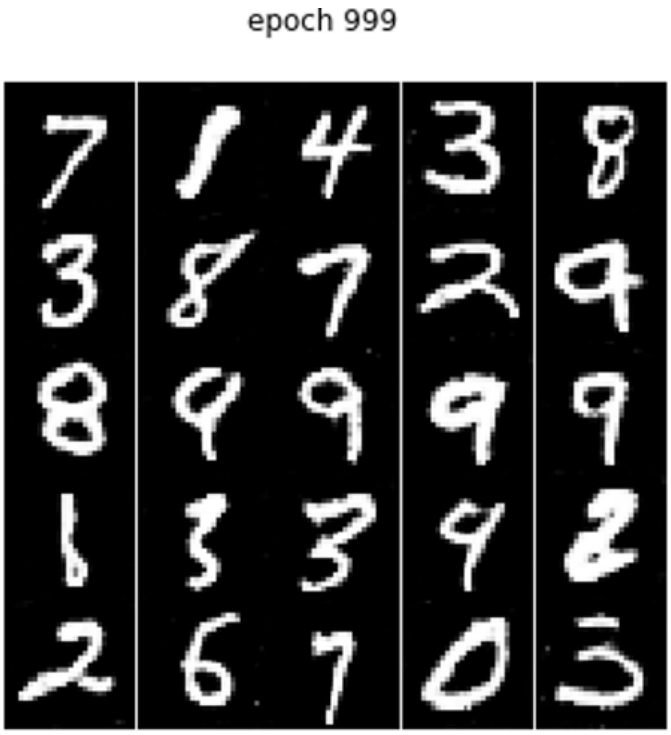

Figure 23. Generated images after 999 epochs using DC 2.2

Appendix IV - CGAN-v1

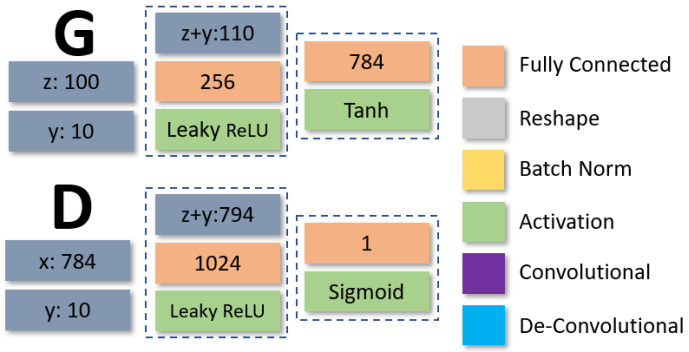

Figure 24. CGAN-v1 Architecture Overview

Configuration	CC 1.1	CC 1.2
Epochs	1000	1000
Batch Size	64	64
lr_g	0.001	0.002
lr_d	0.001	0.002
β_1	0.5	0.9
α	0.2	0.2
Update Ratio (G:D)	1:1	1:2
Batch Normalization (BN)	NO	NO
One-Sided Label Smoothing	NO	YES

Table 3. CGAN-v1 Hyperparameter Configurations

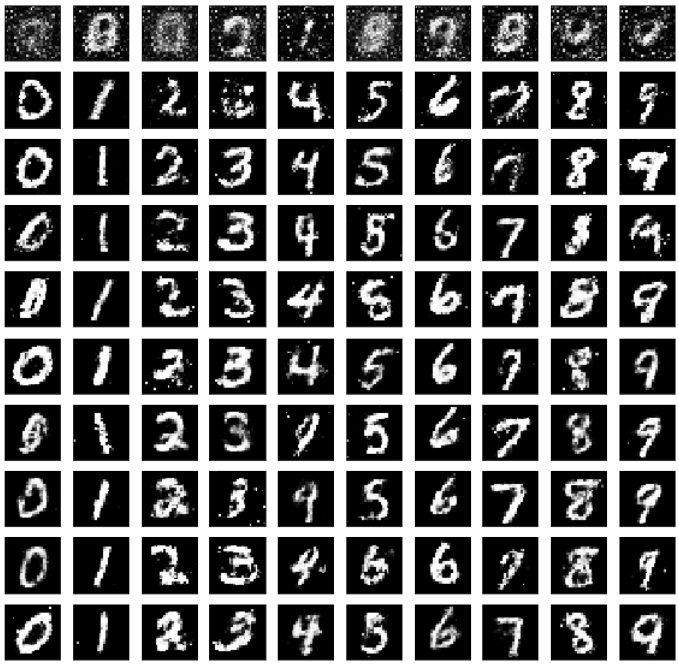

Figure 25. Generated images per 100 epochs in row descending time order starting from top row = 100 epochs when using CC 1.1

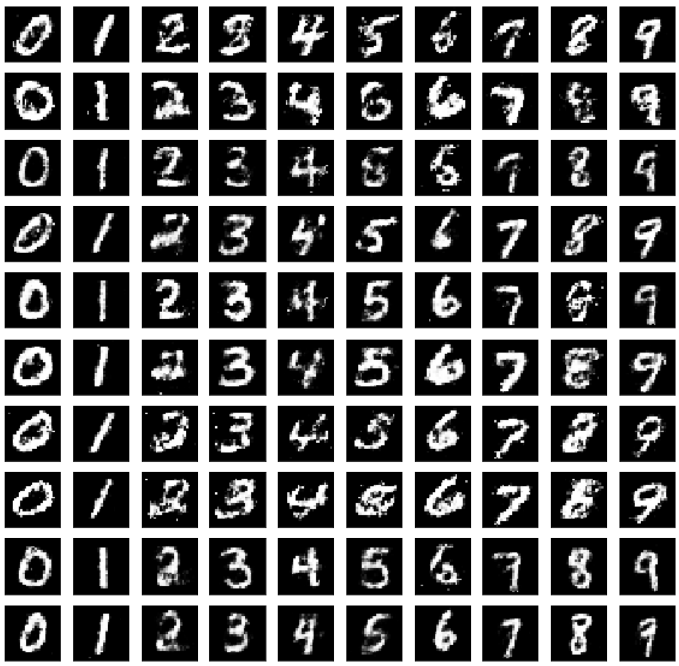

Figure 26. Generated images after 1000 epochs when using CC 1.1 where column indices are class labels and row indices are the handwritten styles

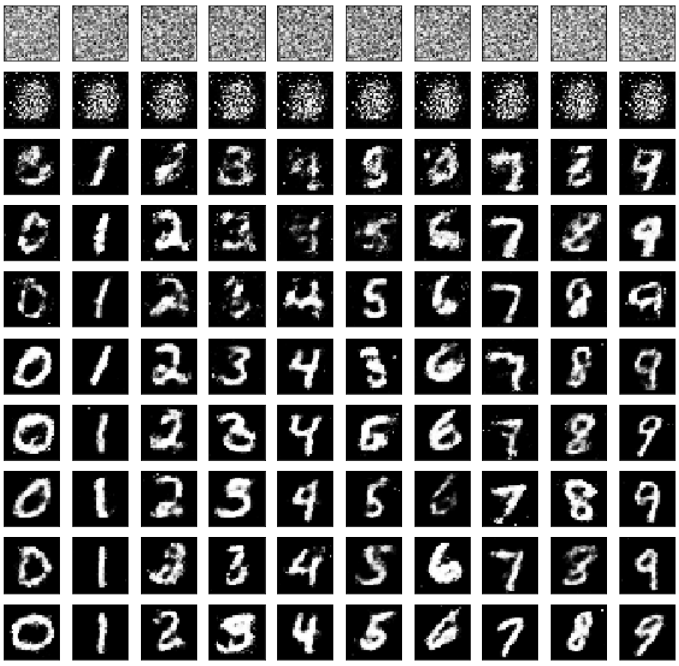

Figure 27. Generated images per 100 epochs in row descending time order starting from top row = 100 epochs when using CC 1.2

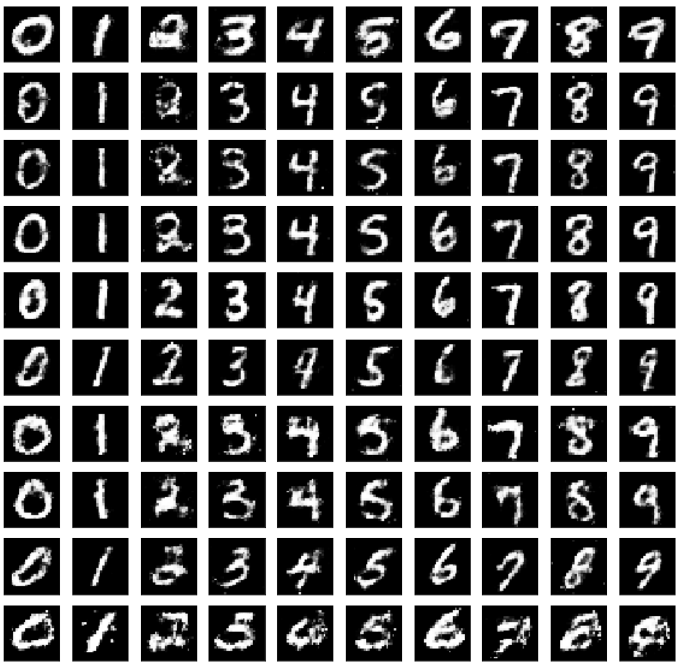

Figure 28. Generated images after 1000 epochs when using CC 1.2 where column indices are class labels and row indices are the handwritten styles

Appendix V - CGAN-v2

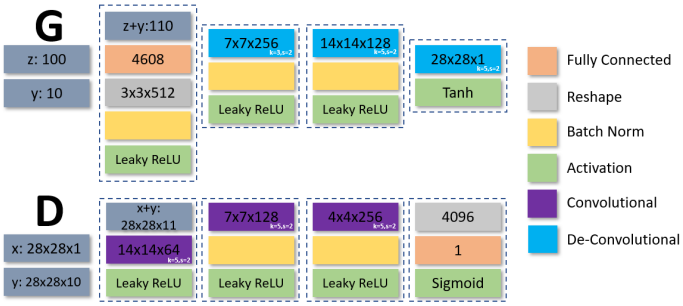

Figure 29. CGAN-v2 Architecture Overview

Configuration	CC 2.1	CC 2.2
Epochs	1000	1000
Batch Size	64	64
lr_g	0.001	0.002
lr_d	0.001	0.002
β_1	0.5	0.9
α	0.2	0.2
Update Ratio (G:D)	1:2	1:1
Batch Normalization (BN)	BN	VBN
One-Sided Label Smoothing	YES	YES

Table 4. CGAN-v2 Hyperparameter Configurations

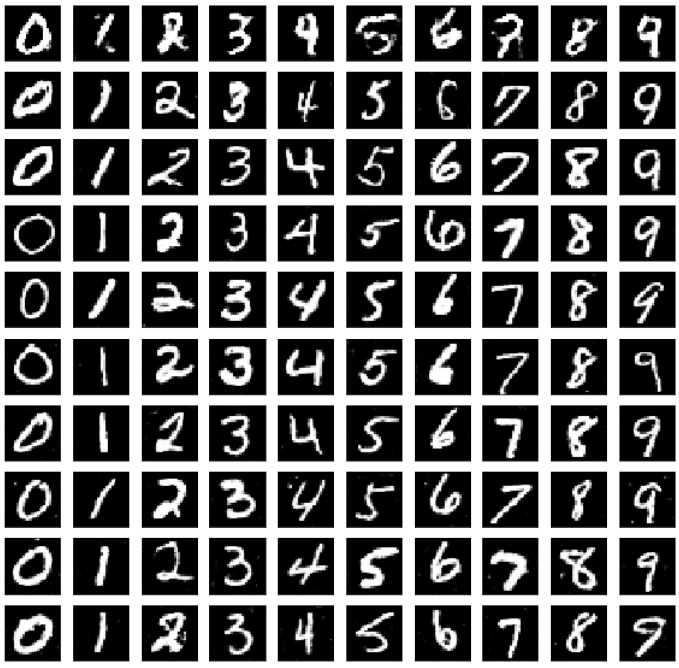

Figure 30. Generated images per 100 epochs in row descending time order starting from top row = 100 epochs when using CC 2.1

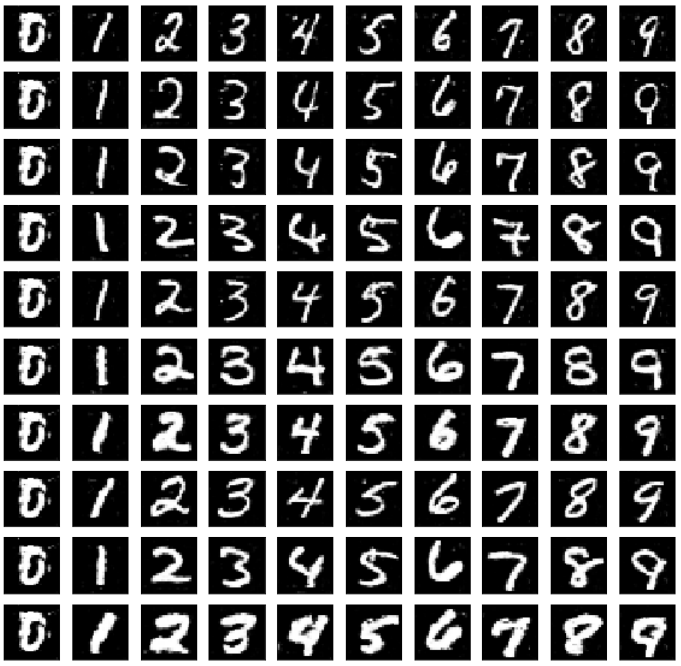

Figure 31. Generated images after 1000 epochs when using CC 2.1 where column indices are class labels and row indices are the handwritten styles

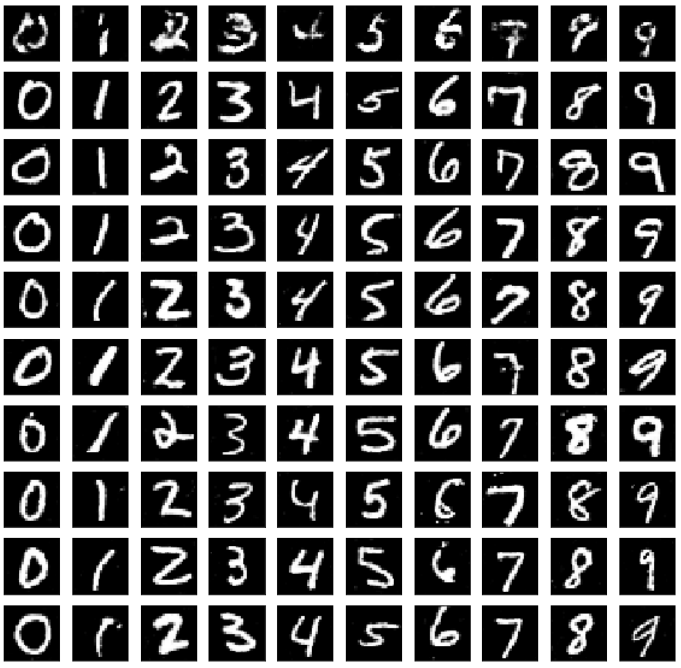

Figure 32. Generated images per 100 epochs in row descending time order starting from top row = 100 epochs when using CC 2.2

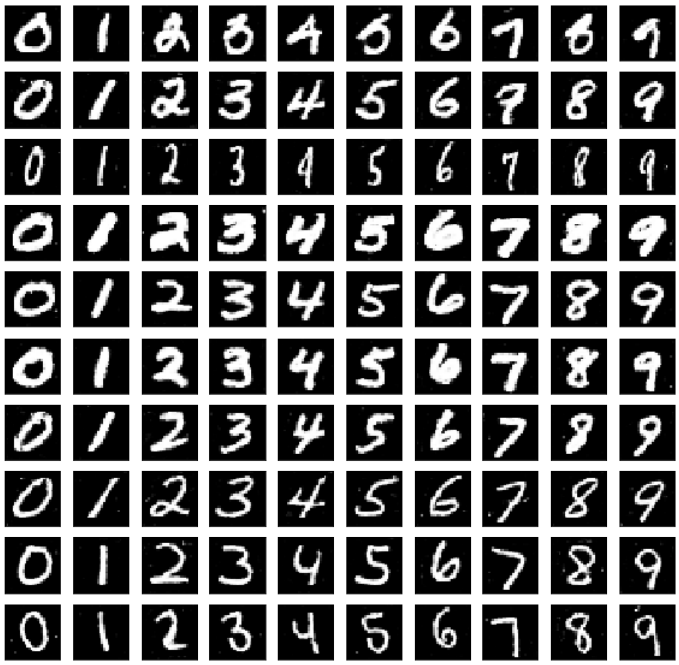

Figure 33. Generated images after 1000 epochs when using CC 2.2 where column indices are class labels and row indices are the handwritten styles

Appendix VI - Inception Score

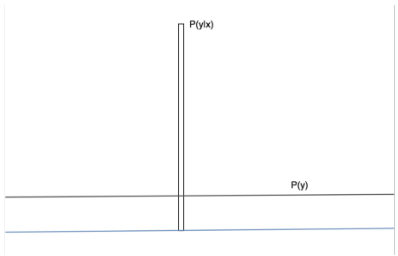

Figure 34. Ideal Distributions of $p(y|\mathbf{x})$ and $p(y)$ that achieve a perfect Inception Score

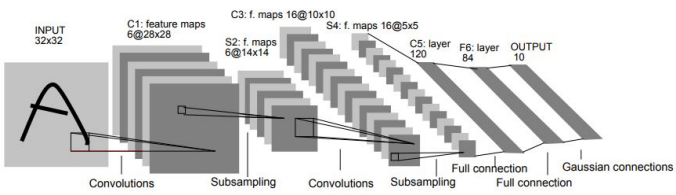

Figure 35. Architecture Overview of LeNet-5 [12]

Appendix VII - Bonus: t-SNE & PCA

Figure 36. PCA projection of S_{real} (left) and S_{fake} (right) without LeNet-5 feature extraction

Figure 37. PCA projection of S_{real} (left) and S_{fake} (right) with LeNet-5 feature extraction