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

Alvaro Robledo Vega
Imperial College London
CID: 01076364

Alejandro Gilson Campillo
Imperial College London
CID: 01112712

## 1. Introduction

### 1.1. Dataset description

This experiment uses the MNIST dataset [1], which consists of 70,000 black & white images of handwritten digits (from 0 to 9) of size 28x28. The popularity of this dataset among the data science community stems from its size and simplicity. For this experiment, the dataset is divided into *training*, *validation* and *test* sets of sizes 55,000, 5,000 and 10,000, respectively.

### 1.2. Problem formulation

The goal of this experiment is to design, train and test several Generative Adversarial Network (GAN) [2] architectures to succeed in the task of generating new synthetic digits. The efficacy of each of the models is evaluated using the train and test binary cross-entropy loss, accuracy, and the inception score of a classifier, whenever reasonable.

GANs are a type of generative models that consist of two neural networks (a *generator* and a *discriminator*) that compete against each other. The *generator* is given random noise as an input and generates a new fake output. The *discriminator* is given both real and fake data and outputs whether each of them is real or fake. These results are then compared with the ground truth and the loss function is used to update the weights of both models.

## 2. Deep Convolutional Generative Adversarial Networks (DCGAN)

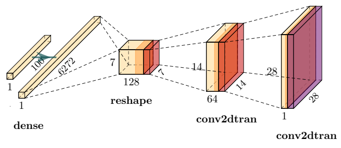
The use of GANs in unsupervised learning was first discussed in a feature extractor called the DCGAN [3]. This network is mainly composed of convolution layers, without any max-pooling or fully-connected layers. These are instead substituted by convolutional stride for the purpose of down-sampling, and transposed convolutions for the purpose of up-sampling.

[3] proposes a specific implementation for a DCGAN. We have taken it as a starting point, and adapted it slightly to the purposes of this experiment and dataset, ensuring that the generated images are of size 28x28, as required.

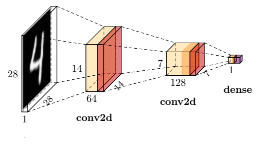
We propose 2 different architectures: a simple one (Shallow DCGAN), and a more complex one (Deep DCGAN). Figures 1 and 3 contain representations of the architectures for both networks.

### 2.1. Shallow DCGAN

Our Shallow DCGAN is characterized by its small number of convolutional layers, which aim to empower training speed and simplicity. It is composed of 2 transposed convolutional layers for up-sampling in the generator, and 2 convolutional layers for downsampling in the discriminator, all of them using stride of 2 and kernels of size 5x5.



(a) Generator



(b) Discriminator

Figure 1: Proposed architecture for a Shallow DCGAN

As in [3], all layers are followed by Batch Normalization (except the last of the generator and the first of the discriminator), as well as either ReLU (generator) or LeakyReLU (discriminator) activation functions. We also use a relatively small number of filters (64 and 128, on both). The final output image is 28x28, as in the original MNIST dataset.

We train our DCGAN using an Adam optimizer [4] for both generator and discriminator, with a learning rate of 0.0002 and a momentum term $\beta_1$ of 0.9, as suggested by [3]. We find these values to give better results than the suggested by [4]. We use a batch size of 128, and train our model for 25 epochs.

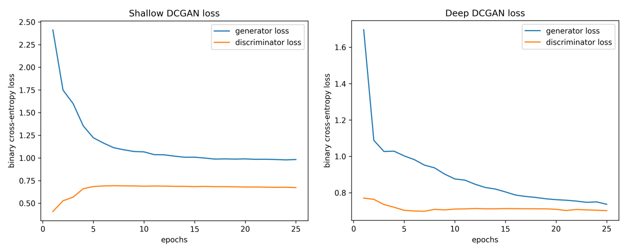
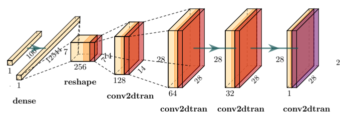


Figure 2: Loss function of the Shallow DCGAN (left) and the Deep DCGAN (right) for a varying number of epochs

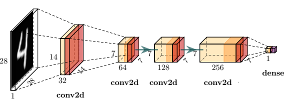
As illustrated in Figure 2 (left), the discriminator starts winning very early against the generator. As we progress through more epochs, however, the generator starts catching up and manages to reduce it's binary cross-entropy loss to about 1. Nevertheless, it never reaches the level of the discriminator. This suggest that more work could be done to balance G and D, for example, training G more often than D. This is something we cover later in Section 3.

## 2.2. Deep DCGAN

Our Deep DCGAN uses 4 transposed convolutions in the generator, and 4 convolution layers in the discrimina-tor. The first 2 in each have a stride of 2, and the last 2 have a stride of 1, maintaining the shape. Again, all layers are followed by Batch Normalization and ReLU/LeakyReLU, as before. The filters used this time are 32, 64, 128 and 256, on both.



(a) Generator



(b) Discriminator

Figure 3: Proposed architecture for a Deep DCGAN

Unlike what happened with the Shallow model, this Deep model does not suffer from such a clear imbalance between G and D, as illustrated in Figure 2 (right). Also, it is noticeable how the generator manages to improve the loss from the Shallow model while the discriminator loss remains about the same (around 0.75).

## 2.3. Qualitative results

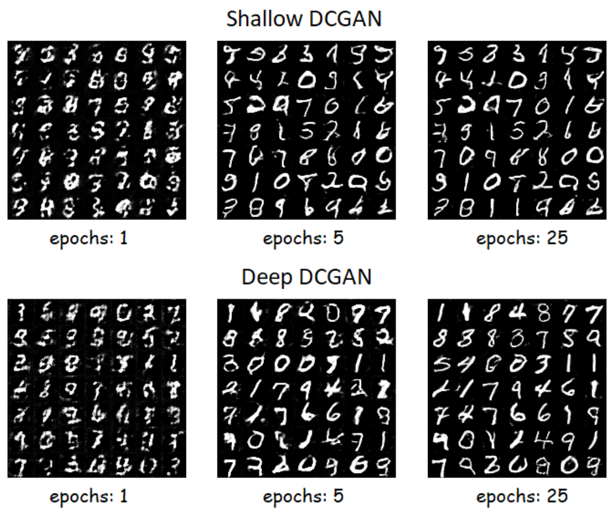


Figure 4: Sample of 49 random digits generated by the Shallow (top row) and Deep (bottom row) DCGANs after 1, 5 and 25 epochs

Qualitatively, in Figure 4 we can see how, as we progress through the different epochs, each model produces more faithful representations of the original dataset. Additionally, and in line with our previous findings, we can see how the Deep network generates more veracious samples than the Shallow one.

Finally, a typical problem around GAN generated examples is *mode collapse*. This occurs when the network becomes too confident in generating too many examples of one class. In this case, while it might be occurring partially, by visual inspection (looking at Figure 4) we do not see a clear mode collapse happening towards any digit number in any of the 2 networks.

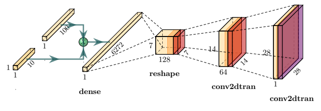
## 3. Conditional Generative Adversarial Networks (CGAN)

In Conditional GANs [5], both the discriminator and the generator are conditioned by a vector **l** that represents some piece of information such as a label.

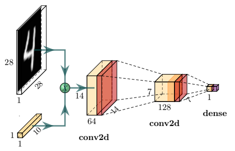
We decided to reuse the architectures proposed in Section 2, into a Shallow and a Deep CGAN. Figures 5 and 6 give a representation of the architectures for both.

The update of the model is achieved by modifying the input. For the generator, a random $U(-1, 1)$ vector of size 100 and a one-hot encoded vector **l** of the labels are embedded and multiplied before being fed to the model. Likewise, the input to the discriminator is composed of the same **l** and a 28x28 digit image, both of which are embedded and multiplied.

## 3.1. Shallow & Deep CGAN



(a) Generator

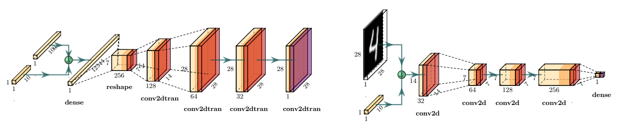


(b) Discriminator

Figure 5: Proposed architecture for a Shallow CGAN

The Shallow CGAN is characterized by its small number of convolutional layers, which aim to empower training speed and simplicity. The Deep CGAN is intended to be better at learning more complex patterns, due to its deeper structure and higher number of parameters. For both cases, after the noise and label are combined, they are processed by essentially the same architectures discussed in Section 2.

The optimizer used for the generator and discriminator is, again, the Adam optimizer with the same parameters $lr = 0.0002, \beta_1 = 0.5$, since they were found to be optimal before, for both cases. The training losses for 20 epochs for both architectures are shown in Figure 7. It can be observed



(a) Generator (b) Discriminator

Figure 6: Proposed architecture for a Deep CGAN

that, in both cases, initially the discriminator has a better performance at classifying images than the generator at creating them. Then, the generator manages to quickly close the gap, but as the number of epochs increases, the discriminator improves significantly. This behaviour is even more noticeable in the Shallow model than in the Deep one.

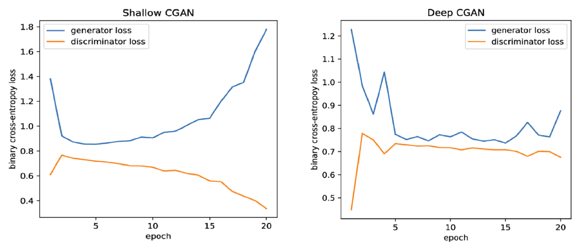


Figure 7: Loss function of the Shallow CGAN (left) and Deep CGAN (right) for a varying number of epochs

## 3.2. One-sided label smoothing

Since in both cases our discriminator shows a good performance, our models can be made more robust by applying *one-sided label smoothing*: this is achieved by changing the label of the real images from 1 to 0.9, which in turn prevents the discriminator from giving a very large gradient to the generator [6]. Figure 8 shows the loss functions of both networks after applying this change.

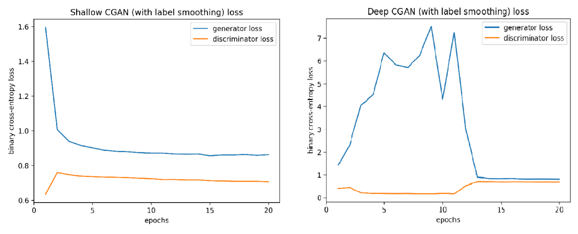


Figure 8: Loss function of the Shallow CGAN (left) and Deep CGAN (right) when applying label smoothing, for a varying number of epochs

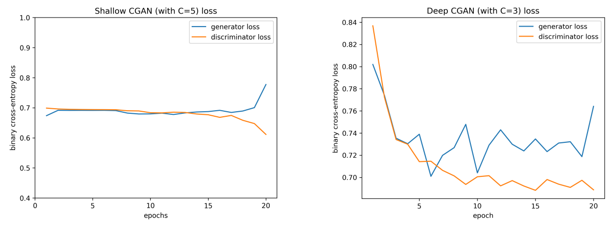
The effects of one-sided label smoothing can be observed immediately, especially in the case of the Shallow CGAN: the generator and discriminator show a more balanced performance than before. Contrary to the previous model, the discriminator does not excessively overpower the generator.

## 3.3. Balancing G and D with a constant ratio

Another solution to avoid the discriminator from overpowering the generator is to have the generator be trained more often. For this purpose, we define a variable $C$, representing the number of times that the generator is trained more than the discriminator. In other words, after the discriminator is trained once on a batch, the generator is trained $C$ number of times.

Given that the Shallow model suffered from this issue more severely than the Deep one, we test a value of $C = 5$ for the Shallow and of $C = 3$ for the Deep model. Figure 9 shows the loss functions for both.

It can be seen how G and D are now more balanced. However, as we will see later in Section 4, this does not necessarily mean that they will produce better examples.



(a) Shallow with C = 5 (b) Deep with C = 3

Figure 9: Loss function of the Shallow CGAN (left) and Deep CGAN (right) when balancing the training with C, for a varying number of epochs

## 3.4. Qualitative results

Next, we analyse qualitatively the performance of the CGANs tested above. Figure 10 contains a randomly chosen sample of a generated digit per class for each of the Shallow and Deep architectures, as well as of the real MNIST dataset. It can be seen that the digits generated by the Deep CGAN are somewhat visually closer to the real MNIST digits (they are slightly clearer and sharper than the ones generated by the Shallow CGAN). However, it fails to generate a good number 7, in this example.

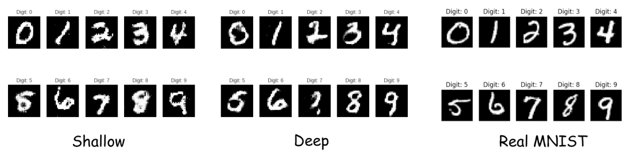


Figure 10: Generated digits from a Shallow (left) and Deep (middle) CGAN, alongside real MNIST samples (right)

The samples for the other architectures tested show a similar performance. None of them show a clear *mode collapse* behaviour, since none of them becomes too confident at generating samples of one particular class saturating the others. However, we did notice how the Deep CGAN with $C = 3$ was not very good at producing examples of class 5, as it can be seen from Figure 11: to the human eye, at least, most of these samples look like digit 6 more than digit 5.

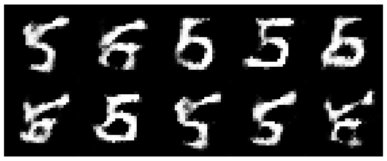


Figure 11: Ten random samples of the output of the Deep CGAN with $C = 3$ for an input of label 5

The rest of the digit classes did not suffer from this behaviour as much as the one depicted above. For a final performance analysis, the inception score is now evaluated in the next section.

## 4. Inception Scores

The loss function for both generator and discriminator gives us an idea about how each of these is performing relative to the other. However, it is not a good metric for measuring image quality or diversity [7].

For this purpose, we use the *inception score*: this is a metric which we define as the accuracy score of the generated examples when passed through a pre-trained MNIST classifier. The inception score gives information about two criteria when measuring the performance of GAN: the quality of the generated images and their diversity. Therefore, it allows us to make a comparison between the generated examples of different CGAN models [7].

| | Inception Score |
|---|---|
| Shallow | 93.94% |
| Shallow *with label smoothing* | 98.04% |
| Shallow *with* $C = 5$ | 92.90% |
| Deep | 94.88% |
| Deep *with label smoothing* | 95.37% |
| Deep *with* $C = 3$ | 93.03% |
| **Real MNIST test set** | **99.21%** |

Table 1: Inception scores (as accuracy % over a classifier) for different CGAN architectures and hyperparameters, as well as for the real MNIST test set

Table 1 contains the combined Inception Score for all classes for each of the CGAN architectures tested in Section 3. The classifier used is a simple CNN (as described in [8]) trained on the real MNIST training set (and validated using the real MNIST validation set). The synthetic examples are class-balanced (same number of examples generated per class).

None of the architectures manage to achieve an inception score higher than the real MNIST test set accuracy (99.21%), which speaks poorly about the quality and generalisation of our CGAN generated examples.

Interestingly, even though it did not have the best loss function behaviour in Section 2, the Shallow CGAN with label smoothing architecture obtains the highest inception score of all. This shows that there is no direct correlation between the loss function and the quality and diversity of the generated examples.

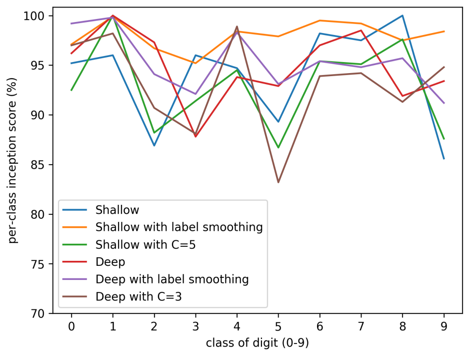


Figure 12: Per-class inception scores (as accuracy % over a classifier) for different CGAN architectures and hyperparameters

Figure 12 contains the inception scores for each of the 6 architectures discussed above. We can see that the problem regarding the digits of label 5 in the Deep CGAN with $C = 3$ manifests here as well: it has the lowest per-class inception score of all (83.13%). This links back to our observations from Section 3.4, where we assessed qualitatively the images generated and saw that the examples generated for an input of 5 were actually visually closer to the digit 6, in many cases.

## 5. Re-training classifier

In this section, we retrain the previous classifier from scratch. However, instead of using the entire real MNIST training set (55,000 samples), we try out different combinations of real and synthetic images. The aim of this experiment is to test the generalization of the generated images from the CGAN. For this reason, we use the model with the best inception score from section 4: the Shallow CGAN with label smoothing. The original validation (5,000 samples) and test (10,000 samples) sets are the ones being used and are, therefore, left unchanged.

### 5.1. Maintaining original dataset size

First, we try out several combinations of real and synthetic images while maintaining a combined training set size fixed at 55,000, to allow for a direct comparison with the real MNIST training set.

We then use two different strategies for training the classifier: training with the real samples first, and training with synthetic samples first. These strategies could lead to different initialization and parameter fine-tuning results, which could, in turn, lead to different performances. The results for both strategies can be observed in Figure 13. In addition, two other strategies are implemented. The generated images with the highest confidence values from the softmax layer are taken to train the classifier.

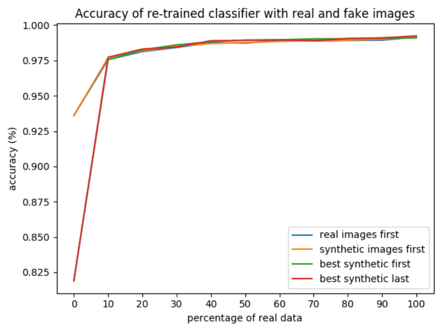


Figure 13: Accuracy (%) of the retrained classifier for real and generated images for both strategies

It can be clearly observed that, in general, no matter the strategy chosen, the larger the proportion of synthetic images (and thus the lower the proportion of real images), the worse the performance of the classifier becomes. The results from the strategies choosing the best (most confident) images are surprisingly poor, particularly for an all-synthetic dataset (0% real samples). An explanation for this is the following: as explained in [9], higher confidence does not necessarily lead to a higher probability of correct classification.

Other than this, all 4 strategies seem to give similar accuracy levels for the rest of the % real data tested.

### 5.2. Augmenting the original real training set

Next, we try out a different approach: using the entire real MNIST training set (55,000 samples), but augmenting it with different quantities of synthetic samples. Again, we test for both real samples first and synthetic samples first. Figure 14 contains the results for both strategies.

Once again, it can be seen that the inherent low quality and low diversity of our generated examples prevents the

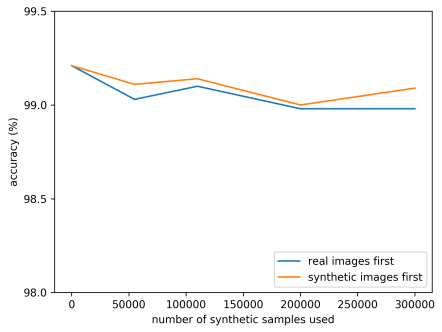


Figure 14: Accuracy of a retrained classifier on a combination of the full real training set (55,000 samples) and an additional varying number of synthetic samples for data augmentation

classifier from outperforming our baseline (trained on the full real training set only) in all cases. Finally, it is worth noting how this time it becomes more apparent that feeding the synthetic examples first has a positive effect on accuracy. This is likely to be due to the fact that our synthetic examples, since they do not generalise well, are worse for fine-tuning the network than the real samples, and therefore are better off being used for initialisation of the network parameters instead.

## References

[1] Yann Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[2] Ian J. Goodfellow et al. “Generative Adversarial Networks”. In: *arXiv e-prints*, arXiv:1406.2661 (June 2014), arXiv:1406.2661. arXiv: `1406.2661 [stat.ML]`.

[3] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[4] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). arXiv: `1412.6980`. URL: `http://arxiv.org/abs/1412.6980`.

[5] Mehdi Mirza and Simon Osindero. “Conditional Generative Adversarial Nets”. In: *CoRR* abs/1411.1784 (2014). arXiv: `1411.1784`. URL: `http://arxiv.org/abs/1411.1784`.

[6] Tae-Kyun Kim. *EE4-62 Selected Topics in Computer Vision - Lecture Notes*. 2019.

[7] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. “How good is my GAN?” In: *CoRR* abs/1807.09499 (2018). arXiv: `1807.09499`. URL: `http://arxiv.org/abs/1807.09499`.

[8] Keras-Team. *keras-team/keras*. URL: `https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py`.

[9] Chuan Guo et al. “On calibration of modern neural networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR.org. 2017, pp. 1321–1330.