CS492(A) Machine Learning for Computer Vision

Anar Rzayev
KAIST
rzayev.anar1@kaist.ac.kr

Berhane Weldegebriel
KAIST
breteklu@kaist.ac.kr

1. Introduction

This report will be focused on training and testing of Generative Adversarial Networks (GANs) to generate synthetic handwritten digit images from MNIST dataset. Their corresponding evaluation performance has been obtained visually through inspection and an inception score that was calculated using a LeNet convolutional neural network.

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

2. Deep-Convolutional Generative Adversarial Network

Generative Adversarial Networks are an example of generative models. In fact, 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 Generative Adversarial Network (DCGAN), similarly to Class-Conditional Generative Adversarial Network (CCGAN), does it through the use of convolution and deconvolution layers.

2.1. Architecture

The architecture design of the networks consists of 2 separate pieces: a *Generator* and a *Discriminator*. The main goal of the generator is to generate new examples from random noise and the functionality of the discriminator is to distinguish these generated examples from real ones.

2.1.1 Generator

Our network architecture is based on the original DCGAN, as seen in the following Figure 1. Initially, the 1D noise input with length 100 is processed through a fully connected (Dense) layer. Subsequently, it is propagated forward by Batch Normalization (BN), followed by non-linear Rectified Linear Unit (ReLU) activation. The major objective of applying Batch Normalisation after each Dense/Convolution layer is to provide regularization for avoiding overfitting and manage smooth, normalized signal propagation. Moreover, non-linear activation ReLU is mainly implemented for resolving saturation issues and consequently, vanishing gradients. Afterwards, Batch Normalization and ReLU will be followed by a Dense layer and BN $\&$ ReLU, in which reshaping function occurs to output in 3 dimensions. Then, upsampling layer will be utilized, where it operates non-linear deconvolution and therefore, will generate new data which is later refined by convolution. Whenever BN and ReLU will follow the later resultant stage, features will be upsampled once again to the original dimensions, and subsequently, convolution will follow the stage.

2.1.2 Discriminator

As discussed earlier, generator merely creates random noise in which proper there needs to be proper guidance onto which images it should produce. In fact, discriminator learns that guidance through the learning procedure of decision boundary between inputted images being real or fake.

The discriminator starts with convolution to extract meaningful features from original picture, following the Batch Normalization and a non-linear activation Exponential Linear Unit to ensure it does not saturate the output at zero compared to ReLU, making training more efficient. Subsequently, features will be refined monce again using convolution and later, weights will be regularized through Batch Normalization and outputted through Exponential Linear Unit into pooling layer, in which most important features are chosen in terms of magnitude. In the end, discriminator flattens the features that were refined by Dense layers into probability measures. The last phase will output desired result through the sigmoid activation function to classify the input as real or fake.

Overall, such architecture will examine the capabilities (number of channels) of both networks by hyper-tuning parameter F. Modifying the network capabilities will enable for generator discriminator to extract generate more distinctive features, resulting in more clear image and higher prediction accuracy. Another consideration of hyper-

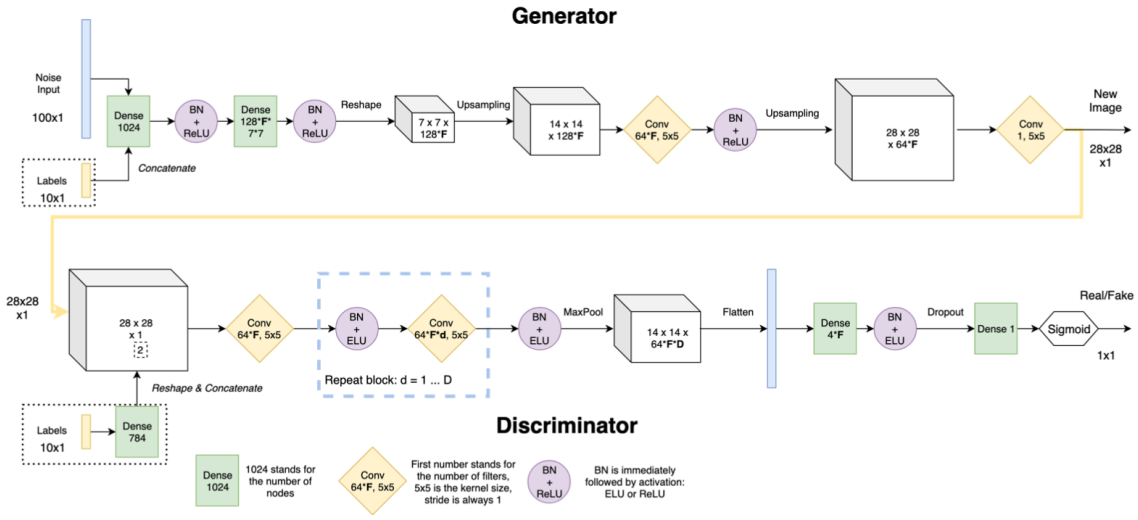

Figure 1. Proposed Generative Adversarial Network architecture

parameter could be making the discriminator deeper so that balancing of the discriminator and generator may occur and be measured. Indeed, we suggest to include more convolutions by changing D so that there could be some influence of empowering the discriminator. Figure 1 will depict those changes carefully.

2.2. Training

2.2.1 Loss-Function

In general, training procedure for this joint architecture will be achieved by alternating gradient descent (GD) on the discriminator network, while the generator's weights are frozen, and applying such approach vice-versa. Overall, General Adversarial Network is optimizing against the min-max game:

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

in which the variable G corresponds to the generator, D corresponds to the discriminator, variable $\mathbf{x}$ corresponds to the real image data, and $\mathbf{z}$ corresponds to the input noise, particularly a 100 dimensional Gaussian variable with 0 mean and 1 variance.

The main purpose of the discriminator is to maximize the recognition rate, corresponding to $\max_D$. However, the major intention of the generator is to generate images with the highest possible value of $D(G(\mathbf{z}))$ in order to fool the discriminator, which corresponds to minimizing in $\min_G$.

The equilibrium will be obtained whenever $p_{\text{model}}(\mathbf{x}) = p_{\text{data}}(\mathbf{x})$ and the discriminator is not sure about images being real or fake, which gives $D(\mathbf{x}) = \frac{1}{2}$.

One of the most important hyper-parameters is the batch size, B, referring to the sample number every network observes through the training per iteration. The major characteristics of batch size is to prevent mode collapse via the diversity encouragement during training stage and later throughout the inference phase.

In general, such joint network of generator and discriminator will utilize 3 hyper-parameters during the experiment: batch size B, capacity F, and depth of discriminator D. During the experimentation, different ranges of multipliers for D, F were chosen, particularly 1 to 3, whereas we decided to design batch size with $B = 32, 64, 128$. It is expected that increase in B, and F will lead to better training, and discriminator prediction, respectively.

2.3. Evaluation

In this section, we will discuss more details about the evaluation of joint-network performance. Particularly, it is noteworthy to include how the performance was evaluated using the training accuracy of the discriminator, losses of the generator $\&$ discriminator to highlight overfitting/underfitting issues.

Every possible combination was trained over $20 - 25$ epochs with optimizer Adam where learning rate for generator $\&$ discriminator were chosen as 0.002. We can have a deeper look to the Figures $2, 3, 4$ to observe loss functions $\&$ training accuracy over the trained epochs.

From the general view, one can argue that basic combination of $F = D = 1$, and $B = 32$ is capable of achieving best overall results for discriminator accuracy compared

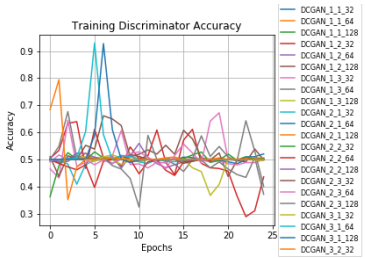

Figure 2. Training discriminator accuracy for DCGANs, the first number in the name means the size of F, the second number the depth D and the last number batch size B.

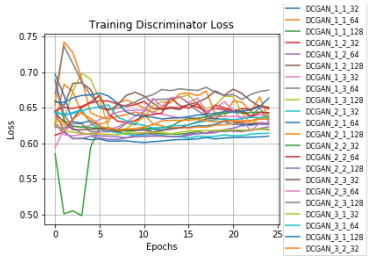

Figure 3. Descriptor loss for DCGANs, the first number in the name means the size of F, the second number the depth D and the last number batch size B.

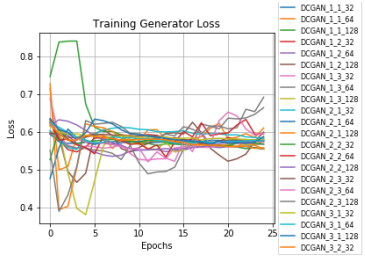

Figure 4. Generator loss for DCGANs, the first number in the name means the size of F, the second number the depth D and the last number batch size B.

to other outcomes. Moreover, the discriminator & generator loss were also converging faster compared to other networks. In the subsequent Figure 5, several samples from this particular DCGAN with relevant epochs were illustrated correspondingly.

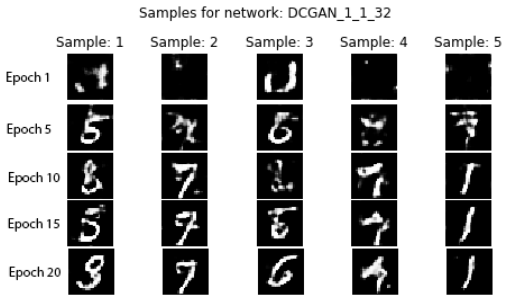

Figure 5. Sample images for DCGAN with $F = 1, D = 1$, and $B = 32$.

It is worthwhile to include that discriminator had approximately $310,000$ parameters, whereas around 7 million parameters for the generator. Interestingly, such simpler network architecture managed to outperform the more complex and balanced models, along with better extracting important features given the data complexity, and requiring least resources compared to previous alternatives.

Afterwards, we decided to apply virtual batch normalization to the proposed network architecture, in which we randomly choose half of the batch size in every epoch and normalize the other half corresponding to the previously chosen half. More detailed observations of the discriminator accuracy and generator, discriminator loss functions can be viewed in Figures $14, 15, \& 16$.

Nevertheless, it is very crucial to point out that there has not been considerable improvement compared to previous network model. From the generated numbers, we can argue that regardless of loss saturation in these graphs, generator was not able to improve further and from the visual characteristics, network was in fact producing unrecognisable numbers.

3. Class-Conditional Generative Adversarial

Compared to the Deep-Convolutional Generative Adversarial Network (DCGAN), Class-Conditional Generative Adversarial Network (CCGAN) accepts class labels both for the generator & discriminator, allowing sample generations targeting specific class which was impossible to implement previously.

3.1. Architecture

One of the key differences in this network compared to the observed model Figure 1 will be concatenation of the label into Dense layer in the generator part with original number of nodes. Meanwhile, for the discriminator, the labels will be processed first by a Dense layer with $28 \times 28 = 784$ nodes and later, reshaped into image dimensions where it follows by the concatenation to the original input.

3.2. Training

Taking into account the effect from label when added as an input to the architecture, the desired objective function will change accordingly into the following form:

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

in which l corresponds to the label, $p(l)$ indicates the distribution from which the label l was uniformly selected for generator training and $p_{\text{data}(\mathbf{x},l)}$ represents the original distribution of labels.

As performed in the previous experiment, the capacity (F) and depth of discriminator (D) values will range from 1 to 3, while the batch size will receive $32, 64$, and 128. Throughout the possible options, every combination will be trained over $20 - 25$ epochs along with hyper-parameters of learning rate 0.0001 and Adam optimizer for generator & discriminator.

3.3. Evaluation

In this section, we will provide more details on how evaluations were conducted and which comparisons were measured during the process. Considering the given network architecture allows labels, one can evaluate inception score and compare against LeNet, in which the latter model achieved 99.4% accuracy for the training dataset and 98.8% accuracy for the test set, along with the application of 5 epochs of training and Adam optimiser.

In the similar manner with previous experimentation, one may imply that after adding inception score to the observed metrics, simpler network architectures still perform marginally better compared to sophisticated complex algorithms, as depicted in Figures 6 & 17. Out of the possible combinations, $D = 1, F = 1$, and $B = 64$ reveals the best architecture, along with inception score of $\mathbf{94.7}\%$.

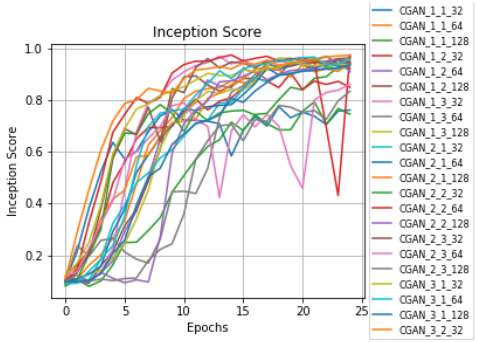

Figure 6. Inception scores for CCGANs.

One may argue that the quality of sampled images has improved considerably where class-label has been included as an input for this architecture, showed in Figure 7. Moreover, we may also highlight from the Figure 17 in which discriminator's accuracy remains almost constant from the beginning of training phase compared to the volatile discriminator accuracy occurring in DCGAN model as shown in Figure 2. Overall, the training process for generator had been more difficult compared to the discriminator's training phase, where early epochs were yielding ideal state of 0.5 accuracy.

As in the previous experiment, Virtual Batch Normalization did not enable better developments, mostly due to already implemented Batch Normalizations in the process. Nevertheless, it is crucial to point out how the label smoothing improved the inception score on average by approximately 3% per epoch, resulting in the best inception score **97.6 %** for such network model. More detailed view of confusion matrix is provided in Figure 8.

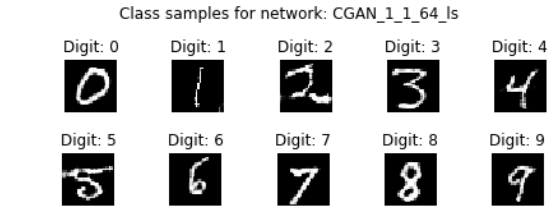

Figure 7. Sample images for CCGAN $D = 1, F = 1$ & $B = 64$ with label smoothing.

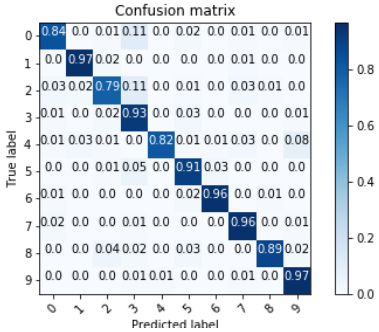

Figure 8. Confusion Matrix for CCGAN $F = 1, D = 1$ & $B = 64$ with label smoothing.

Diving deeper into class samples, we may argue one of the most wrongly classified numbers were 2 and 4, where some shape similarities occurred in those numbers. Moreover, there is a noteworthy fact in which MNIST dataset does not provide equal amount of samples per class number, subsequently worsening the observed accuracy metric.

4. Training with Synthetic Data

4.1. Data preparation

The network may be trained using the generated data from the CGAN in section 3. To guarantee a fair comparison, the same amount of training data was maintained, including proportion of individual classes.

4.2. Approach

When comparing the testing accuracy loss (after training with synthetic data) to that acquired using actual training data, it can be seen that using more real data rather than synthetic data yields better results(See figure 9). However, we get better results with training accuracy loss when the training data comes from one of the two types of training images (training data either mostly consisting of real images, or mostly consisting of synthetic images.) The model has a high performance even when the amount of real train-

ing samples is quite small (around 10%). However, this is improved a bit more when the percentage of real training samples is increased in the training set. The learning rate and batch size were the two network hyperparameters that we tried different values for. Choosing a good learning rate requires a balance between convergence time and performance.

Figure 10 shows the outcomes of optimizing the network parameters. A learning rate of 0.001 achieved the best results. It was also seen (Also shown in figure 10) that the varying the value of batch size did not make significant improvement in model performance when the model was already performing well under a given learning rate. Batch size 32 gave better results for different values of learning rate. Figure 11 shows the confusion matrix for this model.

Figure 9. Accuracy loss for different percentages of real data

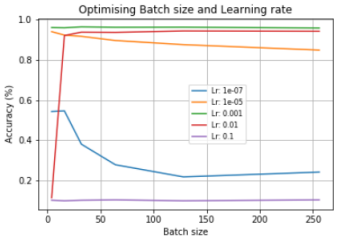

Figure 10. Trying different values for batch size and lr.

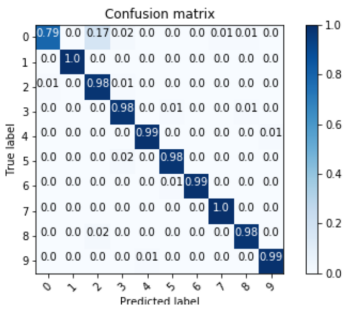

Figure 11. Confusion matrix on hybrid dataset.

4.3. Strategy change (optional)

It was seen that that class 0 was frequently misidentified as class 2. To improve on this, the proportion of actual and synthetic data was changed for that specific class. Prior to this, each class had the same percentage of synthetic data and the same number of samples. Increasing the proportion of real data did not guarantee an increase a better performance for each classes. So for this case, the percentage of real data for class 0 was increased to 40%(other classes' real data remained at 10%) and better results were obtained as shown in the improved confusion matrix in figure 12.

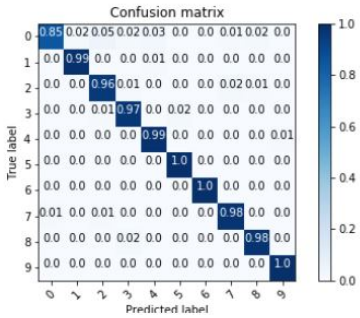

Figure 12. Confusion matrix on hybrid dataset after strategy change.

4.4. Analysis

This suggests that for simple datasets (such as the MNIST handwritten digits), GAN produced synthetic images may be used for data augmentation. However, this might not work on other datasets and works if it adds to the extrapolation power of GANS.

5. Comparison with Principcal Compoment Analysis

MNIST dataset was used as a training data for the PCA model. The n eigenvectors which corresponded to n largest eigenvalues were then used to extract n components. After that, each generated images were reassembled to their original size of 28 x 28 pixels. The different models are then compared using inception score.

5.1. Data and metrics

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 CGAN.

5.2. Results

In Section 3, the CGAN has acuracy of 97.6% against LeNet. This is the point at which the PCA accuracy should converge. As seen in Figure 13, there is a significant variation in performance when the number of primary components is increased. After 64 major components, the improvement became saturated. The pictures created by the top performing CGAN model outperform the images generated by PCA with one component. For PCA produced pictures with one main component, the inception score was over 10% lower.

5.3. Comparison

It's worth noting that GANs can learn more sophisticated distributions than basic linear PCA and hence make graphics. However, compared to employing the latter approach as a generative model, they require a significantly bigger dataset. GANs, in contrast to PCA, have a large number of hyperparameters to tune and are also more complicated to implement. Their deep learning technique also makes them a black box, with minimal explainability and inference ability. PCA, a well-known approach, needs far fewer computer resources and is far easier to execute.

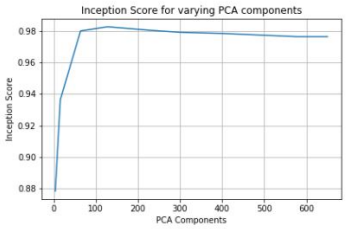

Figure 13. Inception score for different values of PCA principal components

6. Conclusion

In this coursework, we experimented with DCGAN, CCGAN and PCA for generating sample images based on the MNIST dataset. The best observed network architectures for DCGAN & CCGAN were with parameters $F = 1, D = 1$, and batch sizes $B = 32$ & $B = 64$, respectively. Moreover, the observed inception score which was evaluated accordingly with best CGAN architecture model obtained **97.6 %**. During the best performance of PCA as a generative model, we should underscore the important points where gradient's values were changing abruptly, as depicted in the Figure 12. Since such spots occurred at **64** components, we decided to continue with that dimensional space to generate images.

One more aspect during these experiments was that more complicated network models gave a better performance by a mere margin, while such use could be limited due to memory complexity and computation time. Moreover, GANs seemed to be useful for implementing data augmentation in regards to MNIST dataset, in which small portion of real data boosts the accuracy metric. In the end, in-depth analysis of PCA indicated that 64 of the 784 potential features of the images were sufficiently enough to achieve a higher evaluation.

For the future proceedings, one may further improve the training by pre-training the discriminator before the generator's training. Moreover, Probabilistic PCA could also be another methodology that needs to be explored for more in-depth analysis of data augmentation.

References

[1] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," $\mathrm{CoR}\, R$, vol. $\mathrm{abs}/1511.06434, 2015$.

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

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

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

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

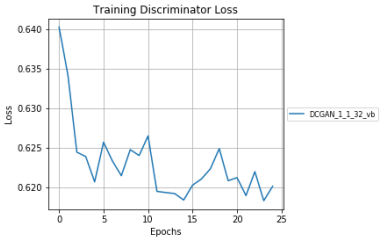

Figure 15. Discriminator loss for DCGAN with F being 1 , D being 1 and trained with batch size B being 32 together with virtual batch normalisation.

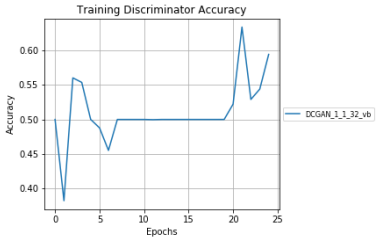

Figure 16. Discriminator accuracy for DCGAN with F being 1, D being 1 and trained with batch size B being 32 together with virtual batch normalisation.

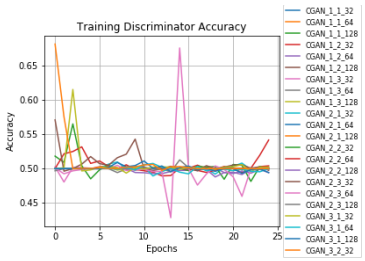

Figure 17. Training accuracy for CCGANs, the first number in the name means the size of F, the second number the depth D and the last number batch size B.