Computer Vision Coursework 2 - Generative Adversarial Networks

Paul Streli - ps4715
Imperial College London
CID: 01103106

Karoly Horvath - kth15
Imperial College London
CID: 01088730

Abstract

The following paper examines various Generative Adversarial Network (GAN) architectures to generate synthetic handwritten digits based on the MNIST dataset. The focus is put on Deep Convolutional GANs (DCGANs) and their extensions to Conditional DCGANs (CDCGANs). They are compared based on the Inception Score and the Frechet Inception Distance and the generated pictures are used to train a digit classifier. The best performance was provided by a CDCGAN with three convolutional layers in the generator and in the discriminator trained using one-sided label smoothing. This network was used to augment the MNIST dataset for training the LeNet architecture. The highest recognition accuracy reached was 96.25%.

1. Introduction

The advent of the Generative Adversarial Network (GAN) sparked an increasing interest in using Deep Learning for generative models (see Appendix 7.1).

The aim of this coursework is to experiment with different GAN architectures to synthetically generate black-and-white (i.e. one channel) pictures of handwritten digits (ranging from 0-9) (see Appendix 7.4). For this, the MNIST dataset is used which consists of 60,000 training and 10,000 testing samples of resolution 28x28 pixels. The resulting images will be evaluated based on a qualitative basis, the required training time, the stability of the generation process, and supplementarily using the Frechet Inception Distance (FID) (see Appendix 7.3). Furthermore, as the outputs of the Conditional GAN (CGAN) include a class label, they can be quantitatively assessed by the Inception score and their ability to improve the recognition accuracy of a handwritten digit classifier when used for data augmentation.

2. DCGAN

Convolutional Neural Networks have been proven to operate very efficiently on visual data for classification. When Deep Convolutional Neural Networks are implemented as discriminator and generator in a GAN, the resulting architecture is called a Deep Convolutional GAN (DCGAN). Due to its properties, such as invariance to translation and the use of fewer parameters, it is a suitable choice for image generation.

As reference, the MNIST DCGAN architecture presented in lectures [8] was tested adapted from [7], which consists of five convolutional layers in the generator and the discriminator (see Appendix 7.5 for a more detailed description of the discriminator and generator architecture).

For training, the discriminator and generator were alternately trained with a batch size of 100. For each of the 25 epochs, all training data was considered. While the GAN learned to generate high-resolution images of several digits after few epochs, the architecture exhibited several issues. The discriminator was able to overpower the generator. This resulted in instability causing the generator to create similar looking noise images. The generator eventually recovered to a certain extent, but the range of created digits was limited indicating mode collapse (see Appendix 7.6). The high number of output features facilitates overfitting and makes it harder for the generator to generalise over the whole data distribution. Furthermore, due to the high number of channels and layers, the training was slow with an average epoch time of more than 710s (using Google Colab's Tesla K80 GPU).

To resolve this issue, it was decided to remove the fourth layer of the generator and half the channel sizes of the previous layers (512-256-128). Previously the discriminator took an image with resolution 64x64 pixels as input and the original training dataset was resized to this size. In this case, the output of the generator would be an image of size 28x28 pixels - lifting the need for the resizing of original images - achieved by implementing the transpose of a 13x13 convolution filter with no zero-padding and stride 1x1 to upsample from the previous layer of size 16x16. The resulting digits were of similar quality, and due to the lower resolution they did not seem to suffer from mode collapse or instability and required a training time of only 300 seconds.

The training time per epoch was further reduced to 130 seconds by removing the fourth layer of the discriminator without seemingly any trade-offs in terms of image quality (see Fig. 1). While it reduces the chance of overpowering, weakening the discriminator can have detrimental effects as its ability in giving accurate feedback reduces. Neverthe-

less, the results pointed to the fact that the reference network was overdimensionised for the given task and that a more compact network would guarantee stability. Fig. 2 shows that the GAN started to converge after 20 Epochs with only very small improvements in FID onwards (min. FID: 17.30). G and D also seem to operate in a stable relation, with G_{loss} only slightly increasing on average over time. When taking a closer look at the generated digits in Fig. 1, it can be seen that some of them look very real and easily recognisable. Others, however, are more difficult to distinguish and look like a mix between different digits. As handwriting varies and the original dataset is rather large, it could either be that similar shapes were written down by people or that the GAN did not manage to fully understand the shape boundaries for all varying forms of digits. For a closer analysis on the discriminator loss see Appendix 7.7.

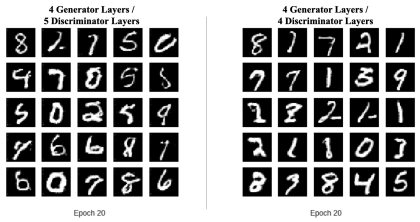

Figure 1: Generated images with compact DCGAN architecture and batch normalisation

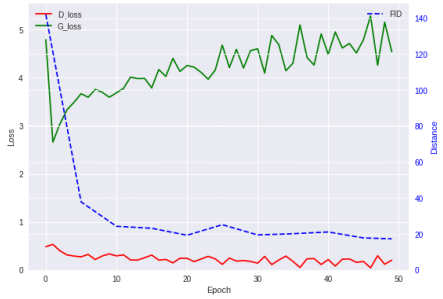

Figure 2: Training history for compact DCGAN

The effect of batch normalisation was investigated by removing all normalisation layers from the GAN. While it needed a higher number of epochs to create sensible digits and certain images were of a lower semantic quality, the generator still managed to learn the underlying distribution up to a certain level (min. FID: 24.40) (see Fig. 21, Appendix 7.8). However, as the training time per epoch only slightly decreased (123 seconds), the added regularisation effect and the increase in learning speed justify the use of batch normalisation. Moreover, when rerunning the code it happened that the GAN sometimes exploded and only generated black pictures hinting to the fact that the batch normalisation guarantees more stability as well.

Increasing the number of discriminator updates per iteration to improve the feedback quality, did not really increase the visual quality of the digits (min. FID: 23.44). However, as expected, training time increased to 295 seconds per epoch. Thus, given the same overall training time (45 min - 20 epochs) the equally balanced GAN created images of higher quality (see Fig. 22, Appendix 7.8). Another indicator that the discriminator is already strong enough without training it over several iterations. Increasing the batch size led to a slowdown in image quality improvements per epoch caused by the lower number of iterations per epoch. Since each epoch still took 127 seconds, this adjustment seemed counterproductive. The learning rate for the Adam-Optimizer [3] was also changed from 0.0002 to 0.002. While FID score (min. FID: 20.6746) slightly rose due to the smaller capability in making fine grained adjustments, the visual quality of the images and the learning history do not show lasting effects (see Fig. 23, Appendix 7.8). Further increasing the learning rate resulted in instability and the generation of black images after few epoch, showing that the changes in weights were too large for the GAN to make reasonable weight adjustments.

Overall, the reduced architecture of the DCGAN presented in lectures with the original settings seems to be a good choice for the unsupervised generation of MNIST digits. The chosen GAN architecture revealed robust performance as it was able to recover the underlying training distribution and generate new samples (see Appendix 7.9). Please see Table 1 in Appendix 7.10 for a summary of the different DCGAN architectures' results.

3. CGAN and Inception Score

To further improve the generated image quality, the Conditional GAN (CGAN) architecture was examined. The first model implemented is an extension of the previously tested DCGAN, where the input of G and D are extended to include an additional vector, l, which contains information about the class label of the images. In the traditional DCGAN, there is no control over what class of images are generated. Feeding l into both the G and D effectively instructs G to generate images belonging to a certain class and allows D to distinguish between real and generated pictures better using the additional label information (see Appendix 7.2).

In the implementation, l is created by randomly sampling from a uniform distribution - between 0 and 9. This value is encoded into a one-hot vector representation, which is then concatenated to the end of the noise vector when taken as input to the generator. Similarly, the supposed class labels of the generated images and the real class labels of the training images are concatenated to the pictures along the third dimension, with l repeated per pixel before being fed into the discriminator.

The baseline architecture consists of three convolutional

layers in the generator and in the discriminator (see Appendix 7.11 for a more detailed description of the model). The code was adapted from [6] to fit the above described architecture, with a bug fixed that was preventing the generation of digit "9". This baseline architecture was able to generate visually recognisable images. The image quality improved until about 10 epochs, which can be observed both qualitatively on the generated images and on the improving FID score as well (see Fig. 3 and Fig. 4).

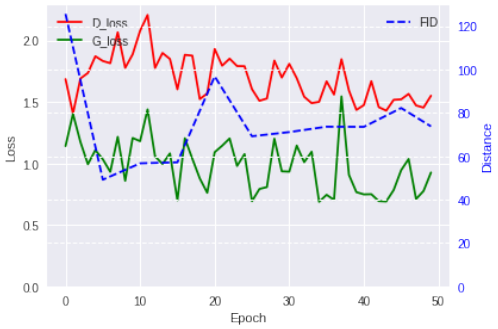

Figure 3: G_{loss}, D_{loss} and the FID score at each epoch of the baseline CGAN

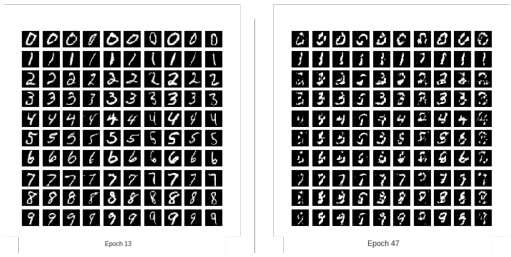

Figure 4: The generated digits at epoch 13 and epoch 49 of the baseline CGAN

While for the DCGAN, the FID score was used as quantitative measure for the quality of the generated images, for CGANs, another metric, the Inception score can be introduced. The score can be expressed as in Equation (4) in Appendix 7.12. To calculate the Inception score, a classifier network is required, which was chosen to be the LeNet architecture introduced in the second Lab session, consisting of two convolutional and three fully connected layers, trained on the MNIST training set. When the MNIST test dataset is passed through the trained LeNet model the classification accuracies reach up to 94.7%, which indicates that this network is very good at recognising hand-written digits from the original distribution. Hence, comparing this with the recognition accuracy of the generated set will give a good indication of the quality of the generated distribution.

The Inception score of the first CGAN architecture's best epoch was 4.75, and the highest Inception accuracy achieved was 84%, which is lower than the LeNet's recognition accuracy on the test set, however it is still reasonably high. Example success and failure cases are presented in Appendix 7.12 on Fig. 26. It is observable that the unrecognised images visually look worse than the others. This means that the generated images are approaching in quality the original dataset's samples, but they are not as high quality yet. In addition to the Inception score, the recognition accuracy of the LeNet of the generated images is also displayed on the left of Figure **??**. It clearly shows the oscillatory behaviour of the image quality.

In hopes of improving the quality of the generated images, another architecture was examined, with two extra convolutional layers appended both to the generator and to the discriminator. This new architecture did not result in an improved image quality, due to the generator overpowering the discriminator very early in training process, slightly recovering by the last epoch. This can be observed by the measured FID scores as well as the large variation of the G_{loss} and D_{loss} and the significantly decreased Inception score and Inception accuracy compared to the original architecture, presented on Figures 27 and **??** (see Appendix 7.13), which are in line with the perceived image quality, shown on Figure 29 in Appendix 7.13. The Inception score and the Inception accuracy reflect this behaviour. The best Inception recognition accuracy achieved was 68%, which is lower than the previous architecture's.

To overcome this issue, the G-D balance of the model was updated to emphasize the discriminator, which resulted in very low recognition accuracies after a few epochs, indicating that the discriminator won over the generator too quickly, which resulted in unrecognizable images generated. Experimentation with decreasing and increasing the number of layers in the generator and the discriminator and varying the G-D balance did not lead to any improved performance over the original architecture. Adding more layers to the CGAN seems to introduce too many trainable parameters for the given recognition task resulting in instability. The high number of layers causes overfitting and makes it harder for the generator to generalise over the input data distribution of the MNIST dataset.

As simpler models had better performances, another architecture is proposed. A classical CGAN was implemented where the convolutional layers were changed to dense layers. This architecture reached good quality generated images after significantly more epochs, however when the Inception score and accuracy reached their peaks, the performance approached the conditional DCGAN's metrics. The best Inception accuracy was only 6% off the DCGAN's (78% at epoch 47 vs 84%) and FID score was close as well. (See Fig. 30 and Fig. 31 in Appendix 7.14) From the Figures it is visible that even though the original CDCGAN still performs better, the CGAN has the advantage that the

oscillatory behaviour disappeared. The generated images are getting visibly recognisable after about epoch 70. Fig. 32 shows the evolution of the image quality in Appendix 7.14.

Similarly to the DCGAN, the effect of batch normalisation was also investigated for both CGANs by removing the normalisation layers from the architecture. In similar manner to the DCGAN, the model generated good images after a few extra epochs, but the performance of the generator did not get much worse. The training time was barely affected by using batch normalisation, which lead to the decision of including batch normalisation layers in the future experiments, in addition to the fact that batch normalisation increased the stability of the model with its regularisation effect.

To reduce overconfidence of the model, one-sided label smoothing was implemented. The discriminator is penalised, by setting the target label value to 0.9 instead of 1. This is expected to prevent the generator exploiting the discriminator by less emphasizing particular images that the discriminator is very confident with over the whole input distribution. One-sided label smoothing led to minor improvements. Qualitatively it is hard to notice the difference between the original architecture and the one with one-sided label smoothing (see Fig. 33 in Appendix 7.15), however, Inception and FID scores show improvements (see Fig. 5). Fig. 6. compares the Inception scores of the architectures with and without one-sided label smoothing. It is important to note that the variation of the hand-written digits is bigger with this new architecture. This means that the generator can generate digits that are more different, but realistic - spreading the input distribution more.

An important aspect of the architectures is the way the labels get fed into the models. Previously the layers were simply concatenated to the inputs to both G and D. Another possible way to sprinkle the label information into the system is passing them through an input convolutional layer as well, which in turn gets concatenated with the input convolutional layer of the images/noise. This architecture was investigated, but resulted in worse results, (see Fig. 34 and 35, Appendix 7.16 on). The best Inception score is 3.5 and with training it gets iteratively worse and worse. The same trend is noticed with the FID score and the Inception accuracy. This shows that the label information had more use in the other architecture as in this configuration. The extra layer added unnecessary complexity to the model.

Therefore, based on the numerically summarized results in Table 2 in Appendix 7.17, the best proposed conditional GAN architecture is the original CDCGAN, with 3 convolutional layers for G and D, and one sided label smoothing. To select the best parameters for the architecture, a validation set was used. Out of batch sizes of [25, 50, 100, 128, 200], learning rates of [0.2, 0.02, 0.002 and 0.0002], the best results were reached with a batch size of 100 and a learning rate of 0.0002. This architecture was also able to recover the underlying training distribution and generate new samples (see Appendix 7.9).

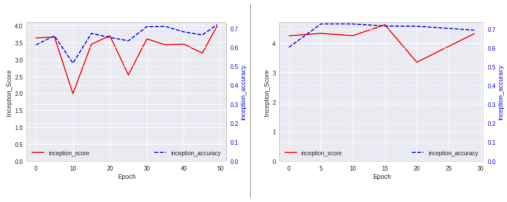

Figure 5: The Inception score and the Inception accuracy of each epoch of the baseline CGAN and its one-sided label smoothened version

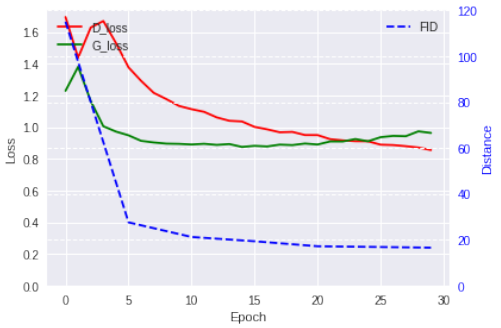

Figure 6: Training history of the CGAN architecture with one-sided label smoothing

4. Data Augmentation - Re-training the hand-written digit classifier

One way to improve recognition accuracy of a classifier for a given training set is to apply data augmentation [11] (see Appendix 7.18). For this reason, it would be interesting to reuse the CGAN generated digits to amplify the given training set and to analyse the response on recognition accuracy for a real test set. The resulting behaviour functions as another indicator for the quality and novelty of the created images.

To start with, it was decided to reuse the pre-trained (30 epochs) CDCGAN with one-sided label smoothing from Section 3 for data generation. The same previously used LeNet architecture was employed as classification network and trained on different combinations of original and generated data (subset sizes: [0, 600, 6000, 12 000, 30 000, 60 0000] corresponding to [0%, 1%, 10%, 20%, 50%, 100%] of the complete training data amount). The subsets were combined and shuffled before the training in batches over 30 epochs started. Afterwards, the recognition accuracy on the test set was collected. Looking at Fig. 7 and Fig. 36, 37, 38 (Appendix 7.19), the following points can be noted:

- Generally, the test accuracy increases with the overall amount of data used, even though increases were very small from a certain point on and the curve reached a plateau (Fig. 36). The best accuracy was measured to be 96.25% when all 60 000 generated and 60 000 original data samples were used. This is higher than for the case without any data augmentation (94.7%) and proves that the generated data contains new information that helps the network to generalise better even for the very large given original training dataset.
- Training only on generated data results in a maximal test accuracy of 47.93% (60 000 generated samples). Interestingly, this increases very rapidly to 89.00% if only 600 (1%) more real samples are added for the training process. The less generated data was used the more original training data was necessary to reach higher accuracy levels (see Fig. 38 for the monotonically increasing plots). A possible reason is that created digits do not capture all significant attributes of the original images but are good enough to bring the classifier's weights to the correct region. The original digits then give the network the confidence to fine-tune its parameters which results in rapidly increasing accuracy. With increasing generated data size the training time and the diversity within the data increases and with it so does the chance of a wider and more realistic selection of samples, leading to the tendency of better results.
- Data Augmentation did not always guarantee better results. The visible negative dips in Fig. 37 and the line with 600 generated samples in Fig. 38 (lies below line with no generated images) are practical examples for the detrimental effects. A likely reason is that the randomly generated samples were of poorer quality by chance. Due to the smaller set size the variance of the mean quality is higher and therefore the probability for generating sets with a larger proportion of misleading examples is higher.

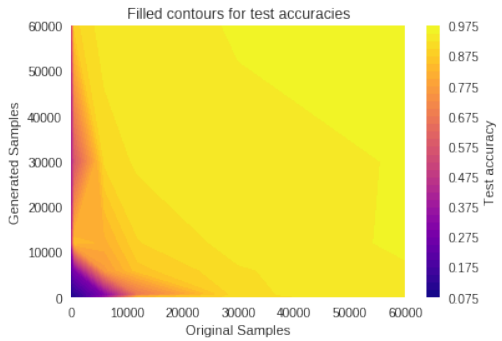

Figure 7: CDCGAN baseline: Contour Plot for test recognition accuracy over differently combined training data

To resolve the latter issue, it was decided to only use created digits that were previously correctly classified by the Inception network and are thus expected to be more realistic and of higher quality. Indeed, Fig 39 does not show kinks and is monotonically increasing, and the line ordering in Fig. 40 is more as expected compared to Fig 38. However, the maximal achieved test accuracy (60000 generated/original) was only 95.76% and the generated data was less of a boost in Fig. 39 than in Fig. 37 when the generated data was used in combination with the real one. While the quality check avoided the occurrence of outliers, it also led to the rejection of samples with a high amount of new information as the quality criterion was based on the training distribution (Inception network was trained on the training dataset). However, when training only on the generated examples higher accuracies were achieved compared to the previous set-up likely due to the avoidance of misleading outliers that do not get corrected by real data in this case.

Another training strategy was to train the LeNet for the first 15 epochs only on the generated data and then to fine tune the parameters in the following 15 epochs with the original samples. The maximum test accuracy was obtained with 95.55%. Otherwise, the results were quite similar to the case without quality check. The classification network performed slightly better than previously when only few original samples where used, indicating that the neural network was able to better adjust and emphasise the higher-value original images in the 15 fine-tuning epochs. In the high set size regime, it performed slightly worse probably undoing the positive generalising effect (see Fig. 41, Appendix 7.19). For further experiments with Data Augmentation, see Appendix 7.19.

Overall, the best results were obtained with the baseline CDCGAN. Rejecting lower Inception scores avoids negative effects on test accuracy, but somehow limits the amount of new information learnt. Instead all data could be used for the given case and a validation set could be utilised to check for negative outcomes. While the data augmentation had overall very positive effects, especially when used on smaller original training datasets, one must not forget that the generated images came from a GAN that was trained on the whole training dataset (60000 images). Thus, given a a smaller training dataset for GAN training, the effect of data augmentation is expected to be less remarkable.

5. Conclusion

Out of the examined unsupervised DCGANs, the Conditional GANs and the Conditional DCGANs, the best generation policy was obtained by the baseline CDCGAN (based on FID). Training a classifier based on the augmented MNIST training set resulted in the highest classification accuracy reaching up to 96.25%. It was shown that augmenting the training set had a positive effect on performance.

6. Bonus Questions

6.1. Visualisation of penultimate classification layer's activations

An interesting evaluation experiment is to feed 100 original test samples and 100 synthetic (from best performing CDCGAN) samples through the LeNet classification network, pretrained on only real samples, and to visualise the respective penultimate layer's activations. These are the final characteristics that are used to assign the respective label probabilities to the individual images. This technique has similarities with the FID score, however this time a network is used that is trained on the same data type. To visualise the 84 dimensional activation vector, the samples are projected to the first and second principal components over the combined generated and test activation set using PCA. This will guarantee the biggest possible separation between the samples in a 2D space due to the fact that the first two PCA directions capture the most variance. From Fig. 8 the following observations can be made:

- While there are several generated samples (green) that lie within the cloud of original data points (blue), there is a great number of synthetic samples that lie outside (but very close) indicating that the GAN was either able to create samples with new characteristics or failed to fully capture the underlying data distribution. However, it needs to be mentioned that the PCA dimension reduction emphasised these differences over the 84 dimensions.
- The original data is more equally spread out capturing the effect of variations in human writing. While the generated samples span a wider range (possibly due to outliers), there is a cluster of points that lie quite close to each other indicating that some of the created images share very similar characteristics or are expected to be generated in their given constellations more frequently. However, the separation between the other generated samples is a positive signal for variability in the generated data.
- Generally, while it is interesting to find out how the classifier network treats generated and original data differently on average, this plot does not prove how the classifier makes its final decision as the LeNet might emphasise certain of the 84 dimensions in different ways for different label predictions.

To check the differences in prediction behaviour for the LeNet, the confidence score distributions between generated and real data could be compared with each other.

As it has been stated in Section 4, the LeNet performed very well on the training data, reaching up to 94.7% recognition accuracies. This is reflected on the histogram pictured in Figure 9, which shows that the classifier was confident for most of the pictures, hence high number of pictures belonging to the higher probability bins. In comparison to this, when the generated images were used, only 35% of the images were in the highest bin. The majority of the images are still above the 0.5 confidence, but this indicates that the classifier was on average less confident with these samples. It is important to note that some samples have very low confidence, which are most likely either the pictures that have considerate amount of new information contained, or they are just of bad quality. Thus, they are harder for the network to recognise.

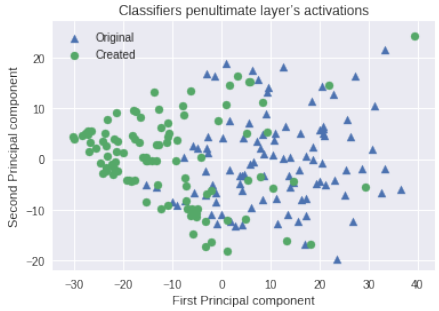

Figure 8: Visualisation of the classification embeddings in a 2D space spanned by the first two principal components

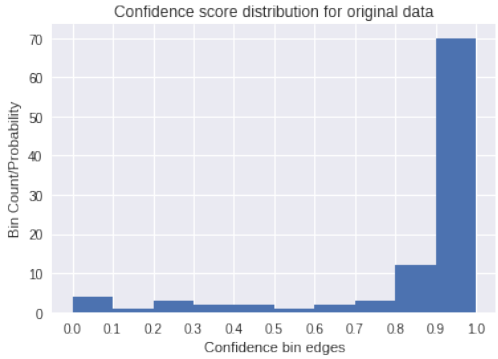

Figure 9: Histogram of the confidence scores of the original dataset

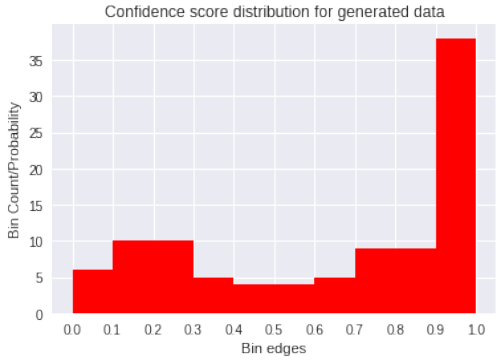

Figure 10: Histogram of the confidence scores of the generated dataset

6.2. Feeding the LeNet's classification loss into the CGAN

To improve the generated images' quality, couple of other methods could be used. A modified CGAN architecture was created where the classification error of the LeNet (see Section 3) was taken into account in the loss function. This is expected to give better results, as the Discriminator and the Generator gets feedback whether the LeNet classifier - which was trained on the distribution that the GAN is trying to model - predicted the digit on the image correctly. In fact, this change in the system's architecture did lead to minor improvements on the generated image quality. This classification loss helps the generator to further improve on its ability to create images that are of the specified class label (represented by input vector *l* concatenated to the input noise). This was represented by an improved FID and Inception score and a higher Inception accuracy. This implementation is based on the best CGAN architecture presented in Section 3 (the baseline CGAN with one-sided label smoothing).

The loss function in this case is formulated as $-(\log(D(G(\mathbf{z})) + Inception\_score \times \alpha)$, where α is the feedback parameter of the Inception score. $\alpha = 1$ was selected from different values ([0.5, 1, 2, 5, 10]) using the validation set of the MNIST dataset.

Figures 11, 12 and 13 show that only moderate improvement were found. Qualitatively the generated images were already reasonably good with the regular CGAN implementation, and thus improvements are not visible, and the improvements on the best FID score (16 vs 17) and the Inception score (5.23 vs 4.98) are marginal. Since the Inception accuracy is also very similar (slightly lower - 82% vs 84%), it is overall hard to say whether this architecture performs better or not.

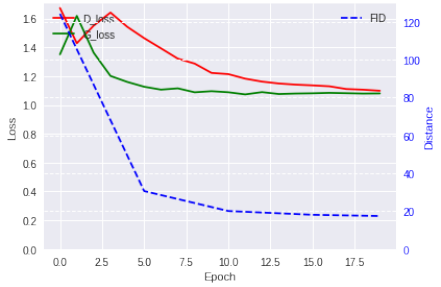

Figure 11: The training results of the CGAN architecture with the LeNet's loss fed back into it

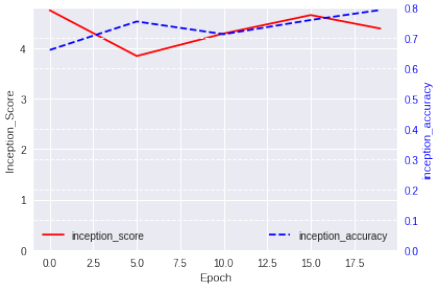

Figure 12: Inception score and accuracy of the CGAN with the LeNet's loss fed back into it

Figure 13: The generated images by the CGAN using the LeNet's loss to improve on the image generation quality

7. Appendix

7.1. GAN and its Objective Function

In its standard form, the GAN architecture consists of two neural networks, a discriminative model - the *Discriminator D* - and a generative model - the *Generator G*. The discriminator tries to detect whether an input sample comes from the original dataset or was created by *G*. Based on *D*'s feedback, the generator tries to adjust its weights so that it becomes more difficult to distinguish its output from the real distribution [5]. This minimax game between the two players *G* and *D* is mathematically represented by the value function $V(D, G)$:

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

In Equation (1), G is a function $G(\mathbf{z}; \theta_g)$ with parameters θ_g that maps a random input noise variable $\mathbf{z}$ to the training data space. The discriminator function $D(\mathbf{x}; \theta_d)$ with parameters θ_d returns a scalar representing the probability that its input comes from the original dataset. [5]

7.2. CGAN Objective Function

Feeding the labels into the generator and the discriminator results in a slightly modified objective function for the CGAN.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x},l\sim p_{data}(\mathbf{x},l)}[\log D(\mathbf{x},l)] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z}),l\sim p_l(l)}[\log(1-D(G(\mathbf{z},l),l))] \quad (2)$$

In Equation (2) $p_l(l)$ is the distribution over the classes. The output of the model is controlled by the controlling variable *l*, and D(**x**, l) and G(**z**, l) show that the generator and the discriminator are generating an image that corresponds to the given label *l*.

7.3. Frechet Inception Distance

For further curiosity, the FID score was also implemented to measure the performance of GANs. FID compares the statistics of generated samples to real samples in an unsupervised manner (i.e. no labels are needed). A set of generated and original images is put through a pretrained InceptionV3 network and the activations of the pool3 layer are recorded. Viewing the resulting outputs as continuous multivariate Gaussians, the FID can be calculated as their distance (Wasserstein-2 distance):

$$FID(r,g) = ||\mu_r - \mu_g||_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}}) \quad (3)$$

(μ_r, Σ_r) and (μ_g, Σ_g) are the mean and covariance of the respective real and generated data distributions.

Lower FID is better. FID aims to improve on the Inception score, is said to be consistent with human judgments, detects intra-class mode dropping and has good characteristics in terms of discriminability and robustness. [2]

7.4. MNIST Dataset

Example images of the MNIST set are shown on Figures 14 and 15.

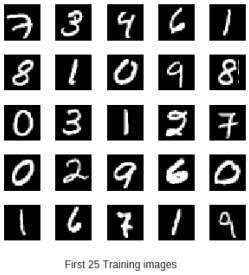

Figure 14: First 25 real training images of MNIST (cmap='gray')

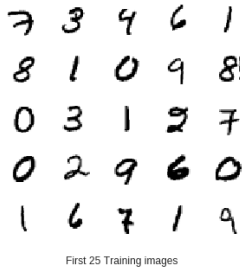

Figure 15: First 25 real training images of MNIST (cmap='binary')

7.5. DCGAN Architecture

The baseline DCGAN's generator takes a random noise vector with 100 channels as input. This is upsampled through five transposed convolution layers with filter size 4x4 (upsampling through the transpose of a non-zero-padded unit-stride convolution in the first layer and the transposes of a zero-padded 2x2 strided convolution in the rest of the layers [4]) and exponentially decreasing channel size to a 64x64 pixel image. The first four layers encompass batch normalisation, using inference of a moving average during testing [9], before their Rectified Linear Unit (ReLU) activation function. The output of the last layer is passed through a *tanh*-function. The generator tries to minimise $-\log D(G(\mathbf{z}))$, a slight heuristical change from Eq. 1 (Appendix 7.1) to provide enough gradient for *G* to learn from [5].

As the discriminator takes an image with resolution 64x64 pixels as input, the original training dataset needs to be resized to the same dimensions. The discriminator network consists of five convolution layers with filter size 4x4 and exponential increasing channel size reaching a maximum of 1024 channels in the fourth layer. The first four layers implement zero-padding, a stride of 2x2 and a ReLU function. All but the first make use of batch normalisation. The output layer reduces its input to a single scalar value that is passed through a sigmoid activation function to give a confidence value determining whether the input was real (1) or fake (0). *D* uses $[-\log D(\mathbf{x}) - \log(1 - D(G(\mathbf{z})))]$ as loss function.

7.6. Reference DCGAN Results

From Figure 18, it can be seen that G_{loss} is relatively small at the beginning as the discriminator has not been trained well yet. The discriminator and the generator iteratively improve until the generator manages to generate digit images. While some numbers are clearly recognisable, some of the examples do not resemble digits indicating that the generator is still in a learning process. Over time, the discriminator continuously manages to decrease its loss and thus becomes better in differentiating between real and

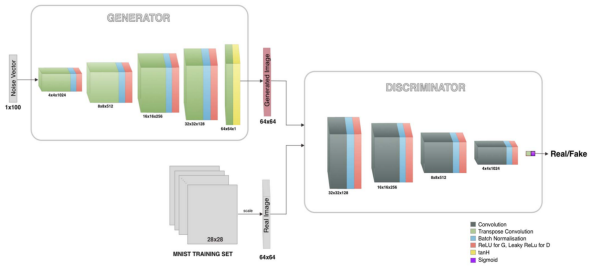

Figure 16: DCGAN Architecture proposed in lectures [8]

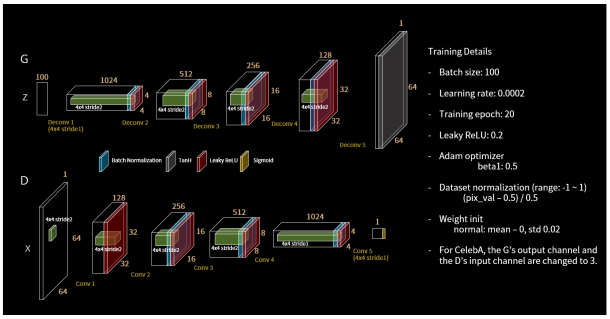

Figure 17: Reference DCGAN Architecture with detailed information about settings [7]

generated data. At some point, it overpowers the generator which loses track on what are sensible imaged and collapses in creating noisy images (see Fig. 19 - Epoch 9). Eventually, *G* recovers but generates a very limited range of lower quality digits (Epoch 24). If the training process would be continued, it can be expected that the GAN would cycle between the previously mentioned stages. The FID score gives a more objective metric for the quality of the generated pictures than G_{loss}. It can be seen that it soars as *G* gets overpowered and that the GAN does not fully recover again.

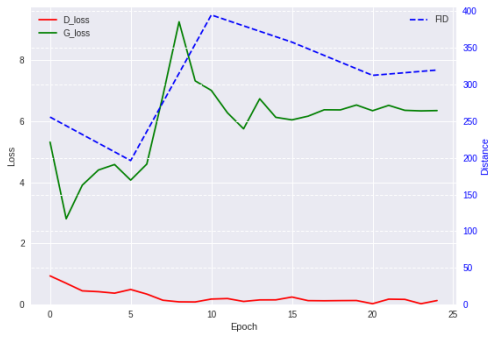

Figure 18: Training history for reference DCGAN

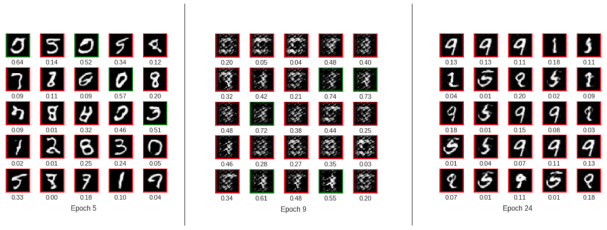

Figure 19: Digit images created by reference DCGAN (with Discriminator probability)

7.7. Discriminator predictions on generated, training and test images

The resulting generator and discriminator error allow only a limited inference on the visual quality of the generated images which is why FID was used as a more objective measure. In case of a weak discriminator, the generator error might be quite low even if the created digits do not resemble the original dataset. As the discriminator increases its capabilities, the generator loss might increase even though its generation abilities have improved. Thus, due to the interdependencies of this minimax game, the generator and discriminator errors do not signal a clear stopping condition. Generally, it is expected that *G* and *D* improve with the number of iterations in a stable and well-balanced setting. Given the assumption that both networks have big enough capacities to take on any function, the generator would completely capture the input distribution causing the discriminator to assign an input sample, generated or real, on average the probability of 0.5 [5]. This assumption does not hold in practise due to the restricted parameters of the network. However, it is still interesting to check how often and on which generated and real images the discriminator fails after both networks were trained over several iterations and reached a certain balance.

From Fig. 20, it can be seen that after 49 Epochs the generator is able to create mostly visually realistically looking images. The discriminator labels around $\frac{2}{3}$ of all pictures as fake (score lower than 0.5). The decisions cannot always be rationally followed by visually looking at the selected images. On average, more training and testing images were recognised as real with the training digits receiving higher scores. While this is no surprise for the training set as the discriminator was especially trained on these images, the difference between generated and test images indicates that there is still a distinguishable difference and that the generator did not perfectly capture the underlying distribution. In Epoch 50, the discriminator starts to label more images as fake to correct the errors made in the previous epoch. This oscillating behaviour between assigning once more fake and in the next epoch more real labels could be observed over the whole training process and is somehow also captured in

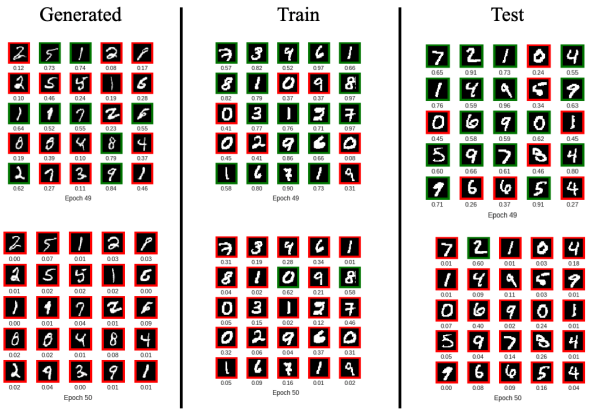

Figure 20: Predictions of Discriminator for generated, training and test images after 49/50 epochs (green - real, red - fake) - Discriminator/Generator: 4 layers with batch normalisation

the saw-tooth behaviour of G_{loss} in Fig. 2.

7.8. Results for varying hyperparameters and compact architecture

Figures 21, 22 and 23 show the Training results when different hyperparameters of the compact DCGAN architecture are varied.

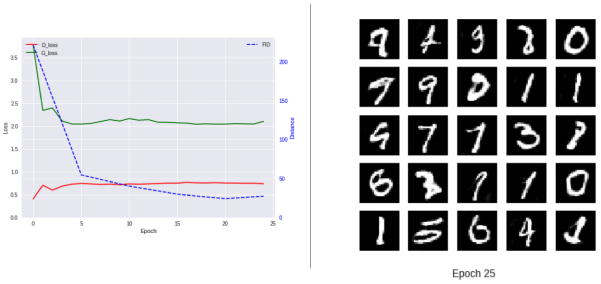

Figure 21: Training history and created digits for compact architecture without batch normalisation

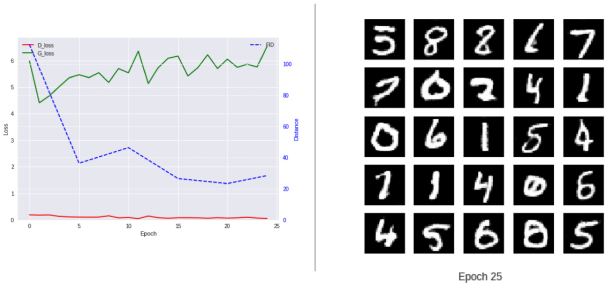

Figure 22: Training history and created digits for compact architecture with 3 discriminator update iterations per generator update

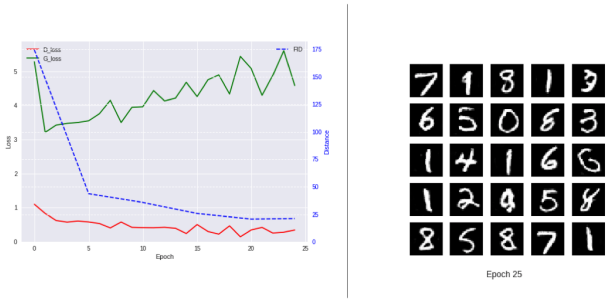

Figure 23: Training history and created digits for compact architecture with bigger learning rate (0.002)

7.9. Variance of generated data from training dataset

One of the issues that can arise with a trained GAN is that it does not generate new samples but just remembers images seen in the training dataset and just recreates those. While FID includes intra-class variance and thus gives an indication that the generated samples differ more and more with decreasing FID, this can also be checked by visualising the nearest neighbours of each generated image in the training set. Fig. 24 shows the four closest images (by Euclidean

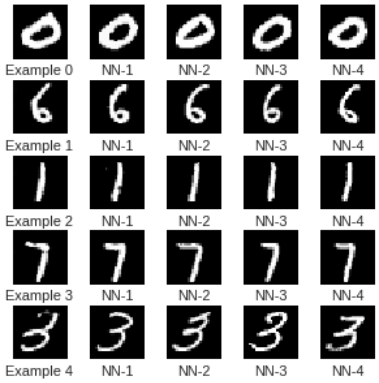

Figure 24: Generated examples with four nearest neighbours in the training set (compact architecture with 4 generator/discriminator layers, batch normalisation and 25 training epochs)

distance) in the training set for several synthesised digits. It can be noticed that due to the large training set size there are several images that look very similar and only differ in very small details. For this reason, it is justifiable that a generated digit also resembles its closest neighbours. However, slight changes in shape are recognisable and indicate that the created images are not only pure copies but differ similarly from the training images as the training images differ among each other.

Similar evaluation was done with the best CGAN's ar-

chitecture as well. The four nearest neighbours are shown on Figure 25.

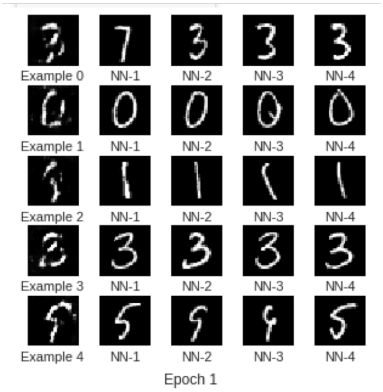

Figure 25: Generated examples with four nearest neighbours in the training set (Baseline CGAN architecture with one-sided label smoothing)

7.10. Summary of the DCGANs' results

Table 1 summarises numerically the results obtained from the various DCGAN architectures.

Architecture	Best FID Score
Baseline	30
Compact	17.3
W/O Batch N.	24.4
G-D Balance.	23.4
LR Increased	20.7

Table 1: Numerical evaluation results of the various DCGAN architectures

7.11. CGAN Architecture

After taking as input the noise and the labels, the baseline CGAN *generator* consists of three transposed convolutional layers, which upsample the input. They have filter sizes 7x7, 5x5 and 5x5 respectively. The first layer is the transpose of a non-zero-padded unit-stride convolutional layer, and the other two layers are the transposes of zero-padded convolutional layers with 2x2 striding; which overall produces 28x28 pictures. The channel sizes are exponentially decreasing from 256. The first two layers encompass batch normalisation, similarly to the DCGAN architecture, before the ReLU activation function. Again, the output is passed through a *tanh* function to bring the values between -1 and 1. The *discriminator* consists of three convolutional layers with filter sizes 5x5, 5x5 and 7x7 respectively and with exponentially increasing number of channels, reaching up to 256. The first two layers are zero padded and use 2x2 striding with a ReLU activation function, which facilitates back-propagation and reduces the likelihood of a vanishing gradient. The last layer's single-valued output is passed through a sigmoid function to determine whether the input image was real (1) or generated by G (0).

7.12. Inception Score

$$IS(x) = exp(\mathbb{E}_x[KL(p(y|x)||p(y)]) \tag{4}$$

The Inception score was created with the goal in mind to *quantify realism*. A human would judge the quality of the generated set based on two main criteria. Based on whether for each separate image it is possible to tell what's on the picture **(saliency)**, and whether the set consist of a wide diversity of samples **(diversity)**. It is calculated as shown in Equation (4), and it should be ideally as high as possible. In the equation $p(y|x)$ represents **saliency** in the formula, and indicates that the class predictions for an image should be confident, and $p(x)$ represents **diversity** and indicates that overall in the generated set, there should be a wide variety of classes, just like in a well-balanced training set [12] (x is a generated image, $p(y|x)$ is the output distribution over the classes given x and $p(y)$ is the marginal distribution over the classes). KL represents the Kullback-Leibler distance, a measure of the difference between the distribution of $p(y|x)$ and $p(y)$. The Inception score fails to be a good estimator for the diversity of the generator, when similar images are created for each class. In this case p(y) will still be uniform even though the diversity is low, which is why the FID score is a better measure than the Inception score. Example failure and success cases are presented on Figure 26. The image shows that the misclassified "2"s are slightly of worse quality than the correctly classified ones. Some of them might be even misclassified by a human observer after just a quick look.

Figure 26: Example failure and success cases of the LeNet trying to classify pictures of generated digit "2"s

7.13. Training results of the extended CGAN

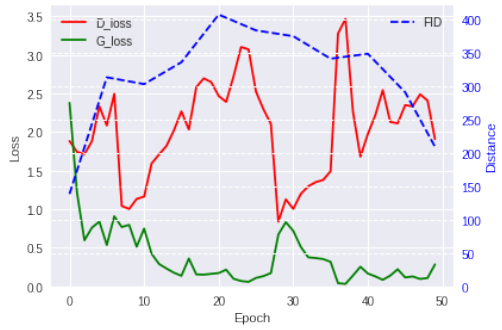

Figure 27: Training results of the CGAN with two extra convolutional layers in the generator and in the discriminator

Figure 28: The Inception score and the recognition accuracy of the classifier LeNet for each epoch during training of the extended CGAN

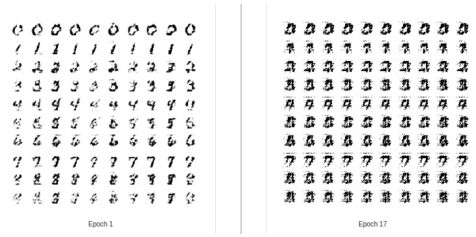

Figure 29: It is easy to observe how the generator overpowering the discriminator can lead to quickly deteriorating image quality (extended CGAN architecture)

7.14. Training results of the classical CGAN

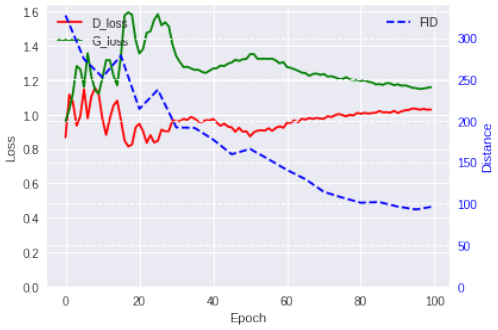

Figure 30: Training results of the CGAN with dense layers

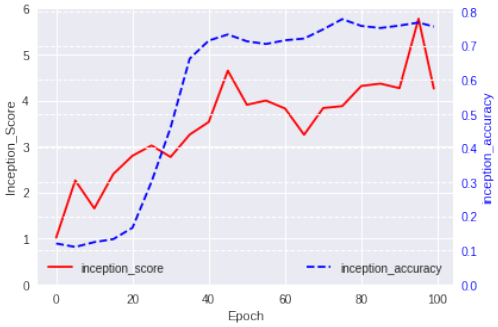

Figure 31: The Inception score and the recognition accuracy of the classifier LeNet for each epoch during training of the dense-layered CGAN

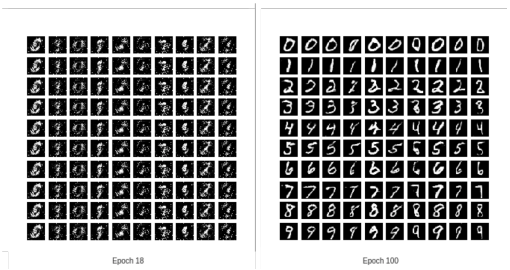

Figure 32: The improvement of the image quality of the dense-layered CGAN is improving slowly but significantly with training

7.15. Figure for one-sided label smoothing

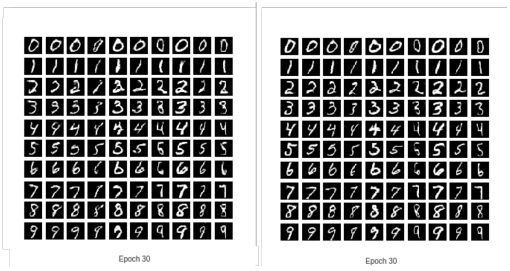

Figure 33: Examples of pictures generated by the CGAN architecture with one-sided label smoothing

7.16. Figures for modified label concatenation

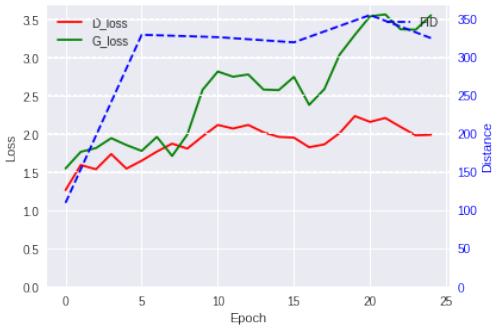

Figure 34: Training results of the CGAN with the labels fed through an input convolutional layer

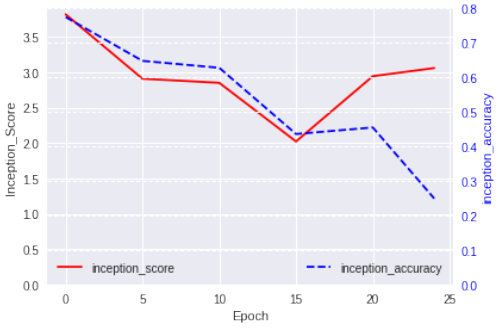

Figure 35: Inception results of the CGAN with the labels fed through an input convolutional layer

7.17. Summary of the CGANs' results

It is interesting to note that as it has been described in Section 7.3, the best FID and second best Inception score corresponds to the architecture with the most intra-class diverse samples, the baseline architecture with one-sided label smoothing (see Figure 33 for example images.

Architecture	Best IS	Best FID	Best Inception Accuracy
Baseline	4.75	52	84%
Extended	4.21	154	68%
G-D Balance.	4.34	187	36%
Dense CGAN	5.20	97	78%
W/O Batch N.	4.71	65	73%
One-sided LS	4.98	17	87%
New Concat	3.52	101	60%

Table 2: Numerical evaluation results of the various CGAN architectures

7.18. Data Augmentation

For training, the existing dataset is expanded with generated samples whose labels are known. This can improve generalisation capabilities and be helpful even for very large datasets [1, 10]. While traditional data augmentation methods rely on randomly selected transformation and perturbation operations (i.e. translation, rotation, adding noise, interpolation) to create new data samples, GANs have recently become an attractive alternative due to their possible broader augmentation policy [1].

7.19. Figures Re-training the handwritten digit classifier

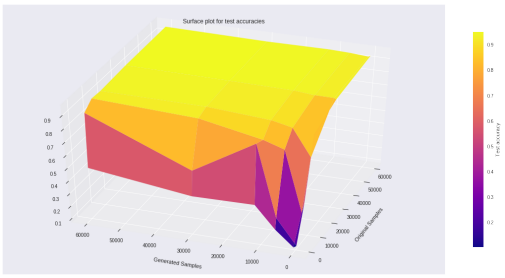

Figure 36: CDCGAN baseline: Surface Plot for test recognition accuracy over differently combined training data

As another experiment with Data Augmentation it was also tested whether the classifier network would benefit from receiving information on whether the data was created or not. This was done by increasing the input depth of the first input filter to two and concatenating a 32x32x1 array of 1 (real) or -1 (fake) to the channel dimension of each image. The highest test accuracy was obtained with 94.25% (60 000 original, 30 000 generated) which is similar to the only-original samples case. From Fig. 42 and Fig. 43, it seems like the network was encouraged to treat the two types of images very differently instead of benefiting from the increased generalisation capability. Especially in the case with only few original samples, the test accuracy

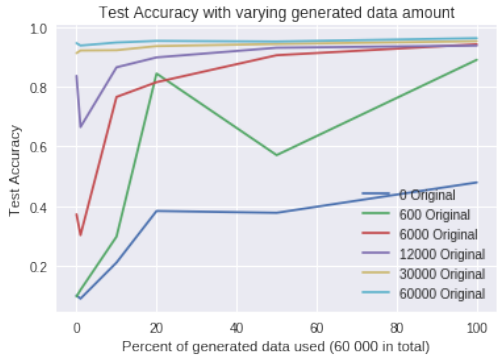

Figure 37: CDCGAN baseline: Plot showing the effect on adding different amounts of generated data to a original training set

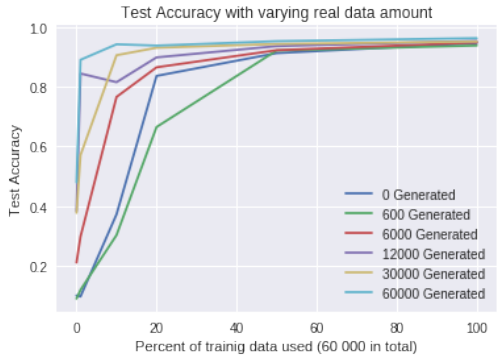

Figure 38: CDCGAN baseline: Plot showing the effect on adding different amounts of original data to a generated training dataset

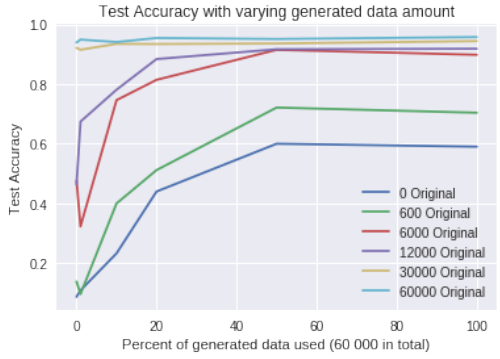

Figure 39: CDCGAN baseline with quality check: Plot showing the effect on adding different amounts of generated data to a original training set

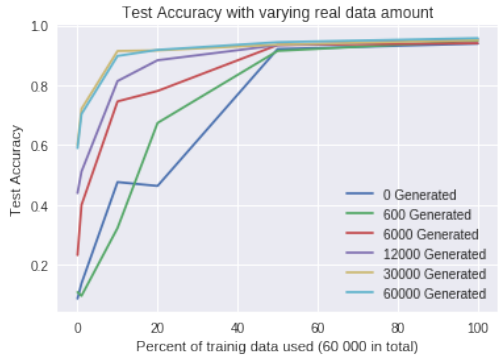

Figure 40: CDCGAN baseline with quality check: Plot showing the effect on adding different amounts of original data to a generated training dataset

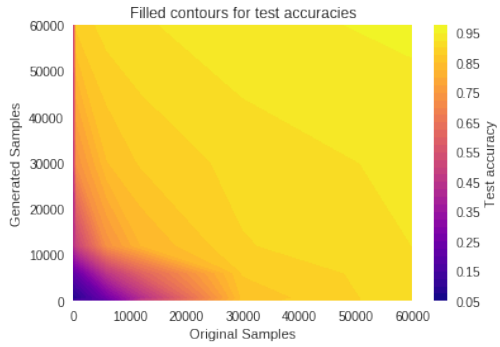

Figure 41: CDCGAN baseline with fine tuning: Contour Plot for test recognition accuracy over differently combined training data

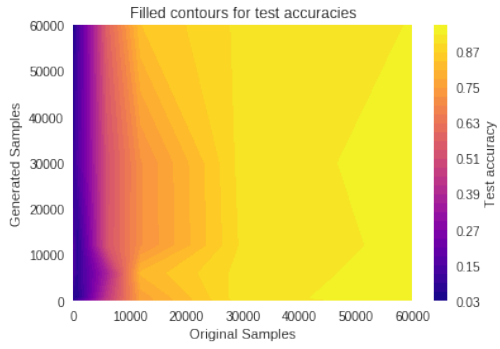

Figure 42: CDCGAN baseline with source information for LeNet: Contour Plot for test recognition accuracy over differently combined training data

dropped significantly pointing to the fact that the network opposed to learn from the high amount of generated digits to classify the real labelled test images. The responses for different amounts of generated samples with varying original training set size also became much closer proving that the artificial images were kind of ignored (see Fig. 44, Appendix 7.19).

Since smaller hyperparameter changes to the baseline CDCGAN did not lead to different results, it was decided to experiment with the classic CGAN to have some comparison in terms of architecture. The results (see Fig. 45, Appendix 7.19) for the CGAN created images without quality check show a similar trend to the baseline CDCGAN with only slightly less boost when training with lots of original

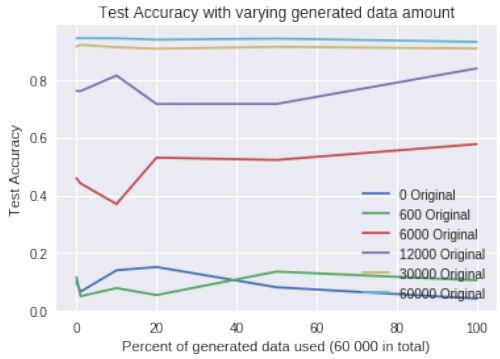

Figure 43: CDCGAN baseline with source information for LeNet: Plot showing the effect on adding different amounts of generated data to a original training set

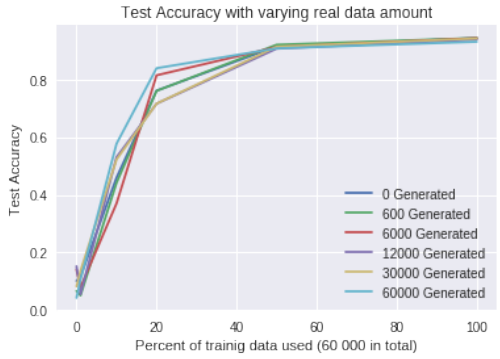

Figure 44: CDCGAN baseline with source information for LeNet: Plot showing the effect on adding different amounts of original data to a generated training dataset

images (see 46, Appendix 7.19). This shows that the classical CGAN also learnt a lot about the underlying training distribution but possibly was slightly worse in generalising to new examples.

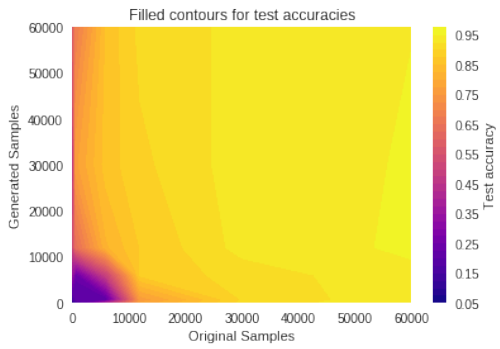

Figure 45: Classic CGAN: Contour Plot for test recognition accuracy over differently combined training data

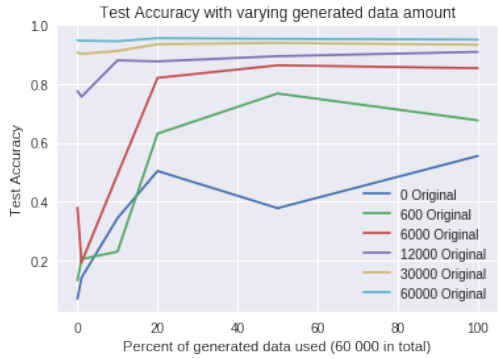

Figure 46: Classic CGAN: Plot showing the effect on adding different amounts of generated data to a original training set

References

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

[2] A. Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[3] J. Brownlee. Gentle introduction to the adam optimization algorithm for deep learning, Jul 2017.

[4] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] H. Kang. tensorflow-mnist-cgan-cdcgan, Aug 2017.

[7] H. Kang. tensorflow-mnist-gan-dcgan, Aug 2017.

[8] T.-K. Kim. Generative adversarial networks (gans), Feb 2018.

[9] F. Peccia. Batch normalization: theory and how to use it with tensorflow. Sep, 2018.

[10] A. Romero, N. Ballas, S. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Imagenet classification with deep convolutional neural networks. In *International Conference on Learning Representations*, 2015.

[11] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *null*, page 958. IEEE, 2003.

[12] R. Trusov. Inception score evaluating the realism of your gan. May 2018.