EE4-62: Selected Topics in Computer Vision - Coursework 2

Jorge Sanmiguel - CID: 01064565 Ricardo Vera - CID: 01061547
Electrical and Electronic Engineering Department
Imperial College London

1. Introduction

This coursework explores convolutional GANs, specifically the DCGAN and cGAN architectures, that will be used to learn the data distribution of a commonly used dataset, MNIST [1]. The generated synthetic handwritten digit images will be then used as a means of dataset augmentation to train a classifier, and the performance improvement will be reported via the inception score obtained by the classifier.

2. Generative Adversarial Networks - GANs

Generative adversarial networks (GANs) describe a framework introduced by Goodfellow et al. [2], in which two networks, a generator and a discriminator, *'play'* a minimax game. The generator is trained to learn and generate data samples that resemble those of a training distribution. The discriminator is in charge of determining if a data sample is real or has been forged by the generator.

Overall, the framework is described by the loss function in Equation 1: the discriminator tries to maximise its accuracy and hence the loss function (both terms), whereas the generator tries to minimise it by fooling the discriminator (second term).

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

Each network is trained individually in an alternate pattern to avoid one network overpowering the other and halting the training process at an early stage (e.g. gradient collapse). To force the generator of creating a wide range of different samples, it is fed noise at its input, which ultimately affects the network's output.

3. DCGAN

The first GAN explored for the task of image generation is a Deep Convolutional GAN (DCGAN), which consist on the generator and discriminator architectures being implemented with convolutional layers.

Proposed Architectures

Choosing the depths of the networks is a trade-off between their capacity to develop complex data representations and their complexity: a deeper network will be more capable of learning data distributions, but is prone to overfitting and a larger dataset will be required to tweak the much larger number of parameters in the network. The smallest network that maximises the overall maximum performance is desired.

Two different architectures based on the original DCGAN paper by Radford et al. [3] are designed and implemented with Keras using Tensorflow as backend. The first GAN is composed of shallow networks consisting of three convolutional layers (both generator and discriminator), whereas the second is deeper and has four convolutional layers each.

The **generator** of the first proposed implementation, shown in Figure 1, takes as an input a random, normally sampled, noise vector of size 100, which gets projected and reshaped into a $7 \times 7 \times 128$ tensor. Two transposed convolutions with stride 2×2 follow that up-sample the image to the desired resolution of 28×28 pixels. Transposed convolutions are used to avoid sparse gradients and help with stability during training [4]. Last, a normal convolution reduces the number of filters to one, hence obtaining the output image.

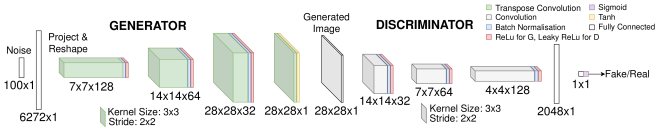


Figure 1: DCGAN architecture

The **discriminator**, also shown in Figure 1, is symmetric to the generator, the 28×28 input gets halved at each layer, in this case using normal convolutions with stride 2×2. Max-pooling was avoided to allow the spatial downsampling to be learnt, as proposed by Springenberg et al. in *The All Convolutional Net* [5]. The $4 \times 4 \times 128$ output of the last convolutional layer is flattened into a 2048 tensor and mapped to a unique output using a fully connected layer. The validity of the image is determined by the sigmoid function.

The **number of filters** used in the generator have been chosen to half at each layer, whereas they double in the discriminator architecture, following the proposed architectures in [3]. All **kernels** used are of size 3×3 to help reduce the training time of the network, and are expected to work well given the relatively small size of the largest layers. This design choice has been made following the popular VGG16 architecture [6].

Each layer of the generator and discriminator uses **ReLU** and **leaky ReLU activations** as proposed in [3], except for the last ones which uses a **tanh** and **sigmoid** activation: The output image by the generator is made to have pixels in the range [-1, 1] to match the normalised MNIST dataset used, whereas the output of the discriminator is desired to be between 0 and 1. This choice of activation functions is important to avoid sparse gradients and help with training stability [4].

After each activation layer, **batch normalisation** is introduced to reduce the amount by which the hidden units shift, to help the gradient flow into deeper layers, thus preventing vanishing gradient, and to introduce some independence between the layers [4]. Even though batch normalisation is used for stability, it can lead to model oscillation and instability as reported in [3], hence it is not applied at the output of the generator.

The second proposed architecture has one extra layer in both the generator and discriminator, with no strides to not alter the overall size of the other layers. See Figure 13 in Appendix. All other parameters remain unchanged.

Binary cross-entropy loss is used as loss for all implementations due to its resemblance to Equation 1. Additionally,

the kernel initialiser used was the popular Xavier uniform [7] which aims at matching input and output variance.

Training

To train the networks, the discriminator is trained on a set of real data, and on a set of fake data from the generator, without mixing these sets as proposed in [4]. For the generator, the loss function is not directly implemented as in Equation 1. As explained in [2], initially the discriminator can get extremely good at discriminating the poorly made samples of the generator, and will results in gradient collapse. To train the generator, the entire structure (generator plus (locked) discriminator) is trained with flipped labels, hence maximising $\log D(G(z))$.

In order to maximise the performance of the DCGANs, a range of hyperparameters have been explored for both networks. To avoid mode collapse, k, which defines how many times more the discriminator is trained with respect to the generator (can be seen as *balancing* G and D), was set to 1, 2, and 3. By keeping the discriminator well trained, it will be able to distinguish if the generator is generating the same, fake image all the time. On the other hand, increasing k results in increased training complexity, and hence it is kept small.

The choice of optimiser is another vital element for fast convergence of the networks during training, so two where tested *SGD* with learning rates $0.0001, 0.00001$, and *Adam* with $\beta_1 : 0.5, 0.9$. The choice of these betas, which controls the decay of first moment estimates and affects the optimiser's ability to determine the next step, follows from what is recommended in [3] (former) and the default used by Adam (latter) [8]. The search was not made finer, since training the two different proposed structures with more parameters would become prohibitive. Nevertheless, the proposed search allows to check all mayor different possibilities whilst keeping training times reasonable. Networks were trained to 15 epochs each.

Results

Measuring the performance of GANs is difficult in general, as the losses of the networks do not generally converge, since the networks are always trying to improve over the other one. It was determined best to pick the network that worked best with a visual inspection of the generated, fake images.

Interestingly, both architectures proposed worked best when using the Adam optimiser with $\beta_1 = 0.5$, and with k set to 2. Figures 14a & 14b (Appendix) show a comparison of the generated images by all of the parameter combinations. The two best performing networks configurations where further trained to **30 Epochs**; 10 randomly generated digits by each optimal architecture are shown in Figure 2, together with examples of the original dataset.

Balancing G and D with $k = 2$ did help the generator produce realistic results, as in all cases where $k = 1$ the generator did only output noise, see Figure 14 (Appendix). By keeping the discriminator well trained, the generator is forced to output realistic results. Increasing k further did not show increased performance, hence it was deemed best to keep it small due to complexity requirements.

The losses of both optimal architectures are shown in Figure 3. It can be seen that the losses vary much throughout the training process, thus can not be used to determine the quality of the network. In the case of the first architecture it can be seen how initially the discriminator could perfectly distinguish the real/fake images for the first 5 epochs, but the generator did eventually catch up.

Figure 2: 10 train digit examples (top), plus 10 random digits generated by the shallow architecture (middle) and deep architecture (bottom). $k = 2$, Adam optimiser, $\beta_1 = 0.5$ and trained 30 epochs.

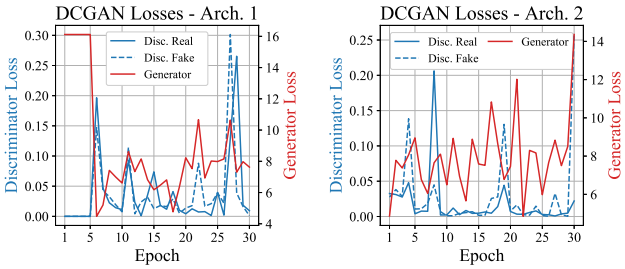


Figure 3: DCGAN losses for both architectures

The choice of optimiser was crucial, as SGD failed to generate any results at all in any combination of hyperparameters, Figure 14a. This was characterised in the discriminator loss as it went to 0 rapidly. Adam showed its superiority, and allowed results in most of the configurations. The choice of β_1 was important, and optimal results where found when set to 0.5, just as suggested in [3]. Decreasing the momentum of the optimiser allows the overall network stability to increase.

Overall, both proposed architectures yield similar results, with realistic numbers generated most of the time. Over a 100 random sample of fake images, the distribution of each network was manually noted, shown in Figure 15 (Appendix), and the second architecture turned winner by generating around 3% less of indecipherable images. This shows that the deeper network allowed an increase learning capability in the same number of epochs as the shallow network. It must be also noted, that both networks generated a fairly uniform distribution for all classes, without suffering from mode collapse, see Figure 16 (Appendix).

4. cGAN

Conditional GANs (cGAN) are a type of GAN that condition the output of the networks based on some input. Building on the implementation of the DCGAN, the cGAN generator gets input noise plus a label at each time as the condition. The discriminator is then fed an image and a class label, and determines the veracity of the image conditioned on the label. There is the need of extra layers in the networks to account for these extra inputs. The loss function changes to accommodate the conditional inputs, Equation 2.

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

Proposed Architecture & Training

Two different implementations are again proposed to explore the effect of the network's architecture on the desired task: a shallow and a deep one, both following a convolutional structure. Both networks are based on the previous proposed DCGAN architectures, but with fully connected layers introduced on the layers directly after the label inputs. The two

branches of the two inputs (noise and label for G, image and label for D) are merged by concatenation in an early stage, and are followed by convolutional layers as in the DCGAN architectures, see Figure 4 (shallow architecture, Figure 18 in Appendix for the deep structure).

The **generator** of the first proposed implementation, Figure 4, takes as an input a random, normally sampled, noise vector of size 100 and a label vector size 10 (one-hot encoding). Each of which is follows by a fully connected layer to expand the number of nodes, and is then projected and reshaped into a $14 \times 14 \times 128$ tensor. Both are merged by concatenation, which is followed by a transposed convolution with stride 2×2 to upsample the image to the desired resolution of 28×28 pixels. As with the DCGAN, transposed convolutions are used to avoid sparse gradients and help with stability during training [4]. Last, a normal convolution reduces the number of filters to one, obtaining the output image.

The **discriminator**, also shown in Figure 4, follows the same steps, the 28×28 input gets halved in the first layer, in this case using normal convolutions with stride 2×2, whilst the input labels pass through a fully connected layer and get projected and reshaped to match the tensor shape. As with the DCGAN, maxpooling was avoided as proposed by Springenberg et al.[5]. The two branches are then merged, and a final convolutional layer halves the image size to $7 \times 7 \times 128$. This gets flattened and passed through a fully connected layer to output a single output.

The second implementation has an additional convolutional layer after the merge, in both the generator and the discriminator. This requires slight adjustments to the final layers of each network to accommodate the correct output sizes, see Figure 18 in the Appendix for all details.

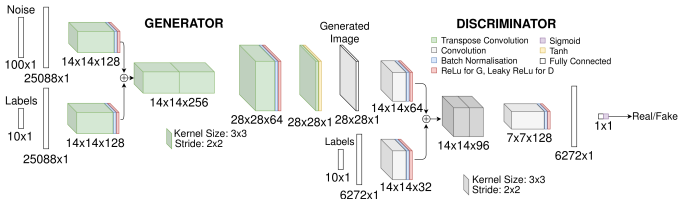


Figure 4: cGAN architecture

All other structural parameters have been copied from the DCGAN architectures following the same rationale to address possible challenges: the **number of filters** in the generator/discriminator half/double following Radford et al. in [3], **small kernels** of size 3×3 are used to have a low complexity, **ReLU and leaky ReLU** for G and D respectively are used as activation to avoid sparse gradients, with the exception of the output layers which use **tanh** and **sigmoid** respectively. **Batch normalisation** is again used to also help with the training gradients and to avoid oscillation of the system.

For training, only the *Adam* optimiser was used, as it clearly outperforms the *SGD* optimiser (seen from the results in the DCGAN results). It was again tested with two settings $\beta_1 : 0.5, 0.9$. k was varied in the range $1, 2, 3$.

Results

Similar to the DCGAN case, the losses of the networks do not converge and cannot be used to determine the overall performance of the networks, see Figure 5. The best results were determined by image quality, Figures 19a & 19b in Appendix show the outputs by all the configurations. The shallow network generates really good outputs, with only some slight distortions (see number 6), or with some random strokes around the generated numbers (see numbers 8 and 9). The deep structure generates perfect images, with almost no signs of being fake. The best results were visually determined to be for $k = 2$ and $\beta_1 = 0.9$ for both architectures, but by a thin margin. The generated digits with these two architectures after 30 epochs of training are shown in Figure 6. The cGAN networks benefited of a higher momentum employed by the Adam optimiser, which can be the result of the networks being larger than in the DCGAN case (extra inputs).

The shallow implementation did suffer from mode collapse for $k = 1$ (any β_1) or $k = 2$ and $\beta_1 = 0.5$. This was tested by generating many different digits of the same class, Figure 20 in Appendix. k again proved keeping a strong discriminator is key during training, as the use of a larger k manages to overcome mode collapse by providing a more robust discriminator that can learn if the same image is always being generated, forcing the generator to vary its outputs. Figure 20 shows also how the optimal deep structure example does not suffer from mode collapse. It's likely that the extra depth of the discriminator provided it with the ability to discriminate better the digits and thus evolve appropriately during training.

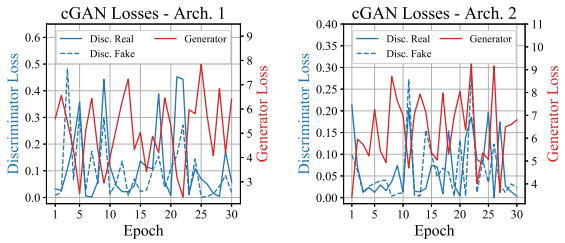


Figure 5: cGAN losses for both architectures $k = 2$, $\beta_1 = 0.9$

Figure 6: 10 random digits (one per class) generated by the shallow architecture (top) and deep architecture (bottom). Adam optimiser, $\beta_1 = 0.9$, $k = 2$ and trained 30 epochs.

One-sided label smoothing

One-sided label smoothing helps deal with overconfidence of the discriminator and mode collapse, when the discriminator bases its decision on very few features from the input images, which the generator eventually learns to generate, resulting in no long term benefit. Smoothening the labels helps penalise the discriminator to prevent this, however setting $0 \rightarrow 0.1$ gives the model no incentive to move nearer to the data and so only $1 \rightarrow 0.9$ is smoothed.

Figure 7: 10 random digits (one per class) generated by the shallow smoothened architecture (top) and deep smoothened architecture (bottom). Adam optimiser, $k = 2$ $\beta_1 = 0.9$ and trained 30 epochs.

One-sided label smoothing was tested on the best implementations (deep and shallow) and on $k = 1$, $\beta_1 = 0.9$ for the shallow network that suffered from mode collapse. Results showed that mode collapse was not avoided, Figure 21 (Appendix). The other networks produced images of similar quality as without label smoothing, Figure 7.

5. Inception Score

Loss measures provide a way of evaluating how well the discriminator is distinguishing fake from real, or how well the generator is managing to *fool* the discriminator, however provide little information on the qualtiy of the generated images. The inception score provides a mean to measure this quality. The inception score, as defined (Equation 3), measures image quality using both the quality of the generated images and their diversity, thus measuring individual quality and the data distribution accuracy.

$$\mathrm{IS}(G) = \exp(\mathbb{E}_{\mathbf{x}\sim p_g} D_{KL}(p(y|\mathbf{x}) \,||\, p(y))) \tag{3}$$

Where D_{KL} refers to the KL-divergence. In the case being reported, the problem is much simpler: to evaluate the cGAN's performance, since the output images can be controlled via the input labels, the need to control diversity among the generated images is not required. To measure the inception score only a classification network is required, to compare its classification accuracy of the generated data against the original data.

The MNIST dataset is a relatively easy dataset so a shallow architecture is enough: a deep Convolutional Network, composed of **2 convolutional layers** and **2 fully connected layers** was developed. Each convolutional layer halves the input image size (**stride** 2×2), **doubles the number of filters** (starting from 32) and uses **kernels of size** 3×3. The **fully connected layers** have **50** and **10** neurons each. All layers use **ReLU activation**, except for the last layer that uses **softmax** for classification. See Figure 22 (Appendix). The network is trained on the MNIST training data (60000 images), and then tested on the MNIST test data (10000 images) as well as on 10000 generated digits for each generative model.

The training accuracy of the network with the real data was 100% with a 100% confidence in its classifications. See Figure 23 (Appendix) for the training loss. The test accuracy was 98.99% with a 99.72%/80.25% confidence in its correct/incorrect classifications. The network is really good at classifying the dataset, and when it fails, it does it with a lower confidence. All the different implementations were tested, which yielded similar accuracies and confidences to the real data, shown in Table 1.

k	β_1	Deep			Shallow		
		Acc.	Conf. C	Conf. I	Acc.	Conf. C	Conf. I
1	0.5	99.02%	99.62%	81.25%	74.02%	97.81%	91.83%
	0.9	99.25%	99.76%	82.08%	73.40%	96.41%	87.21%
2	0.5	98.83%	99.66%	83.77%	98.22%	94.29%	58.51%
	0.9	**99.34%**	99.82%	81.82%	98.90%	99.64%	80.41%
3	0.5	98.78%	99.70%	83.61%	98.59%	99.60%	81.56%
	0.9	99.27%	99.78%	85.34%	98.15%	99.51%	82.41%

Table 1: Classification accuracies and confidences for varying implementations. Conf. C/I represent confidences when correctly/incorrectly classified.

Most implementations manage to emulate the real data accuracy, with the deeper network performing better than the shallow one throughout all parameter combinations. The highest accuracy resulted from the deeper network with $k = 2$ and $\beta_1 = 0.9$. The confusion matrix from the mentioned best performing parameters together with that of the test data is shown in Figure 8. It can be seen how both are extremely similar, with very few errors that are scattered throughout all classes combinations.

The networks that suffer from mode collapse (shallow, $k = 1$) yield the worst accuracies. Interestingly, some generated datasets perform better than the real test data. This can be explained since the GANs are generating digits from a distribution learnt from the training data, which is the same the classification network is trained with. Hence the generated images by the GAN are closer in resemblance to the training set than the test set.

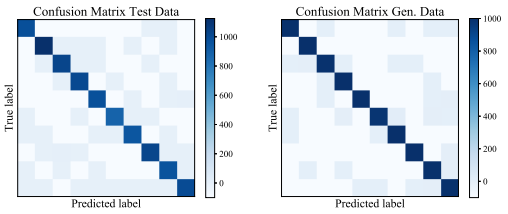


Figure 8: Conf. matrices for real/fake data ($k = 2, \beta_1 = 0.9$, deep)

Table 2 shows how one-sided label smoothing deteriorates performance of the best implementations, whilst improves that of the one suffering from mode collapse. Smoothing offers little benefit for already good networks. For networks with mode collapse it encourages the discriminator to not rely on few features, and hence forces the generator make more realistic outputs. However, as shown, it is not a guarantee that it will avoid mode collapse.

Architecture	k	β_1	Acc	Conf. C	Conf. I
Shallow	1	0.9	79.88%	97.82%	99.61%
	2		98.44%	99.56%	84.67%
Deep	2		98.60%	97.82%	83.61%

Table 2: Accuracies and confidences when using label smoothing.

6. Retraining the handwritten classifier

The inception score provides a way of evaluating the quality of the generated images, however the difference in accuracy may be due to the actual difficulty of classification: the randomness in digit generation may simply generate easier/harder to classify digits than the original test set. By training the classifier with a mix of generated and real images and reporting the test accuracy, the quality of the augmented dataset can be assessed.

Figure 9 shows the test accuracy when using the classifier trained with varying real:fake images ratio (total number of samples fixed). Three generators have been used for this task with respect to the inception performance: the worst one (that did not suffer mode collapse): shallow cGAN $k = 3$, $\beta_1 = 0.9$, the one with an inception score similar to the real testing accuracy: deep cGAN $k = 1$, β_1=0.5, and the best performing network: deep cGAN $k = 2$, β_1 = 0.9. Figure 9 also shows the test accuracy when the classifier is only trained with the proportion of training data (e.g. only 10% of training data).

The results show a clear trend: the lower the real data proportion, the worse the accuracy. None of the tested ratios allow the classifier to exceed the testing accuracy when only using the training data. The first network performs the worst, with a strong accuracy decline after 3:7 data split ratio. The best observed performance without using any real data corresponds to the network with highest inception score, and is 96.70%, still much lower than using the training data only. Nevertheless, the mixed set accuracies can improve those of the classifier when only trained with the **comparative proportion** of training data: e.g. when the classifier is trained with 20% of tr. data plus 80 % of generated data an accuracy of 98.25% is obtained, which is higher than the accuracy of the trained classifier with

only 20% of training data (98.09%). Hence it is proven that using dataset augmentation can be useful when the training data set is particularly small.

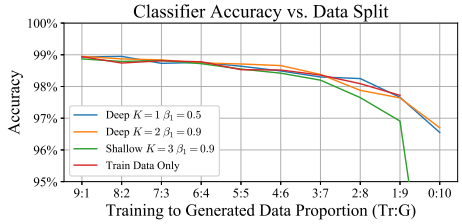


Figure 9: Classif. accuracy for different real to generated data proportions. Figure 24 (Appendix) for whole view.

Figure 25 (Appendix) shows the confidence levels of the classifier for each data split proportion. The classifier is much more confident in its correct classifications than in its incorrect ones. The classifier is however more confident in its incorrect decisions in the cases where there is data augmentation than in the case in which only the comparative proportion of training data is used. This might suggest that the augmented datasets contain images that can easily be confused as different classes.

Another approach was also taken at using the generated data: instead of keeping the total training samples constant, adding extra proportions of generated data to the real was explored. Ideally, the extra data given to the classifier will allow it to generalise better in the testing case.

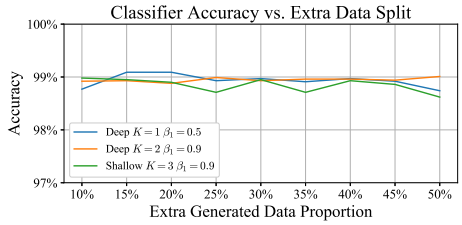


Figure 10: Classif. accuracy for extra gen. training data

Figure 10 shows the effect when adding generated data to the training data at different proportions. See Figure 26 (Appendix) for the classification confidences. The models do not exceed the maximum testing accuracy of 98.99%, and remain largely constant no matter how much generated data is added to the training set. One exception is however that of the deep cGAN $k = 1, \beta_1$ model, that surpasses the 99% accuracy when 15%-20% extra generated data is used. Although marginal, this successfully corroborates that using generated data for dataset augmentation is a valid strategy. Interestingly, the network that achieves this results is not the one that achieves the best inception accuracy, but rather the one that has the most similar inception accuracy as the real test accuracy. That cGAN does not overfit to the training data, and learns how to generalise properly the input dataset distribution.

Last, an extra training procedure was tested, consisting in first training the classifier with the generated data, and then using the real training data for fine tuning. By doing this, the generated data is used by the classifier to learn the general features of the dataset, followed by the real images that allow it to learn the more complex details across the data. By using the complete training data last, any inaccuracies introduced by the generated data will be ideally removed.

Figure 11 shows how the end classification accuracy vary when the classifier is trained first, for varying number of epochs, using the generated data. The results do not show improvement over the real data test accuracy. The training loss when training 15 epochs with the real set and 15 with a generated set is shown in Figure 27 (Appendix). The validation plots of this example show that the classification network really quickly overfits to the generated data, and does not improve until the real data is used. This behaviour can be caused by the generated data not fully capturing the MNIST data distribution, which explains why almost none of the given dataset augmentation examples allow the classifier network to increase its testing accuracy.

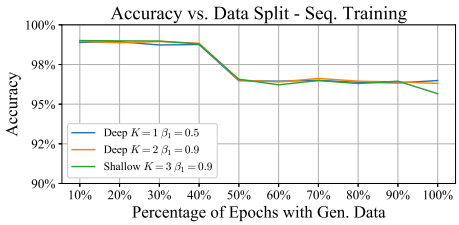


Figure 11: Classifier test accuracy for sequential training. Different percentage of epochs trained with the generated data are shown (out of a total of 30 epochs).

7. Bonus: further developed in Appendix

PCA vs GAN

Principle component analysis (PCA) extracts the components of a data distribution that have the largest possible variance. These components can be seen as the building blocks that form each data sample: the components can be added together using different weighted sums to reconstruct the orginial data from which the components were extracted. It is therefore also possible to use PCA as a generative model if these weights are modified at the users' will.

PCA has a closed form solution, and is much faster to evaluate than having to design and train a GAN. GANs are also usually highly complex, which makes it need much more data to create faithful samples than PCA. PCA is also easier to visualise, whereas it is difficult to understand what a GAN is actually is doing.

The main difference between PCA and GANs is however, that the former one is a linear model, whereas the latter is non-linear. PCA has therefore a much lower ability to generalise from the seen data, whereas GANs can capture the high-dimensional features that are characteristic to the dataset, and then from these create a realistic looking image.

Real vs Fake Visualisation

Inception scores and test accuracies using generated data for training provide means to see how generated data doesn't completely replicate real data, however visually it is hard to see why this difference occurs.

To observe this difference, the embeddings of 1000 real and fake images from a pretrained classifier can be obtained, on which PCA and t-SNE can be performed. By reducing their dimension to the 2 main components, they can be plotted and analysed.

As can be seen from Figure 11 (Appendix), PCA and t-SNE give almost exact plots for real and generated data. This follows from the inception score results obtained, since the accuracy of the generated data is extremely similar to the actual test accuracy. The different classes can be clearly grouped together in the PCA case, and are easily separable in the t-SNE case. In the latter case, it can be easily seen the examples that are prone to be misclassified. Further elaborated in the Appendix.

References

[1] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[2] I. J. Goodfellow et al. *Generative adversarial nets*. 2014. URL: https://arxiv.org/abs/1406.2661.

[3] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. Nov. 2015.

[4] Soumith Chintala et al. *How to Train a GAN? Tips and tricks to make GANs work*. Dec. 2016. URL: https://github.com/soumith/ganhacks.

[5] Jost Tobias Springenberg et al. “Striving for Simplicity: The All Convolutional Net”. In: *CoRR* abs/1412.6806 (2014). arXiv: 1412.6806. URL: http://arxiv.org/abs/1412.6806.

[6] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[7] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[8] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *CoRR* abs/1412.6980 (2014). URL: http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14.

Appendix

Bonus

Real vs Fake Visualisation

Figure 11 shows how the real and generated data are incredibly similar when reduced to 2 dimensions. The PCA plots show how the digits are incredibly similar, although considerably hard to separate in two dimensions, due to large overlap across classes. t-SNE does however show a very clear separation between classes and gives a better visualisation for incorrect classification. As can be observed both the real and fake data have a 9 classified as a 7, or 5s for 6s in the t-SNE plot. The overall similarity reinforces the good quality of the generated images in comparison to the original data.

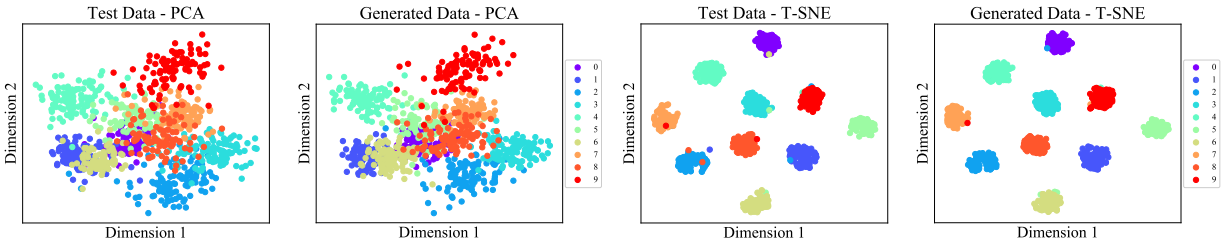


Figure 11: PCA and t-SNE for real and generated images

Figure 11 again shows the similarity in classification confidence for real and generated data. All classes seem similarly confident with no over or underconfident classes. The main difference arises in the classifier’s confidence when incorrectly evaluating, doubting more with real data, whilst showing higher confidence for generated data.

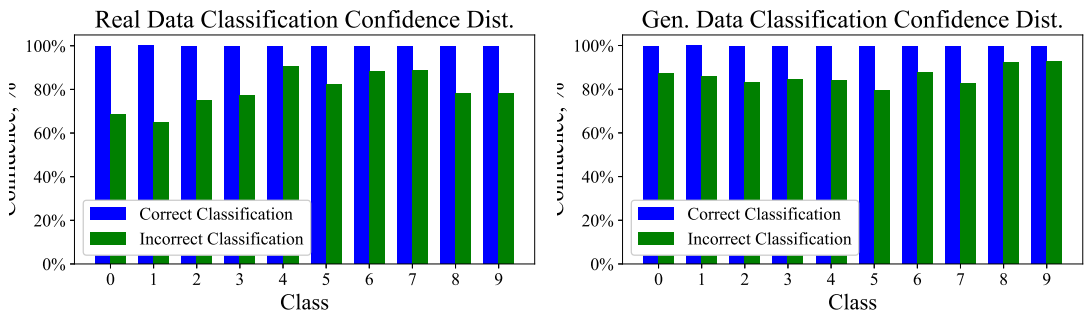


Figure 11: Confidence levels for real and generated digits by class

One Generator/Discriminator multiple Discriminators/Generators

Another possible way of evaluating the performance of GANs is to evaluate the performance of the generator/discriminator with respect to a generator/discriminator of another architecture. By having a fixed generator/discriminator one can evaluate the performance across different generators/discriminators according to the single fixed generator/discriminator, thus providing a good comparison. By forming *new* GANs from the discriminators/generators of previously generated GANs it is possible to see the different strengths and possible imbalances among architectures.

By selecting a generator known to produce good digits, for example $k = 2,\ \beta_1 = 0.9$, one can compare the rest of the discriminators with it to see how the accuracy varies. Table 3 shows how the accuracies vary greatly, the best performing networks in terms of image quality ($k = 2$) perform the worst, whilst discriminators showing mode collapse and those with $k = 3$ have very high accuracies. This to a degree explains the success of the optimum implementation. The GAN is very one-sided towards the discriminator for $k = 1, 3$, making further training necessary for successful digit generation, as the discriminator is still in the early stages being robust enough to avoid being fooled. $k = 2,\ \beta_1 = 0.9$ however has a stable point (~ 0.5) symbolising how the *minmax* behaviour is happening and the generated images are in fact of higher quality.

K	β_1	Accuracy
1	0.5	94.93%
	0.9	69.12%
2	0.5	51.78%
	0.9	43.53%
3	0.5	100%
	0.9	99.94%

Table 3: Accuracy for different discriminators for the same generator ($K = 2, \beta_1 = 0.9$, shallow)

Figures

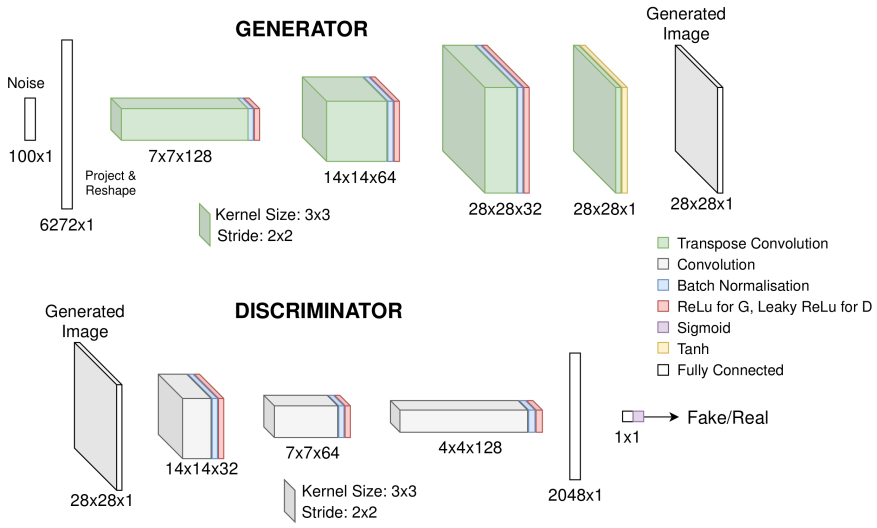


Figure 12: DCGAN shallow architecture (bigger picture for visualisation)

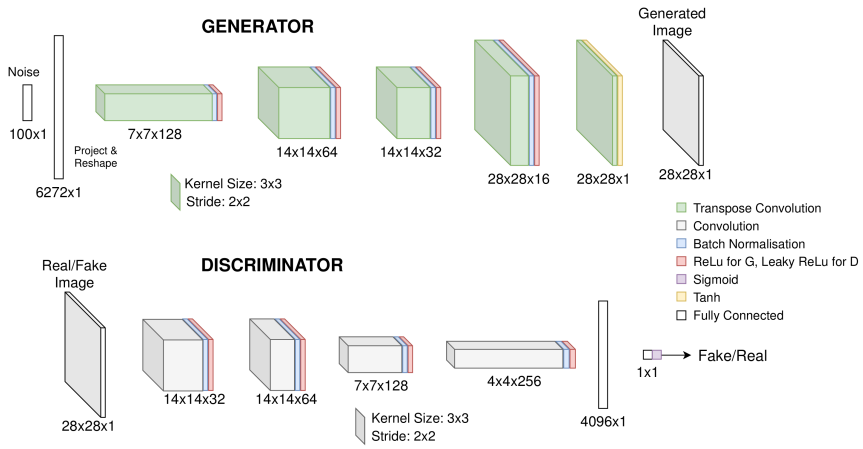
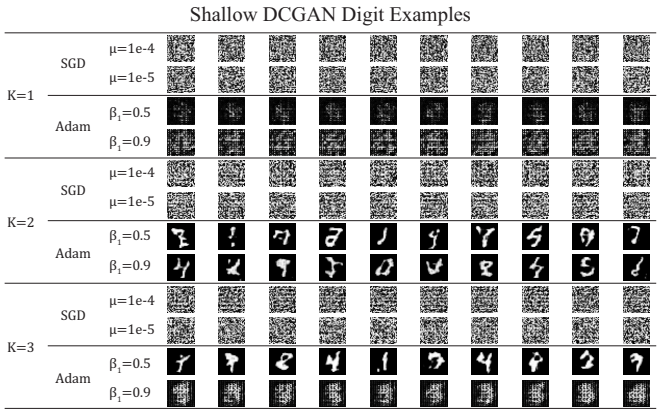
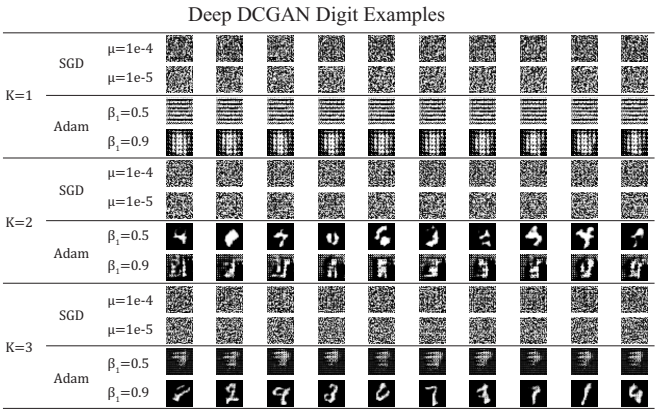


Figure 13: DCGAN deeper architecture



(a) Shallow DCGAN Generated Images for different parameters after 15 epochs



(b) Deep DCGAN Generated Images for different parameters after 15 epochs

Figure 14

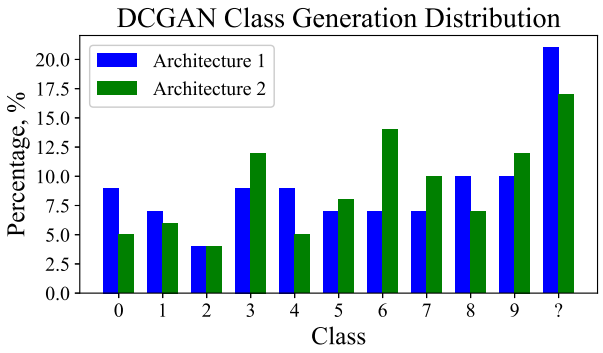


Figure 15: DCGAN class distribution for shallow and deep optimal architectures

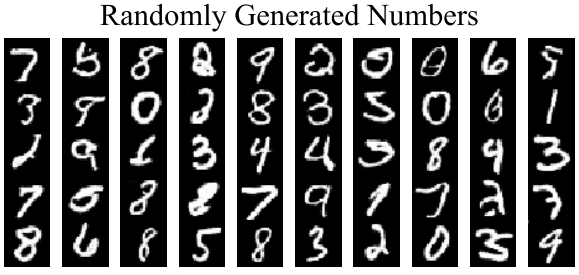


Figure 16: DCGAN example digits, no mode collapse observed

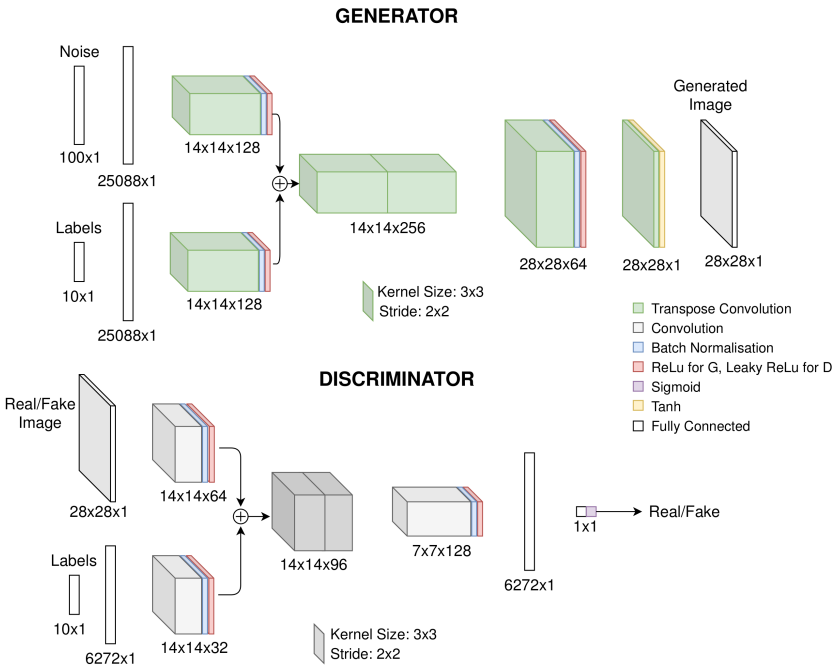


Figure 17: cGAN shallow architecture (bigger picture for visualisation)

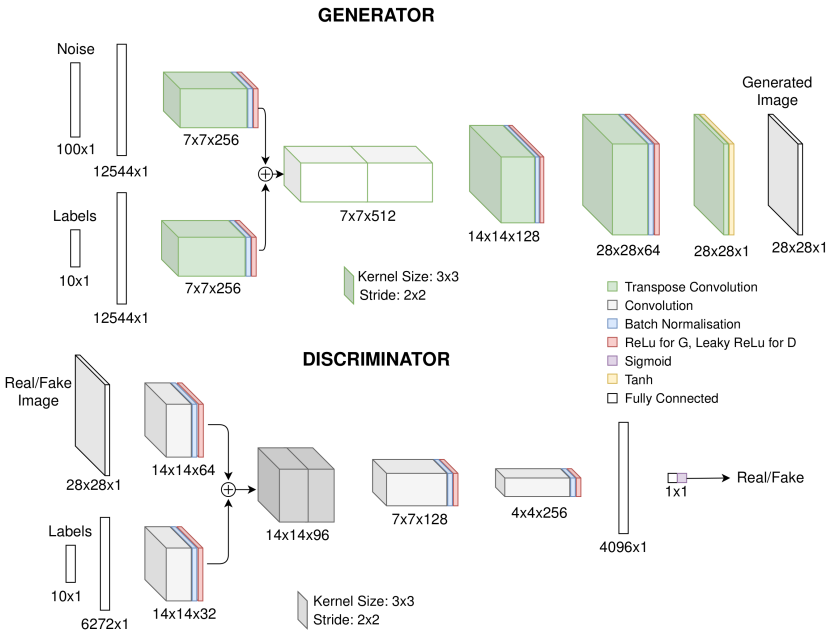
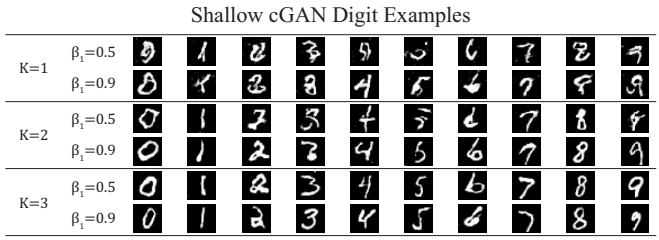
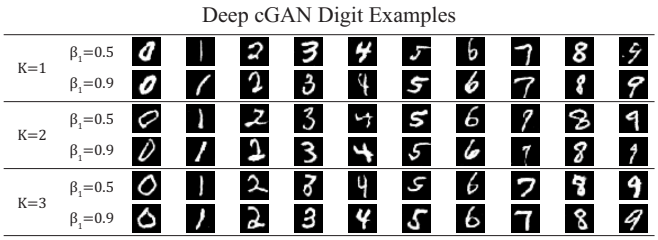


Figure 18: cGAN deep architecture



(a) Shallow cGAN Generated Images for different parameters after 15 epochs



(b) Deep cGAN Generated Images for different parameters after 15 epochs

Figure 19

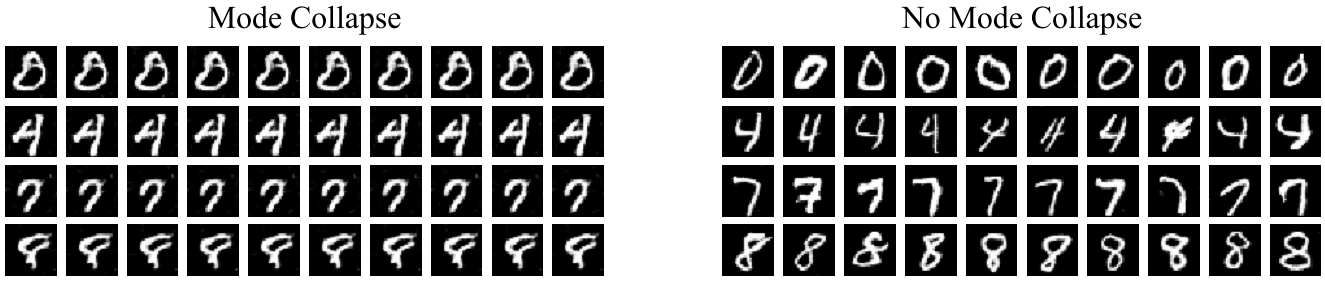


Figure 20: cGAN mode collapse examples (left, $k = 1,\ \beta_1 = 0.9$, first shallow architecture), no mode collapse examples (right, $k = 2,\ \beta_1 = 0.9$, second deeper architecture.

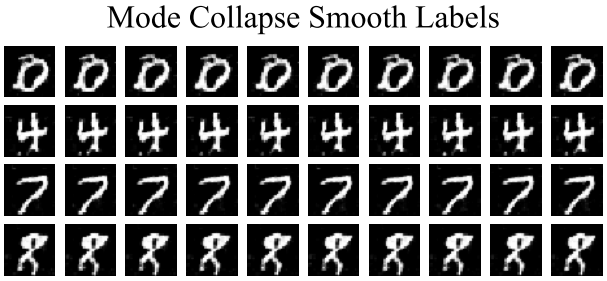


Figure 21: cGAN mode collapse example $k = 1,\ \beta_1 = 0.9$, shallow architecture when trained with one-sided label smoothing.

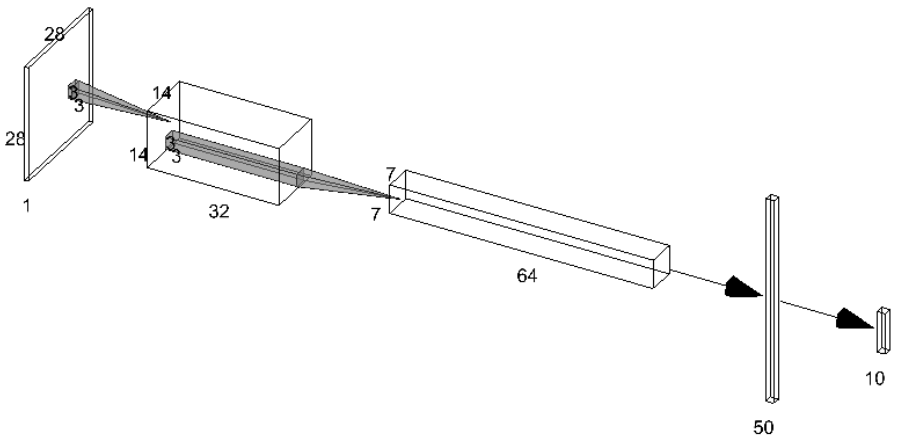


Figure 22: Classification network architecture

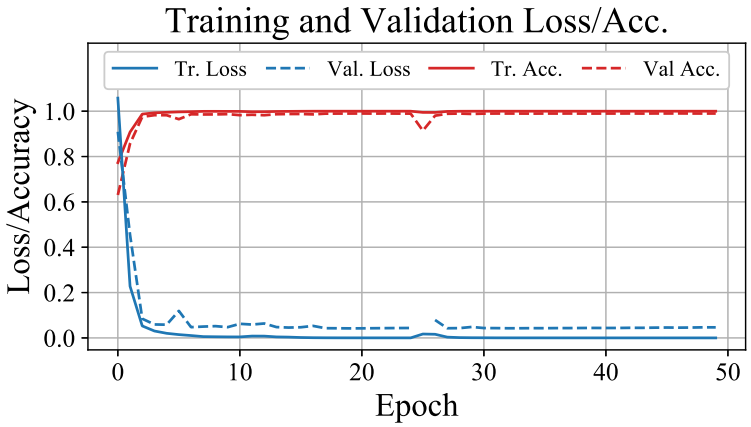


Figure 23: Classifier training loss and accuracy

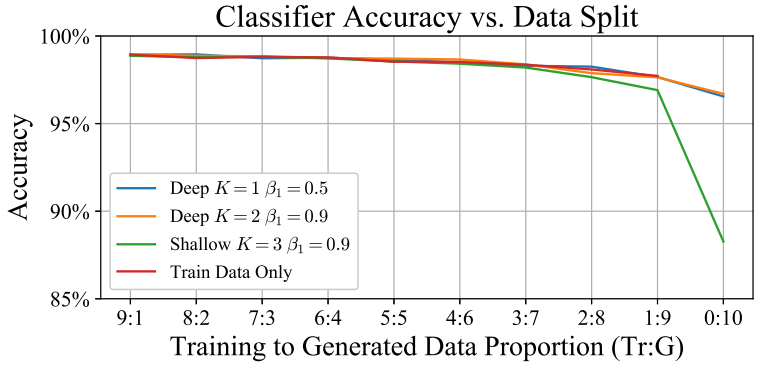


Figure 24: Accuracy for training size fixed

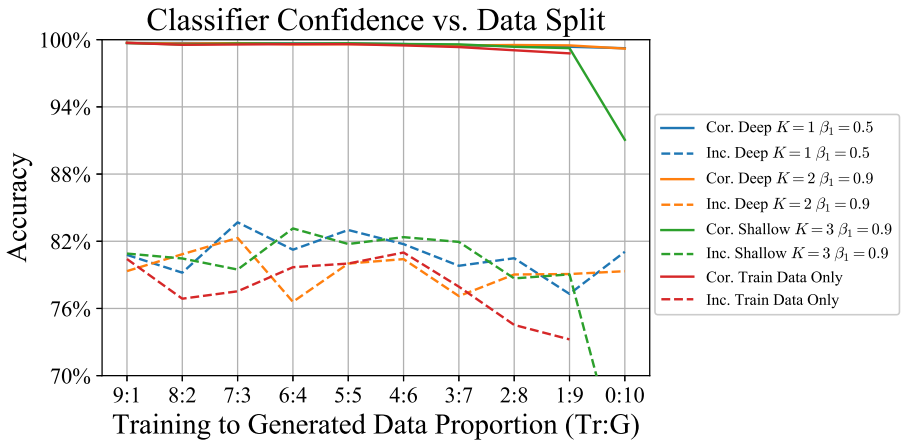


Figure 25: Confidence for training size fixed

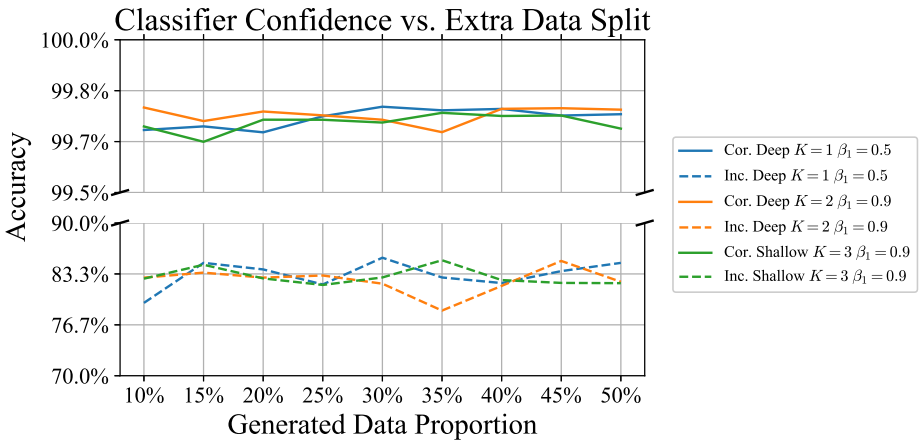


Figure 26: Confidence for training size varying

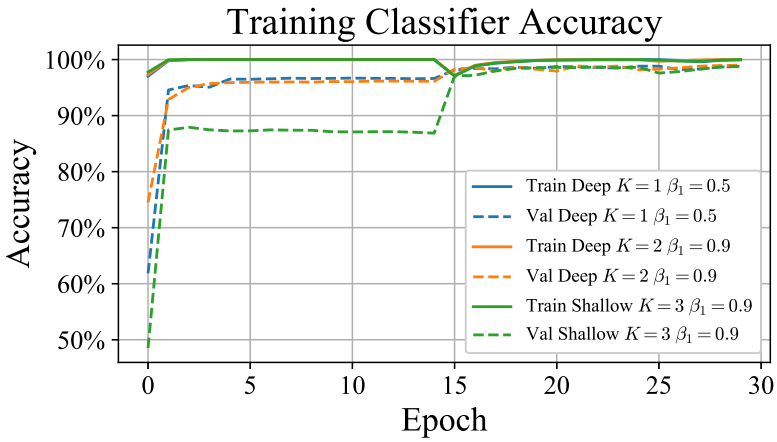


Figure 27: Training and validation accuracy for sequential training of 50% split (epoch 15: fake $\rightarrow$ real data)