GENERATIVE ADVERSARIAL NETWORK
Coursework 2 - Selected Topics in Computer Vision

Ahmet Narman
CID: 01578741
MSc in Human & Biological Robotics

Guillaume Dau
CID: 01554132
MSc in Physics with Nanophotonics

March 15, 2019

1 DCGAN

The DCGAN architectre uses convolutional layers in the generator (G) and discriminator (D) networks. In their paper on DCGAN [6], Radford et al. used 4 hidden convolutional layers for G and D to generate 64x64 images. A code implementing this very architecture using tensorflow was found on GitHub [2] and our own model was built from that code since this architecture already had a proven performance.

As this network worked on 64x64 images, the MNIST images of size 28x28 were resized to 64x64 for training. Although it gave qualitatively good images, even the GPU accelerated training on Google Colab would run an epoch in more than 10 minutes, which was neither feasible nor necessary. Therefore, the architecture was modified so it could work with 28x28 images with less layers and less filters on each layer. Changes were made on the number of layers, sizes of filters and strides to obtain our own 28x28 architecture. Further changes were made to optimize this architecture, which include using different activation functions (e.g. ReLU and leaky ReLU), changing the number of filters on each layer and introducing a dropout at the output layer of D. The results of these changes won't be explained in detail and only the final architecture will be presented here (See the Appendix 4-6 for the results of different parameter choices).

In the final architecture, hidden layers are followed by batch normalization and an activation function (ReLu for G and leaky ReLU with 0.2 negative gradient for D) except in the first hidden layer of D which did not have batch normalization. The output layer of G had a tanh activation to improve image quality, and the output layer of D had a dropout of 0.3 followed by a sigmoidal activation to do the binary assignment of 'real' or 'fake'. The 4 deconvolutional layers of G (the last is the output layer) had 128, 64, 32, 1 filters of size 7x7, 5x5, 5x5 and 5x5 on each layer respectively for G (for D, this architecture is reversed and convolutional layers are used instead of deconvolutional). The following number of strides were used for corresponding layers: 1, 2, 1, 2 for G and 2, 2, 1, 1 for D. Adam optimizer was used to minimize cross-entropy loss for both G and D, as it has a better optimization performance [4], with parameters of beta1=0.5, beta2=0.999 and a learning rate of 0.0002. For training, 30 epochs were done with a batch size of 100.

Before the training protocol, 60K images of MNIST training set were normalized between -1 and 1. In the protocol, random vectors of size 100 were fed to the G as input, and this was done for 100 times (batch size) to generate 100 images. Also, 100 images are taken from the MNIST training set. These image sets were fed to D and weight optimization was done once for D and once for G for every batch. This was done for every epoch, and at the end of every epoch, G and D losses and 100 sample images were recorded. The sample images were generated from predetermined random vectors that did not change over epochs so we could see how the generated images changed for the same vectors.

With the architecture and training protocol detailed above, generated images after the last epoch of training for the DCGAN are shown in Figure 1. Qualitatively, they

resemble the real images from the MNIST dataset, and every digit form 0 to 9 are distinguished, along with some incoherent looking images. The plot of losses for G and D show that D loss decreased over epochs, which means it could discriminate real images from the fake ones better towards the end. Though the loss for G increased because D became better, the quality of the generated images also increased which means G and D was trained in a balanced way. Using Google Colab with GPU acceleration, an epoch took an average of 37 seconds.

The biggest challenge that was seen during development was D becoming overconfident. When that happened, cross-entropy loss for D went down to zero while loss for D increased significantly. The resulting generated images became singular, they were blurry and incoherent (See Appendix 5,6). This problem was overcome by selecting the right hyper-parameters for G and D. Finally, mode collapse was not a big issue and it was encountered only once while tuning parameters (See Appendix 4). The reason for that may be that our model was built on an already proven architecture. Also, for that reason, virtual batch normalization was not implemented as the performance was already satisfactory.

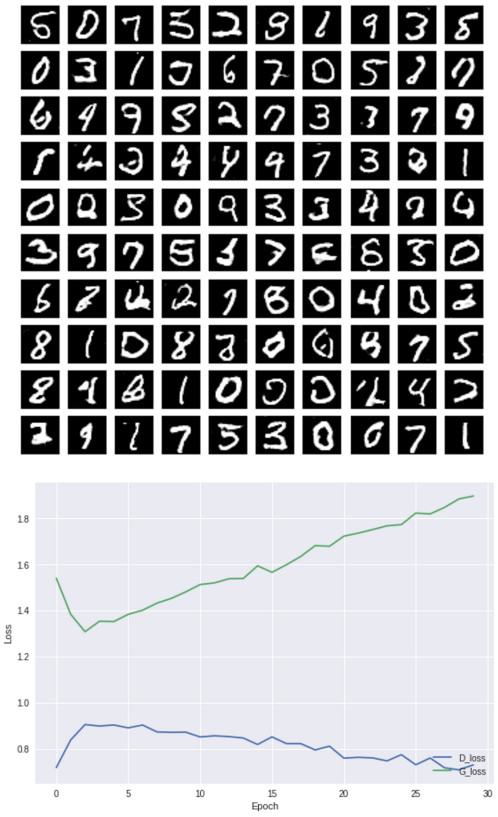


Figure 1: 100 synthetic images generated after training the final DCGAN architecture and cross-entropy losses as a function of the number of epochs for G and D.

2 CGAN

In a CGAN architecture, class labels are also used in G and D so that the classes of the generated images are conditioned on the generated images. The loss functions of G and D are changed so that the outputs are conditioned on the label input [5]. A CGAN code with deep convolutional layers, again, was found on GitHub [1] and thus used as a baseline. Its results were qualitatively bad but we fixed it and implemented an architecture similar to the DCGAN architecture explained in part 1, as our DCGAN architecture exhibited qualitatively good performance.

To generate the class specific images, the labels are incorporated to the G as a 'one hot vector' of size 10. This vector was concatenated to the input noise vector and then fed into the deconvolutional layers. Similarly, the class information was fed to D as a 'one hot image set' of 10 images of size 28x28. One of the images was all 1's and others were all 0's depending on the class label (this was done because it was easier to concatenate this data to the 28x28 image that was fed to the D). The 'one hot image set' was generated according to the conditioned class fed to the G if the image was generated, or the class labels of real MNIST images if the image was real.

After exploring different architectures and parameters and evaluating their results on the quality of generated images while also considering the training time (here, the architecture was kept simple while also generating good looking images, a more objective analysis on the effect of the design choices and parameters will be done with the inception score in the next section), the following CGAN architecture was designed: 3 layers of deconvolution for G with filter sizes 7x7, 5x5, 5x5, filter amounts 64, 32, 1, and strides 1, 2, 2 respectively and 3 layers of convolution for D with filter sizes 5x5, 5x5, 7x7, filter amounts 32,

64, 1, and strides 2, 2, 1 respectively. In both networks, all hidden layers were followed by leaky ReLU activation with 0.2 negative gradient and batch normalization (except for the first layer of D that didn't have batch normalization). The Adam optimizer in DCGAN part was used here (with exponentially decaying learning rate starting from 0.0002) with 30 epochs and 100 batch size. The training protocol was the same with the DCGAN training protocol except the class labels now fed to G and D, and incorporated in the loss functions.

The results for this architecture are given in Figure 2. It was seen the digits generated are coherent and class specific. Also, it was seen that the images of same classes have a certain in class variability depending on the noise input to the G, meaning that not all generated images of a class looks the same. The plot for losses over epochs suggest that D is becoming better at discriminating class specific real images from the fake ones, while G also is also better at generating class specific images.

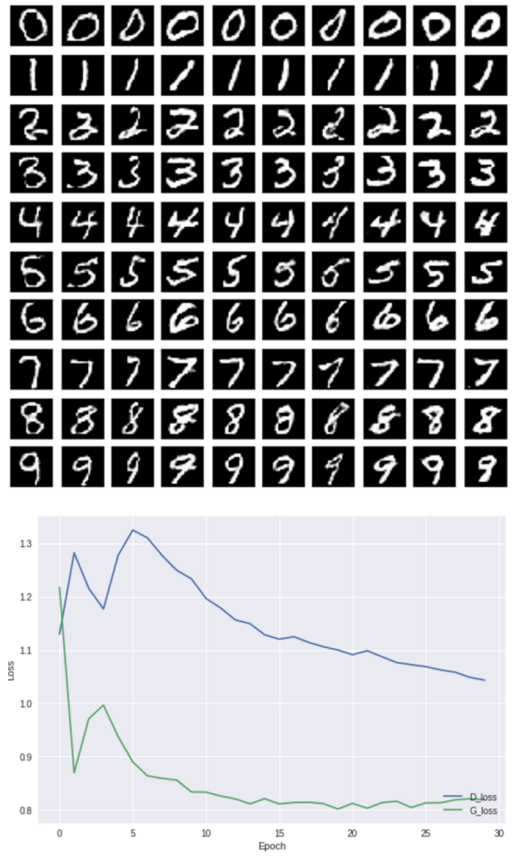


Figure 2: 100 synthetic images generated after training CGAN for 30 epochs and cross-entropy loss for G and D. Losses show waving trends and converge after 30 epochs.

The biggest challenge we faced when developing our CGAN was that we used a problematic GitHub code that made this task hard, as changing parameters quickly destabilized D and G. This problem was overcome by understanding the code and architecture deeply which led fixing a typo that caused a failure in generating labels for digit '9' (see Appendix 7). One sided label smoothing was tried as well by making the positive classification targets 0.9 instead of 1 but no significant qualitative difference was observed, thus it was not included in the final architecture in this part (See the next section for a more objective analysis on its effect). No mode collapse issue was encountered, because with CGAN, we enforce generation of images of different classes.

3 Inception score

In this section, the CGAN architecture previously obtained is used to generate 10k synthetic images. These images are then fed to a pre-trained CNN classifier to predict their class. The multi-class classifier used in this work is based on LeNet architecture with a softmax activation function to allow non-binary classification from class conditional probabilities. It was taken from the 2nd Lab Session baseline code and implemented with an Adam optimizer instead of gradient descent to further boost its performance. The classifier is trained on the original MNIST training set (the images are resized to match the 32x32 classifier input shape by 0-padding) split into 55k images for training and 5k for validation during training (the validation score was of 0.99 at the end of 30 epochs).

Then, the inception score (IS) was calculated using the following equation from [3]:

$$IS(G) = exp(E_{x \sim p_G} D_{kl}(p(y|x)||p(y)))$$

$$= exp(\frac{1}{N} \sum_{i=1}^{N} \sum_{j=0}^{9} p(y_j|x_i) log(\frac{p(y_j|x_i)}{\hat{p}(y_j)})$$

where the marginal distrubutions $\hat{p}(y_j)$ were found by applying softmax activation to the *logits* output of the classifier (evaluated on the synthetic data we generated previously) and then using the following equation:

$$\hat{p}(y_j) = \frac{1}{N} \sum_{i=1}^{N} p(y_j|x_i)$$

IS was computed 10 times for each synthetic image set with batches of size 1k to get its mean value and standard deviation. A problem occured when at least one of the $p(y_j|x_i)$ was equal to 0 (log not defined), which was overcome by adding the machine epsilon to zero values.

Different CGAN hyper-parameters were tried to obtain an architecture giving optimal results in terms of IS and classification accuracy. The best architecture was the base architecture that is improved by one sided label smoothing and twice as many filters (128, 64 for G, 64, 128 for D) (See Appendix Table 2 for its intra-class accuracies) as it can be seen on Table 1.

From the results, it was seen that accuracy and IS were mostly correlated and could have been used interchangeably. Also, it was found that using one sided label smoothing and increased number of filters increased the CGAN performance, such that we even had better accuracy and better IS compared to the real MNIST test set.

Test data from	*Accuracy (%)*	*Inception Score*
MNIST Test set (real)	99	9.962 ± 0.044
DCGAN (synthetic)	–	9.026 ± 0.043
CGAN (synthetic):		
Base	98.7	9.982 ± 0.041
Base + OSLS*	99.3	$\mathbf{10.032 \pm 0.06}$
Base + OSLS +Extra layer	94.9	9.75 ± 0.022
Base + 2*num_filters + OSLS	**99.7**	10.029 ± 0.028
Base + filter size of 4x4 +OSLS	95.8	9.74 ± 0.055
Base + 2*num_filters + Dropout layer + OSLS	99.3	10.08 ± 0.037
Base + num_filters/2	95.3	9.835 ± 0.038
Base + num_filters/2 + 15 epochs	92.1	9.706 ± 0.029

Table 1: Inception score and test accuracy for synthetic images generated for DCGAN and different CGAN achictetures. Accuracies are obtained when training LeNet classifier on the generated synthetic data and tested on the MNIST test set. Base: baseline model of cGAN (see part 2). OSLS: One Sided Label Smoothing. Extra Layer: 3 hidden layers used. Dropout:0.3. Numfilters: the amount of filters on each layer wrt. the base model.

4 Re-training the classifier

In this last section, our aim is to study the classifier performance at predicting MNIST test data correctly when trained both on CGAN-generated synthetic images and real MNIST training images. To do so, 55K synthetic images are generated (equally representing the 10 classes) with our optimal CGAN architecture that is found in the previous part. Then, our LeNet classifier is trained with different proportions of real and synthetic data (and their labels) to always form a training set of a size fixed at 55k samples. Note that when training the classifier, the validation set is still composed of 5k MNIST training images. Finally, the classification accuracy of our classifier is measured on the 10k MNIST test images as test set.

First, to obtain the *asbolute* upper bound accuracy, we train the classifier on all the real MNIST training images (55k for training and 5k for validation) for 30 epochs with Adam optimizer and test it on the MNIST test set. This upper limit is of 99.1% for a training time of 68s.

Then, when training the classifier only with the 55k synthetic images, a prediction accuracy of 93.2% for 30 epochs (goes down to 82.4% for 15 epochs) is obtained which was set as the lower bound accuracy since the proportion of real MNIST data - similar to the test data - is null. Hence, we can say our CGAN produces very 'real-looking' synthetic images and our classifier is capable of a good generalization as it classifies unseen data which contains unseen new patterns - being real images and not synthetic ones - very well while discovering such data.

We, then, varied the proportion of real MNIST images in the 55k-sample training set, the resulting evolution of the accuracy is shown in Figure 3. It can be observed that when adding even a very few real images to the training set classification accuracy improves significantly. Indeed, allocating 0.018% of the training set to real MNIST data (60 samples out of 55k) leads to a 2% accuracy increase and 0.1% of this size to another 1% one. However, we also see that for more than 20% of real images in the training set, reserving a bigger part of the training set to the real samples does not improve our classifier's accuracy further, the latter already reaching its top score of 99.1% or laying very close it (+/- 0.2%), as highlighted in Figure. 3.

However, it is interesting to see if by training the classifier in a smarter way using different training strategies

- without enlarging the training set - we could improve its prediction accuracy. In particular, we looked at what happened if our classifier's trained using only the synthetic samples contained in the training set (X samples) for the first 15 epochs and then fine tuned using only the remaining real images from the same training set (55k - X samples) for the last 15 epochs. (See Appendix 8). This first strategy exhibits classification performance that are slightly lower compared to the previous training scheme for proportions of real images under 1% - nonetheless still improving - as displayed in Figure. 3 *middle*. This was expected since each individual sample from the training set is fed to the classifier half the time compared to the training with no strategy. Consequently training time dropped by almost 50% to 36s, for the most unbalanced proportions within the training set (0-10% and 70-100%). For instance an accuracy of 99% is now obtained for a training time of 42s (gain of 1/3) with a training set containing 40% of real samples.

Furthermore, still following this idea of training the classifier separately on real data and synthetic data from the same training set, we also studied the effect of training our classifier alternatively on synthetic and then real data for still for 30 epochs - i.e. on the restriction of the training set to synthetic samples during one epoch and inversely during the next one (See Appendix 9). Again this pinpoints the already explained twice as less learning that the classifier can benefit from, on each training samples. Results are now better than for the first strategy but still a tiny bit under the no strategy training for proportions of real images under 1%. However, passed this proportion, predictions are almost exactly the same as for the 2 previous training methods (See Figure 3. *bottom*) but the training is now even shorter as it takes 37s regardless of the training set composition which constitutes significant time savings for top performance predictions.

Lastly, the classifier was trained using all synthetic and real training data available - thus forming a massive set of 110k samples - all other parameters unchanged. Its accuracy did not go higher than the upper limit. This perfectly support our final remark, reaching higher accuracies past this limit is very hard as our synthetic samples - already of a really good quality and almost perfectly mimicking real training MNIST images - cannot outperform them whatsoever. The few disparities still requires even a tiny amount of real images to be handled.

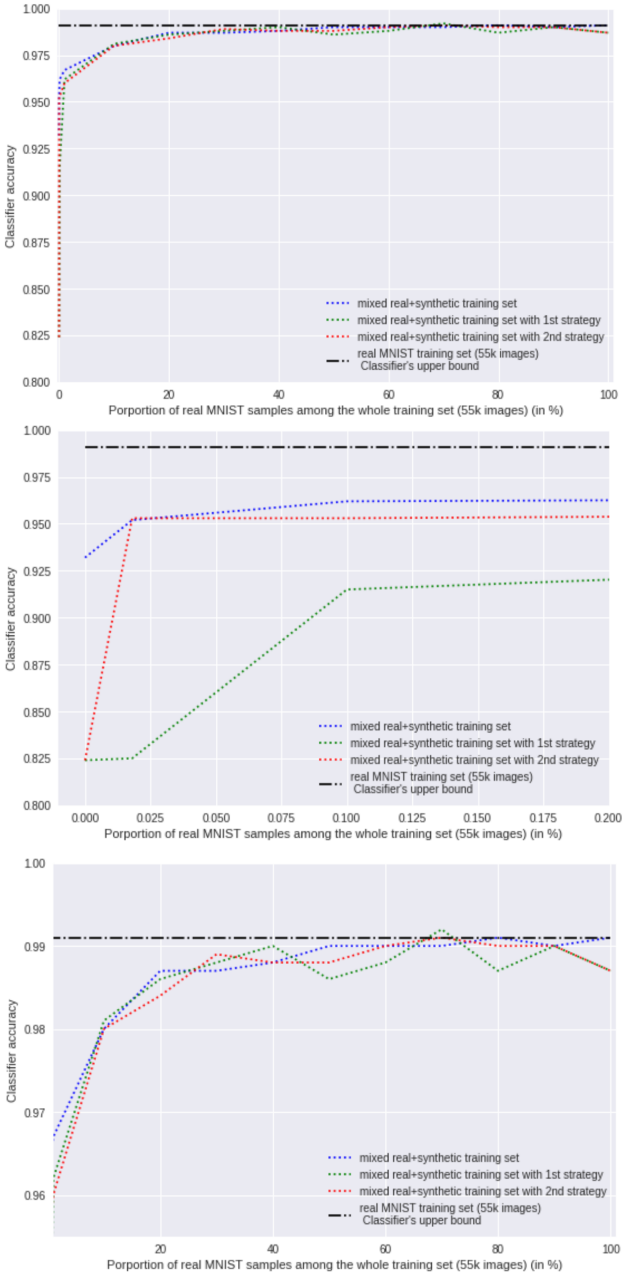


Figure 3: ***top*** Evolution of the classification accuracies (with or without strategy) with varying proportions of real and synthetic samples among the training set,
middle zoom on the left part of the graph,
bottom zoom on the right part of the graph.
Classifier trained for 30 epochs with Adam optimizer (learning rate: 0.001. For the full results, see Table 3 (Appendix).

References

[1] The sample code used for cgan. https://github.com/znxlwm/tensorflow-mnist-cgan-cdcgan.

[2] The sample code used for dcgan. https://github.com/znxlwm/tensorflow-mnist-gan-dcgan.

[3] S. R. Barratt, S. A note on the inceptionscore. Retrieved from: https://arxiv.org/pdf/1411.1784.pdf.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization.

[5] O. S. Mirza, M. Conditional generative adversarial nets. Retrieved from: https://arxiv.org/pdf/1411.1784.pdf.

[6] M. L. C. S. Radford, A. Un-supervised representation learning with deep convolutional generative adversarial networks. Retrieved from: https://arxiv.org/pdf/1511.06434.pdf.

Appendix

Class	*Accuracy (%)*
0	99.6
1	99.8
2	99.5
3	99.9
4	99.7
5	99.8
6	99.6
7	99.7
8	99.5
9	99.8

Table 2: Class dependent classification accuracy of the best performing CGAN architecture (base model, 2*filters, one sided label smoothing), found in section 3

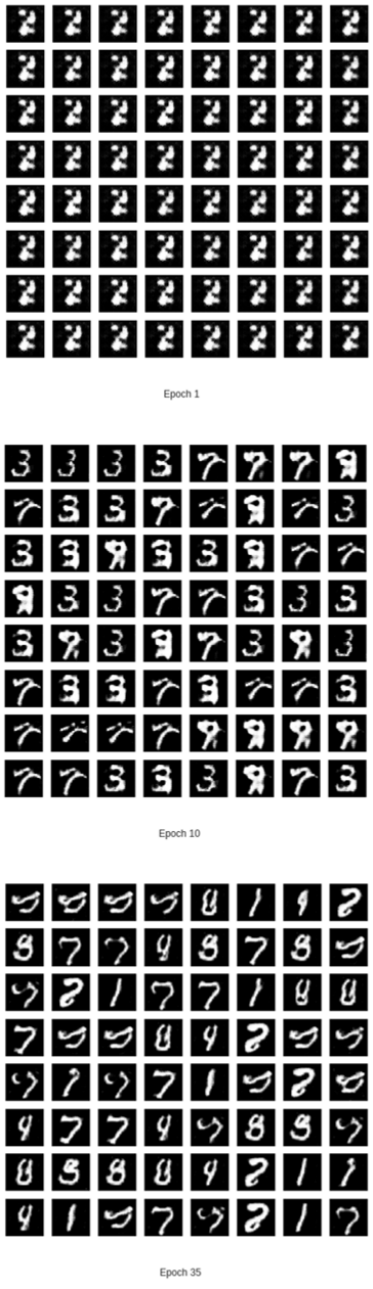


Figure 4: Mode collapse for DCGAN, where only some instances are produced all the time

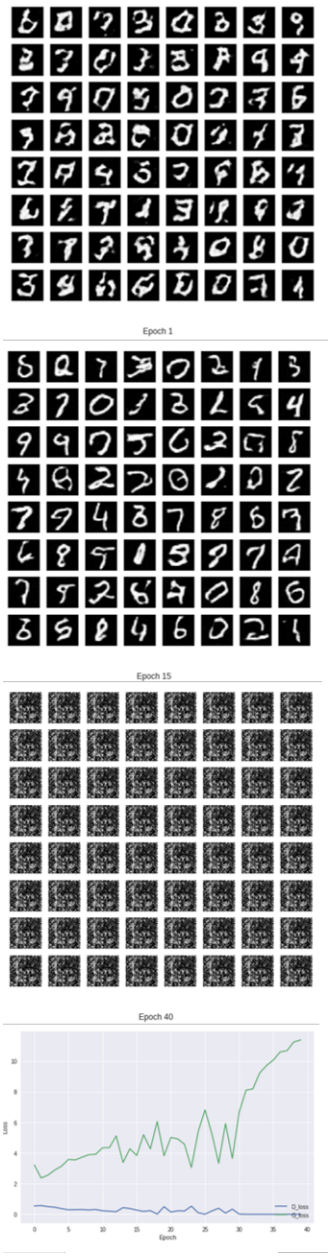


Figure 5: D becoming overconfident in DCGAN. Notice the generated images at the last epoch and the increase of G loss when D loss becomes zero

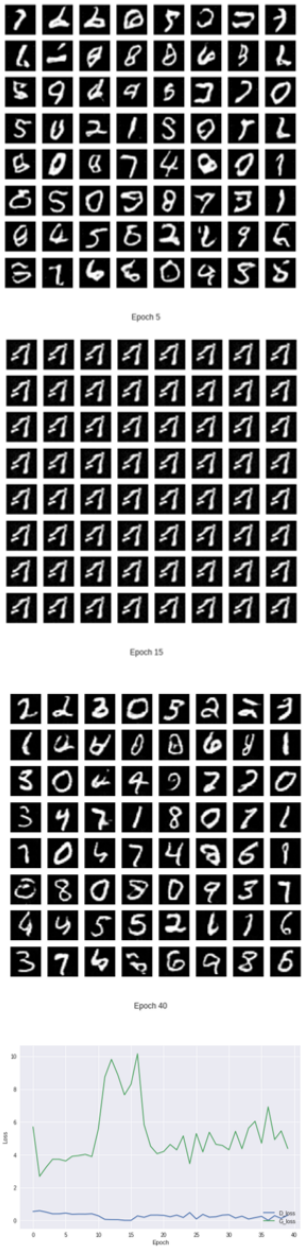
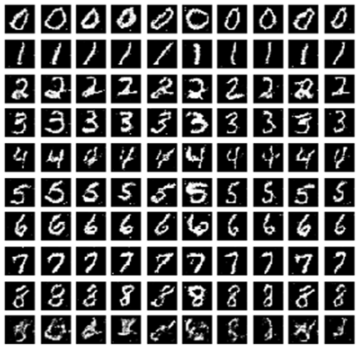
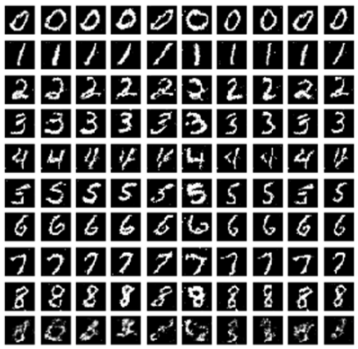


Figure 6: D becoming overconfident and then coming back in DCGAN. Notice the generated images at epoch 15 and the change in G loss when D is overconfident



Epoch 30



Epoch 60

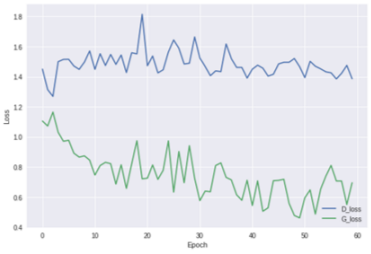
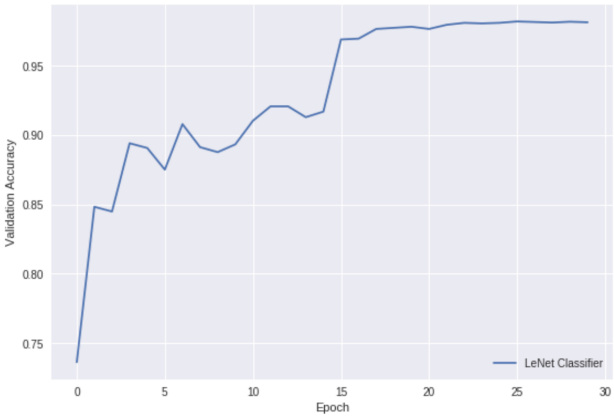


Figure 7: Results of the original GitHub code for cGAN. Notice the problem with generating 9s and the problematic loss attitudes. This was caused by a typo that prevented the GAN from generating 9s and reduced stability



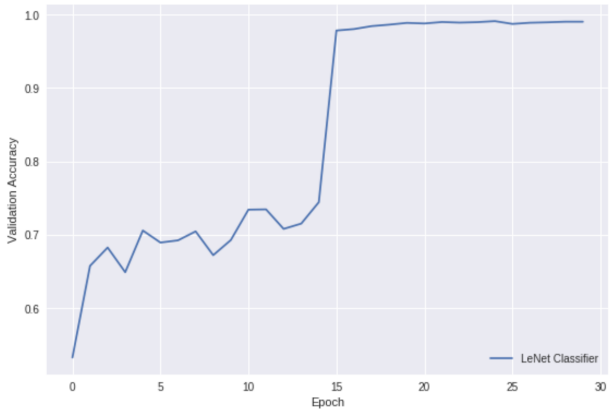


Figure 8: Validation accuracy curves during the classifier training the first strategy for 10% (***top***) and 90% (***bottom***) of real images among the training set of size 55k .
All the curves show the complementarity of these two training steps as a steep and neat validation accuracy gain at the 15th epochs. Indeed, the classifier first learns to initialize its parameters on synthetic data and then fine tune them to reach much higher good prediction scores on real data, supposedly more intrinsically similar to the MNIST test data. Note that the left part of the validation accuracy curve is shifted down as the real sample proportion grows. This is due to the training set containing less and less synthetic samples and eventually with less intra- and extra-class diversity. Thus less training data it fed to the classifier for the corresponding epochs (15) so it learns less and more slowly. On the contrary, even with few real samples to train on the last 15 epochs, the classifier is able to still learn useful features as its accuracy gain still jumps to top score ($>98\%$).

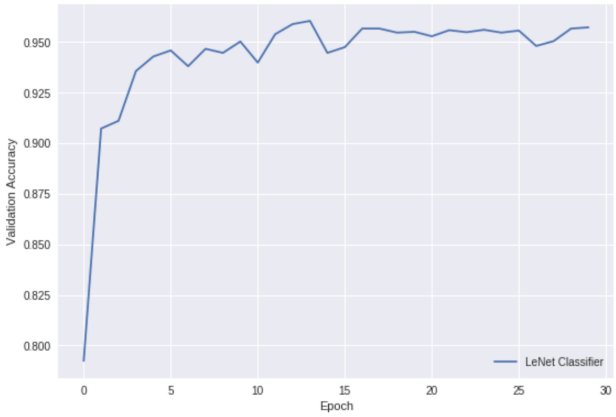


Figure 9: Validation accuracy curve during the classifier training with the second strategy for 0.1% of real images among the training set of size 55k .

Although, when the classifier is trained on 550 real images (55 per class) only during the 'real data' epochs, this curve shows that it still learns at the same pace when using 54450 synthetic samples or only 550 real ones, for the first few epochs. Once again, this is a sign of good complementary in the two training steps. Then, as the classifier already reaches high accuracies, the variations can be explained by this difference on the data used from one epoch to the other but also by randomness of the batches made.

Percentage of real samples among the training set of size 55k / corresponding number of samples	**Accuracy**		
	Training without strategy	*Training with the 1st strategy*	*Training with the 2nd strategy*
$0-0$	0.932	0.824	0.824
$0.018-10$	0.952	0.825	0.953
$0.1-60$	0.962	0.915	0.953
$1-550$	0.967	0.962	0.96
$10-5.5k$	0.980	0.981	0.98
$20-11k$	0.987	0.986	0.984
$30-16.5k$	0.987	0.988	0.989
$40-22k$	0.988	0.990	0.988
$50-27.5k$	0.99	0.986	0.988
$60-33k$	0.99	0.988	0.99
$70-38.5k$	0.99	$\mathbf{0.992}$	$\mathbf{0.991}$
$80-44k$	$\mathbf{0.991}$	0.987	0.99
$90-49.5k$	0.990	0.990	0.99
$100-55k$	$\mathbf{0.991}$	0.987	0.987

Table 3: Test accuracies for the different training approaches pursued in section 4. (See Figure 3).