LINMA2472: Algorithms in Data Science
HW3: GAN and CNN to conquer MNIST Report

Pietro Gavazzi, Siqi Hu, Isabella Sofía Quicaño Pizarro

November 2022

Contents

0 Introduction

The report is about training the GAN model for image generation and the CNN model for image classification. It is also the first attempt at deep learning topics using PyTorch as the main tool to solve problems, especially for image processing.

1 GAN on MNIST for image generation

1.1 GAN architectures

We kept the reference generator and discriminator model.

1.1.1 Generator

In the generator, there are mainly five layers, which are one linear transformation layer and 4 transposed convolution layers, with each layer followed by an activation function, not including the input and the output layers. The size of each layer is $(32, 64 \times 7 \times 7)$ for the 1^{st} layer or $(32, 64, 7, 7)$ after being viewed, $(32, 64, 14, 14)$ for the 2^{nd} layer, $(32, 32, 14, 14)$ for the 3^{rd} layer, $(32, 16, 14, 14)$ for the 4^{th} layer, and lastly $(32, 1, 28, 28)$ for the 5^{th} layer. The types of layers are the fully connected layer and the convolution layer.[1] The total number of parameters for the generator is 246529. The dimension of latent space for the generator is 100. The inputs are in the size of (32, 100), and the outputs are in the size of (32, 1, 28, 28) if the batch size is 32. You will find an illustrated graph of the generator architecture in Figure 1.

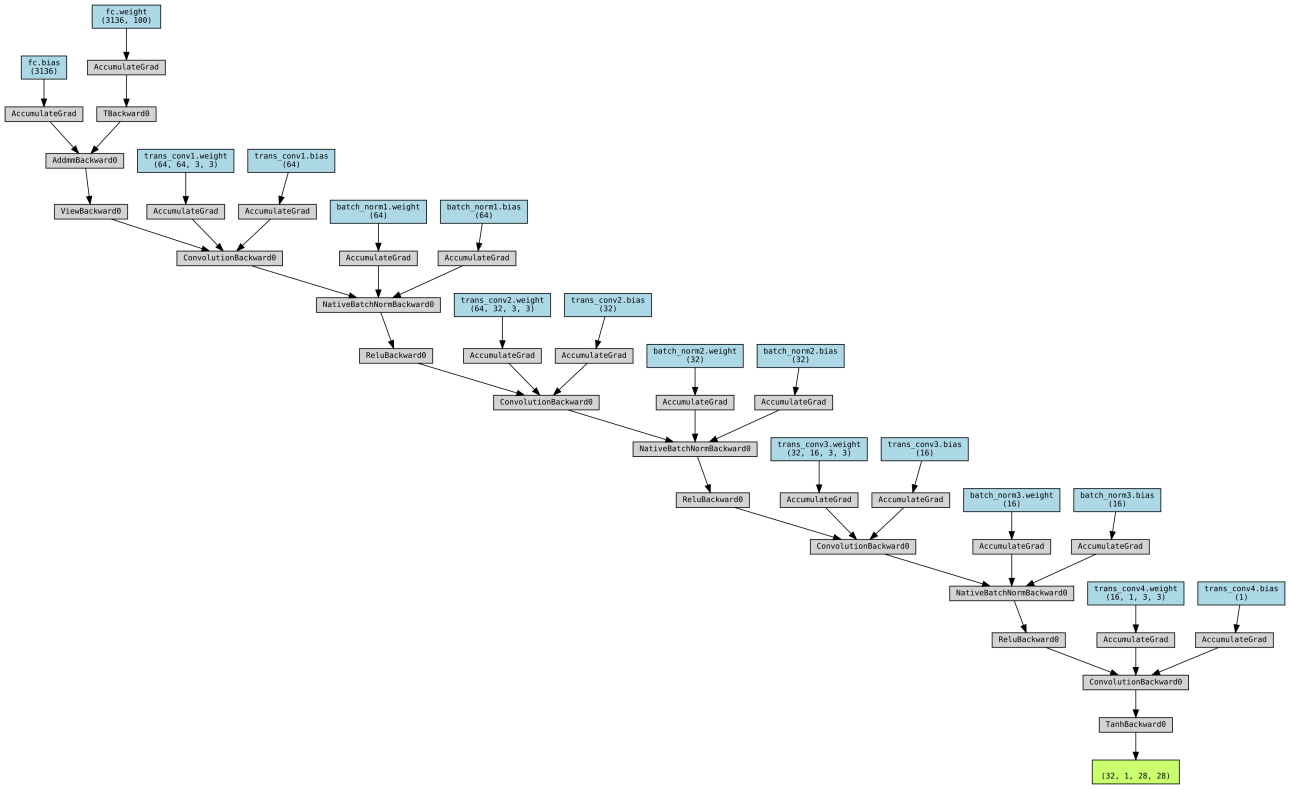


Figure 1: Generator Architecture

1.1.2 Discriminator

For the discriminator, there are also mainly 5 layers, with 4 convolution layers and one linear transformation layer at the end, with each layer followed by an activation function and a dropout layer,

except for the last layer, which is the linear layer. The size of the input data is $(32, 1 \times 28 \times 28)$ or $(32, 1, 28, 28)$ after being viewed, then the 1^{st} layer with $(32, 32, 14, 14)$, the 2^{nd} layer with $(32, 64, 14, 14)$, then the 3^{rd} layer with $(32, 128, 14, 14)$, the 4^{th} layer with $(32, 128, 7, 7)$ or $(32, 128 \times 7 \times 7)$ viewed at this layer after the activation function, then the 5^{th} layer with $(32, 1)$ as the output of the discriminator model, where 32 is the batch size. The types of layers are therefore convolution layer and fully connected layer.[1] The total number of parameters for the discriminator is 377121. The illustration of the discriminator architecture is shown in Figure 2.

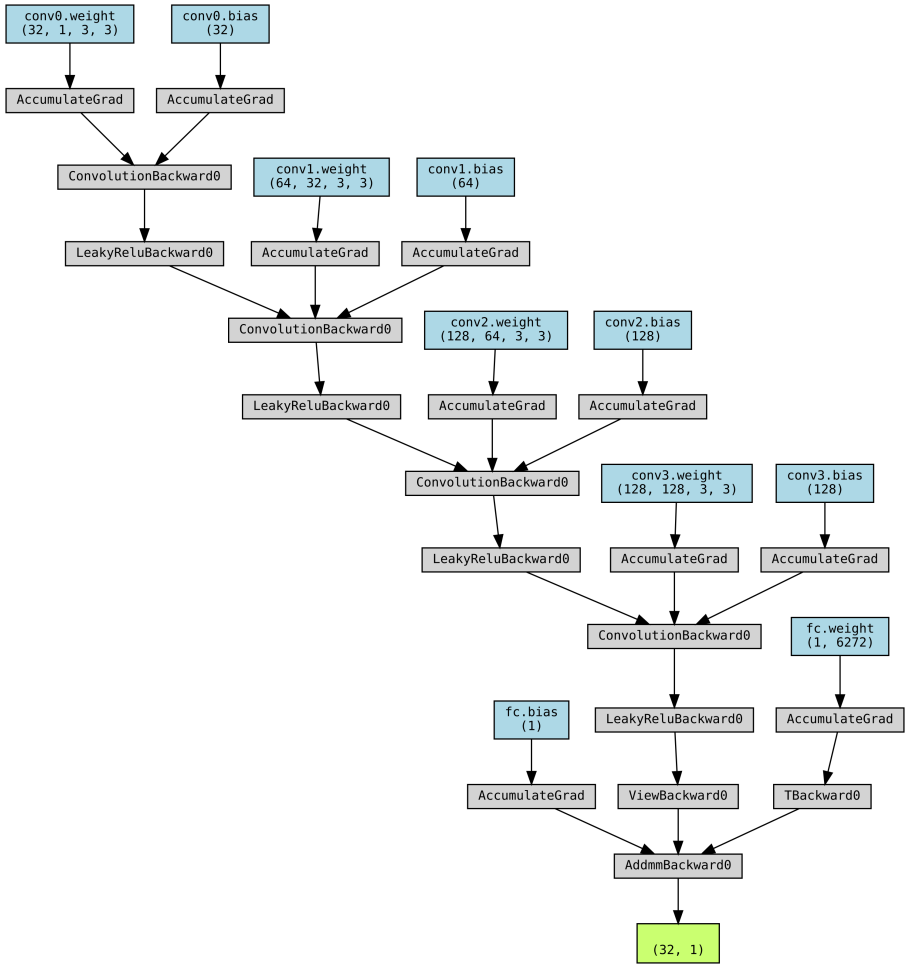
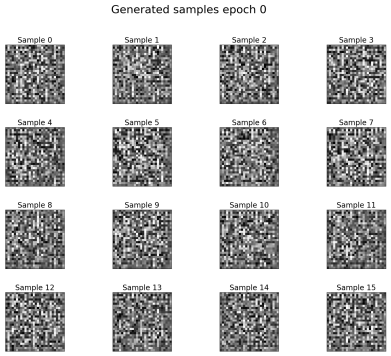


Figure 2: Discriminator Architecture

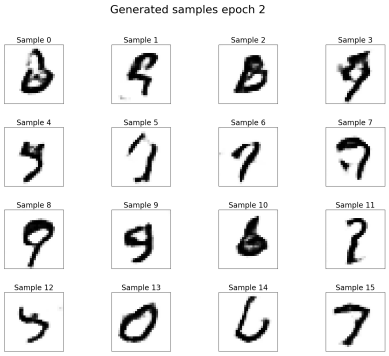
1.2 Train the GAN model

The dataset used for training the GAN model is the MNIST dataset. The device used is a CPU, the number of epochs is 5, and the batch size is 32. For the Adam parameters used, the learning rate is 0.0002, and betas are $(0.5, 0.999)$

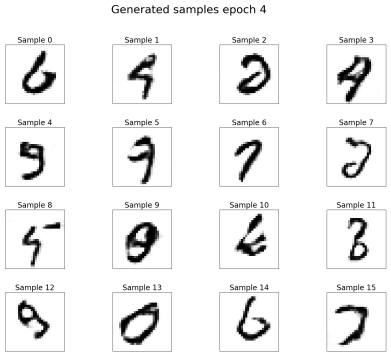
Figure 3 below shows the evolution of generating fake images through epochs and one image from the real samples. We can see that from the beginning (epoch 0), the images are almost noise images, and through training the GAN model (epoch 2), we can see find the outlook of these numbers even though it is not clear enough, but until the end of the epochs (epoch 4), the images become more recognizable. However, compared with the image from the real dataset, in which we can clearly identify which number it is visually, it is obvious that the generated images are still ambiguous even for the images in epoch 4.



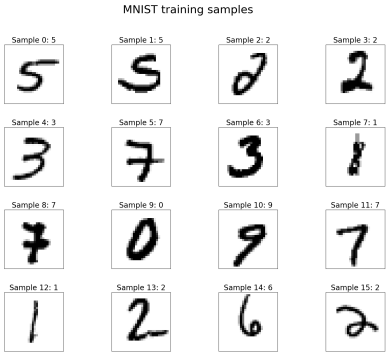
(a) Generated samples epoch 0



(b) Generated samples epoch 2



(c) Generated samples epoch 4



(d) Real MNIST samples

Figure 3: The evolution of generated images in epoch 0, epoch 2, and epoch 4 and the image of real MNIST samples.

The predictions of the discriminator on one batch of the real MNIST samples are listed below in Table 1. If the sample is deemed as real, it will be labeled as 1, and if it is fake, then 0. The input samples here are all from the real dataset, therefore their real labels should be 1 for the discriminator to distinguish real samples from fake samples. However, there are 10 samples that are labeled with 0, demonstrating that the discriminator classifies it as a fake image, in the total amount of 32 samples.

Table 1: Predictions on one batch of real MNIST samples.

Samples	Sample 0	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6
Predictions	1	1	1	1	0	1	0
Samples	Sample 7	Sample 8	Sample 9	Sample 10	Sample 11	Sample 12	Sample 13
Predictions	0	0	1	1	1	1	0
Samples	Sample 14	Sample 15	Sample 16	Sample 17	Sample 18	Sample 19	Sample 20
Predictions	1	1	0	1	0	1	1
Samples	Sample 21	Sample 22	Sample 23	Sample 24	Sample 25	Sample 26	Sample 27
Predictions	0	1	0	0	0	0	0
Samples	Sample 28	Sample 29	Sample 30	Sample 31			
Predictions	1	1	1	0			

1.3 Error metrics during training

The loss function of the generator and the discriminator is shown below. As can be seen in Figure 4, the loss of the generator and the loss of the discriminator tend to level off at a certain range when the as

the epoch goes. It is interesting to see that at the beginning of the training the loss of the discriminator is much higher than that of the generator, but it obviously drops down when reaching epoch 1, while the loss of the generator slightly rises up from epoch 0 to epoch 1. Starting from epoch 1, the loss of the discriminator is generally higher than that of the generator.

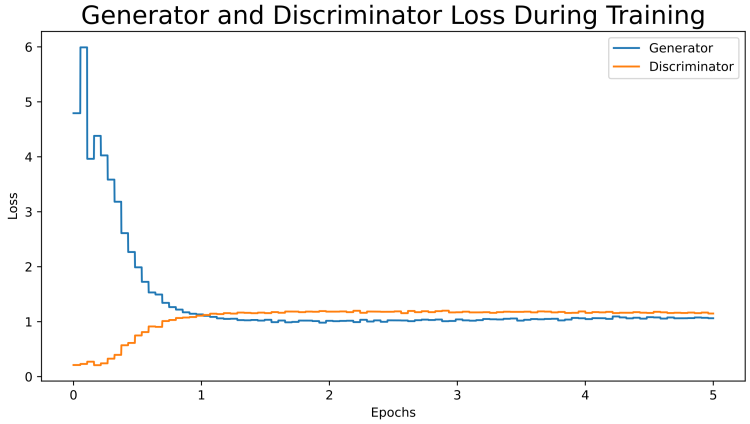


Figure 4: The loss function of the generator and the discriminator by epochs. The x-label is actually iteration numbers, but here it has been converted to epoch numbers. The plot is based on the average of every 100 iterations.

During the training, the accuracy of the discriminator on both real samples and fake samples will be presented below. In our report, if the accuracy of the discriminator on real and fake samples is larger, it means the discriminator is more effective to classify whether the input sample is fake or real. Hence, in Figure 5, as shown below, generally the accuracy of the fake samples is higher than that of the real samples. At the beginning of the training, the accuracy on the real sample is almost 1, but as the epoch goes up the accuracy decreases. The accuracy tends to be between 0.65 and 0.75 as the epoch goes.

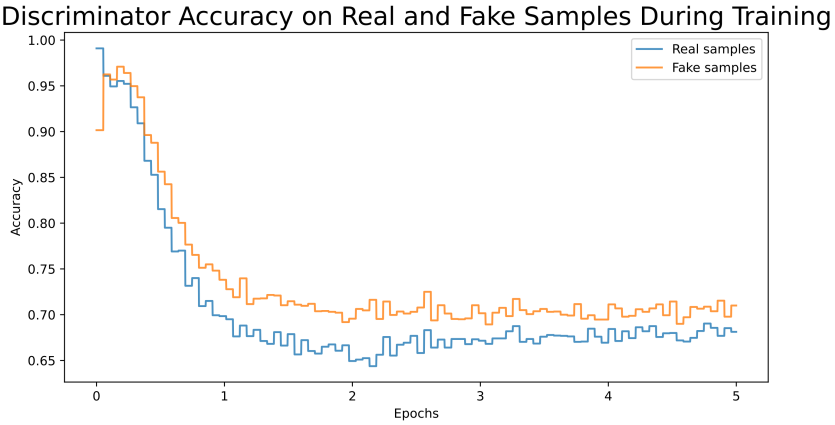


Figure 5: The accuracy of the discriminator on both real and fake samples by epochs. The x-label is actually iteration numbers, but here it has been converted to epoch numbers. The plot is based on the average of every 100 iterations.

The duration of the training is shown in Figure 6.

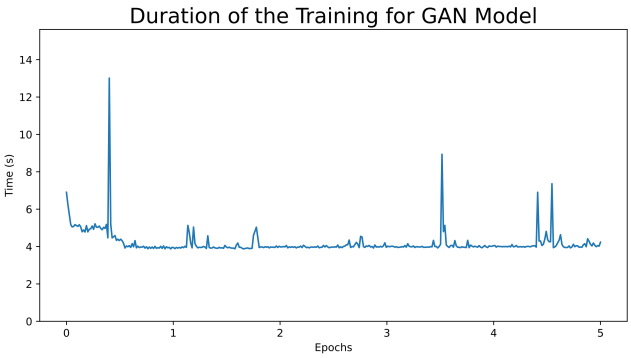


Figure 6: The duration of the GAN model training. The duration of the training is computed for every 25 iterations.

The accuracy of using the trained discriminator model on the training set is 0.8091, and the accuracy on the testing set is 0.8128. Generally, based on the accuracy of the classification, the model works effectively.

To sum up, in the beginning, the discriminator is very good at recognizing fake data, then little by little during the first epoch, the generator is more and more able to fool the discriminator. If we check Figure 4 we see that the generator loss decreases unemotionally proportional to the increase in the discriminator loss until both losses are more or less the same. This corresponds to a precision of the Discriminator 70%. But it is not because the precision remains constant that the model does not improve: We can see in Figure 3 the images are more realistic for epoch 4 than for epoch 2.

1.4 Improve the GAN model

We see that the images the generator makes that are not real numbers are images with a certain symmetry.

We can make the assumption that a label predictor would not predict one label for those with high certainty.

We can thus give the loss function of the generator a "score" that is 1 if the discriminator predicted a true image and if the predictor predicted a value with a high probability.

For speeding up the training, we can also increase the learning rate.[5] If we are not using GPU for the computation, we can multiply the learning rate by several times to spend less time on training.

1.5 BONUS: GAN loss function

The loss function of the generator is:

$$\min_G V(G) = \mathbb{E}_{z \sim P_z}[log(1 - D(G(z)))], \tag{1}$$

where z is from latent space as the input. For the discriminator, the loss function is:

$$\max_D V(D) = \mathbb{E}_{x \sim P_{data}}[log(D(x))] + \mathbb{E}_{z \sim P_z}[log(1 - D(G(z)))], \tag{2}$$

where the input x is from the real dataset and $G(z)$ is generated fake dataset from latent space.

Because the discriminator will label real data as 1, and the other data as 0,

$$D(x) = \begin{cases} 1, & x \; is \; real \\ 0, & otherwise. \end{cases} \tag{3}$$

For the generator, it hopes the generated fake data can be classified as the real data by the discriminator, which means $D(G(z))$ will be close to 1, and therefore, $log[1 - D(G(z))]$ will be the minimum, which comes to the objective function of the generator's loss function.

For the discriminator, it hopes the real data will be classified as 1, and the fake data will be labeled as 0, which means $D(x)$ will be close to 1 and $D(G(z))$ will be close to 0, which leads to the maximum of $log[D(x)]$ and $log[1-D(G(z))]$. That comes to the objective function of the discriminator's loss function.

1.6 BONUS: Discriminator output and loss function

BCELoss-WithLogits is a function where the result is the same as if we applied the sigmoïd function before using the classical BCELoss, but it is more stable.

In fact, we can easily see in the formula hereunder that the logarithm of the entropy and the exponentials of the sigmoid simplify themselves improving round-around errors.

$\sigma(x) == \frac{e^x}{e^x+1}$

$\text{BCE}(\sigma) = y\ln(\sigma(x)) + (1-y)\ln(1-\sigma(x)) = y\ln(\frac{e^x}{e^x+1}) + (1-y)\ln(1-\frac{e^x}{e^x+1})$
$= y(x - ln(e^x+1)) + (y-1)(ln(e^x+1)) = y(x - ln(e^x+1) + ln(e^x+1)) - ln(e^x+1)$
$= yx - ln(e^x+1)$

Where y is the real label (1 if a true image or 0 if false) and x is the sum of the last layer of the discriminator.

2 CNN on MNIST for image classification

2.1 CNN Architectures

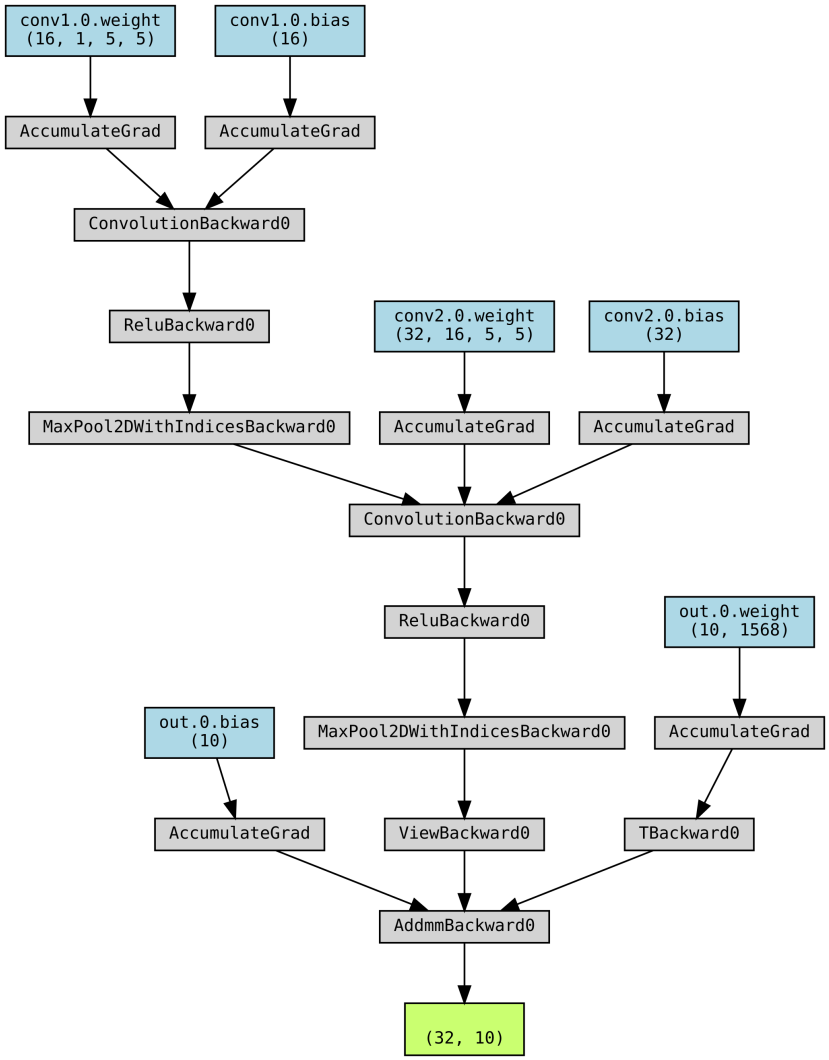


Figure 7: CNN Architecture

We kept the reference CNN architecture given in the Jupyter notebook file provided, where there are mainly 5 layers constituting it: two convolution layers, two max-pooling layers, and one linear

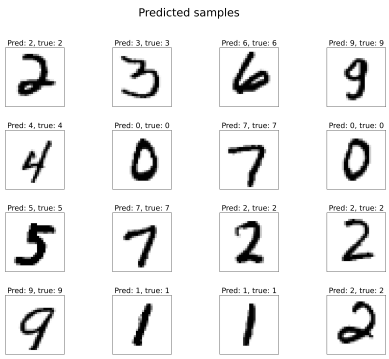
transformation layer. For each convolution layer, it is followed by one activation layer and a max-pooling layer. If the batch size is 32, the size of the input data is $(32, 1, 28, 28)$, then the 1^{st} convolution layer with $(32, 16, 28, 28)$, then the first max-pooling layer with the size of $(32, 16, 14, 14)$, then the 2^{nd} convolution layer with $(32, 32, 14, 14)$, then the 2^{nd} max-pooling layer with the size of $(32, 32, 7, 7)$, after being viewed with the size of $(32, 32 \times 7 \times 7)$, and it finally goes to the final linear transformation layer, with an output size of $(32, n_{class})$. The n_{class} is the number of the classes. The types of the main layers are, therefore, convolution layers, pooling layers, and a fully connected layer[1]. The total number of parameters for the CNN model is 28938. You will find an illustrated graph of the CNN architecture in Figure 7 above.

After that, there are two annex functions:
(1) "scaleToProbabilities" that returns the output of CNNbabilities for each label.
(2) "predictLabels" that returns the most probable label, his certainty, and the probabilities in a tuple.

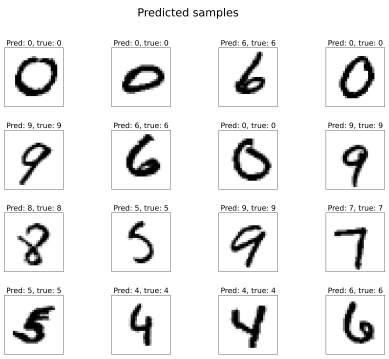
2.2 Training the CNN model

The dataset used for training the CNN model is the MNIST dataset. The device used is a CPU, the number of epochs is 5, and the batch size is 32. For the Adam parameters used, the learning rate is 0.001, and betas are $(0.9, 0.999)$

Based on the parameters set above, the CNN model predicted the samples from both the training dataset and the testing dataset. As shown in Figure 14, the predicted values and the true values of the testing dataset at the first batch and second batch are labeled. Indeed, for most of the batches, the accuracy even reached 100%.



(a) Predicted samples Batch 0



(b) Predicted samples Batch 1

Figure 8: The predicted samples of CNN model at batch 0 and batch 1. The predicted values and the true values are labeled above each subplot.

2.3 Error metrics during training

The loss function of the CNN model is shown below. As can be seen in Figure 9, the loss of the CNN model is higher at the beginning of the training. As the training continues, the loss of the model hugely decreases from 0.7 at the beginning to around 0.1.

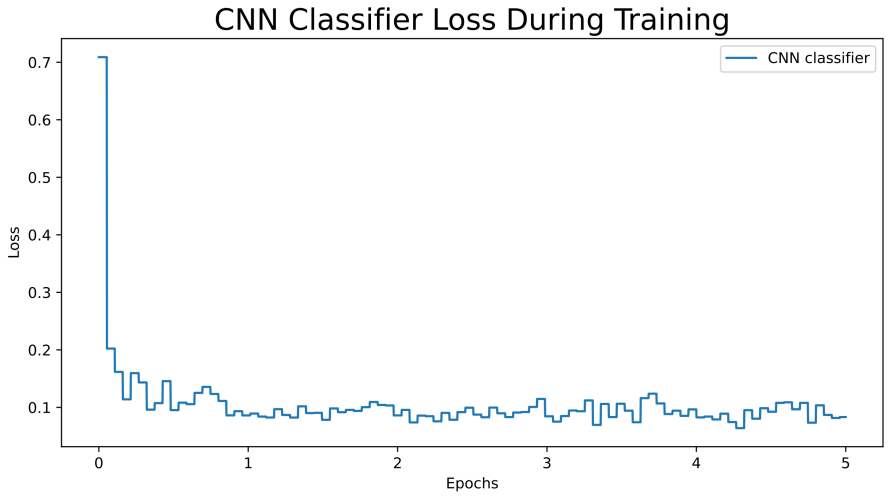


Figure 9: The loss function of the CNN model by epochs. The x-label is actually iteration numbers, but here it has been converted to epoch numbers. The plot is based on the average of every 100 iterations.

The accuracy of the CNN model during the training is presented in Figure 10. We take the same measure as introduced before, using the average values of each 100 iterations for a better interpretation. The accuracy of the CNN model during the training increases and reached more than 95% as the training goes on.

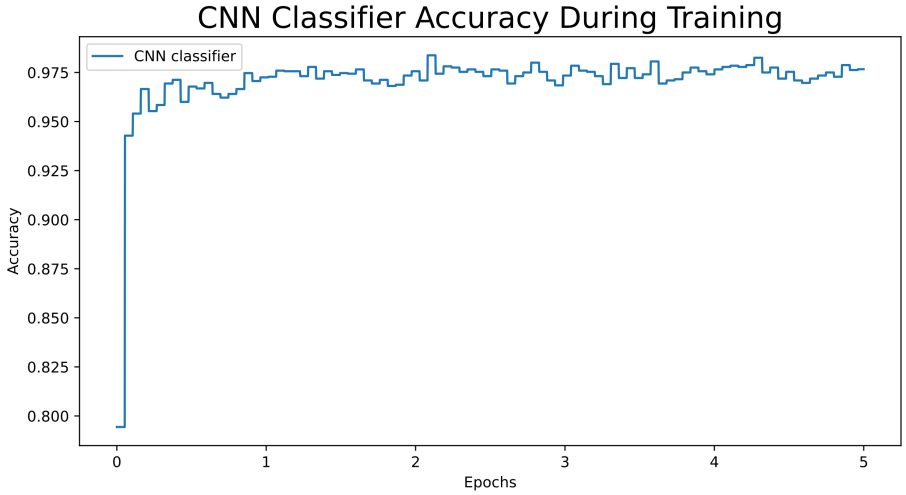


Figure 10: The accuracy of the CNN model by epochs. The x-label is actually iteration numbers, but here it has been converted to epoch numbers. The plot is based on the average of every 100 iterations.

The duration of the training for the CNN model is shown in Figure 11. The training time for the CNN model takes from 0.2 to 0.25 seconds on average.

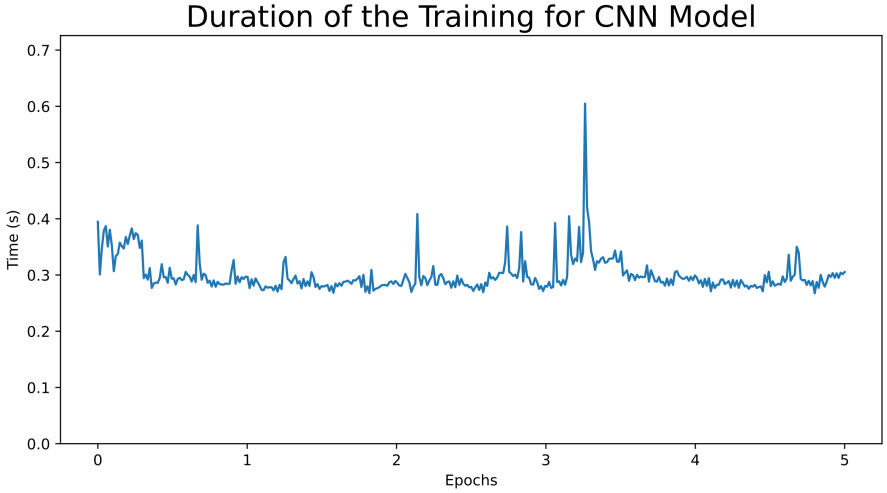


Figure 11: The duration of the CNN model training. The duration of the training is computed for every 25 iterations.

The accuracy of the CNN model on the training dataset is 0.9831, while the accuracy on the testing dataset is 0.9783. We can see that the model predicted the images well, even on the testing dataset.

2.4 Convolutional layers

Convolutional neural networks are used so widely in image classification. It is because the convolution layers reduce the high dimensionality of the images without losing too much information.

Depending on the convolution pattern, which is also a trained parameter, the model can discover the best image pattern that characterizes the labels individually and arrange convolution parameters to make a convolution pattern that highlights those patterns.

They are much better for image recognition than dense neural networks because they have fewer parameters, thus they are rapidly trained, and take advantage of the fact that data in the image is not randomly distributed to lose less accuracy.

2.5 Improve your CNN model

In order to increase the samples, we can use the Image Augmentation technique by rotating, shearing, or zooming the images when the data samples are not sufficient.[4]

For speeding up the training, we can also increase the learning rate.[5] If we are not using GPU for the computation, we can multiply the learning rate by several times to spend less time on training.

Training on images for which the experts have given confident ratings can be used to improve the generalizability.[6]

2.6 BONUS: CNN loss function

What we want to do with a classifier is to approximate the true probability distribution with a function which is our classifier.

The Kullback-Leibler Divergence is a function that measures the difference between two probability distributions: $d_{KL}(q,p) = \int q(y)(\log(q(y)) - \log(p(y)))dy$.

Let $p(y)$ be the true label distribution for the data, we need to find a classifier $q(y)$ that minimizes the divergence.

The best $p(y)$ is the one minimizing the cross-entropy.

Because the cross-entropy is easily approximated for supervised learning we generally use it for training.

3 Explore the latent space

To maintain consistency with the result here above in this report we use for this part 'gen5', 'dis5', and 'cnn_5_epochs' respectively as our generator, discriminator, and CNN classifier models; which are the models trained with 5 epochs.

3.1 Combine the generator and the classifier

Without changing the dimension of the latent space (dim_latent_space = 100) and batch size (batch_size = 32), a lot of triplets are generated with the function torch.rand().

For each consequent label, we calculate its label prediction probability (that will be utilized for the second step of this section).

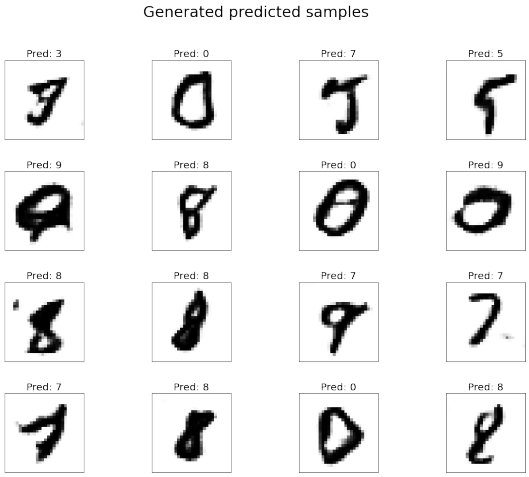


Figure 12: Generation of predicted images by labeled latent vectors randomly generated

As we can see in Figure 13, the quality of the images is the same as if they were written by a three years old child: we recognize the labels with difficulties.

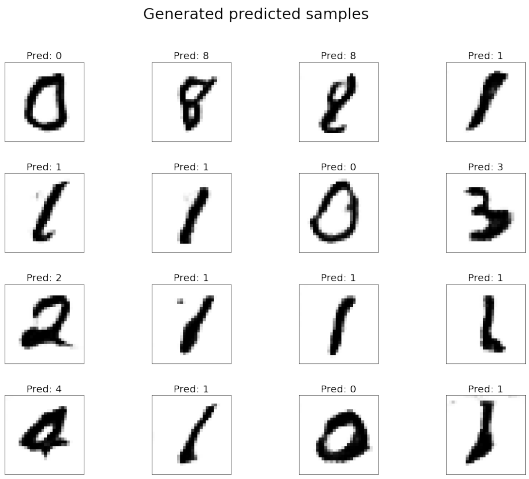
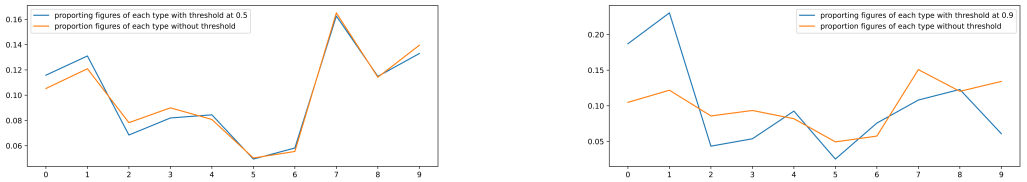


Figure 13: Generation of predicted images by labeled latent vectors randomly generated with a threshold at 0.95

If we increase the threshold the quality improves, confirming the hypothesis that the "certainty" of CNN is a good quality estimator

3.2 Distribution of classes

After generating the corresponding labeled latent vectors and plotting their distribution, we can analyze the representation proportion of each different class.



(a) Distribution of classes with threshold = 0.5

(b) Distribution of classes with threshold = 0.95

Figure 14: Proportion distribution of each class obtained from the generated labeled latent vectors, with different threshold values respectively

All the labels are represented, but the labels with high certainty are 45% of the time 0 or 1 and there are barely any 2 and 5.

3.3 Mean of your latent vectors

In the following section, by computing the mean value of each labeled latent vector with a certainty threshold of 0.95, we name a new list of vectors that we will call 'mean latent vectors' which are the mean of the vectors labeled with a high certainty, from which new images will be generated.

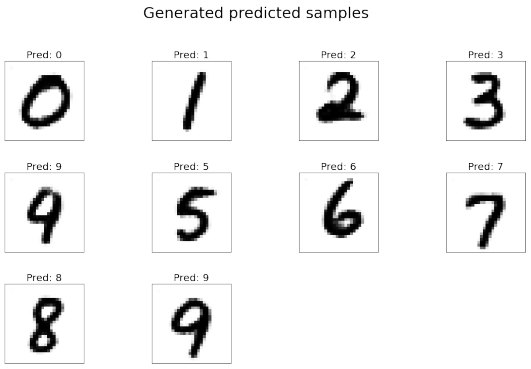


Figure 15: Generation images with the mean latent vectors

We see that the results are especially good, with the notable exception of the number 4. We can justify this situation by the fact that there are two possible handwriting ways to write down the 4, and consequently, the model made a mixture of both.

3.4 Norms and distances

3.4.1 Norm

A very surprising fact is that the mean of the norm of randomly generated vectors is much bigger (having a value of 9 on average) than the mean of the labeled mean vectors (whose value is 1 on average). That is as if the latent vectors of the same label were evenly spaced in each direction.

Our explanation for this scenario is that due to the random generation of latent vectors, during the training process, vectors in every single direction were associated with the same label.

3.4.2 MDM classifier

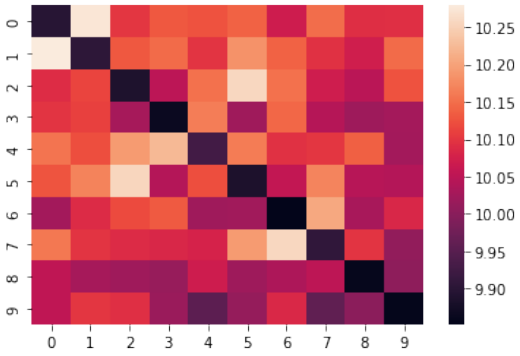


Figure 16: On the vertical axis the mean labeled vector, on the horizontal axis random labeled vectors: mean distance between random labeled vectors and mean labeled vector

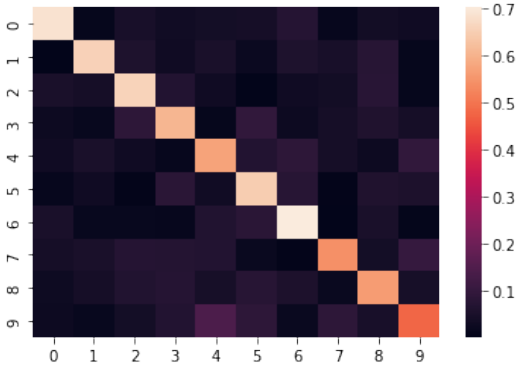


Figure 17: Percentage of random latent vectors classified by MDM on the vertical axis that are also classified by CNN on the horizontal axis

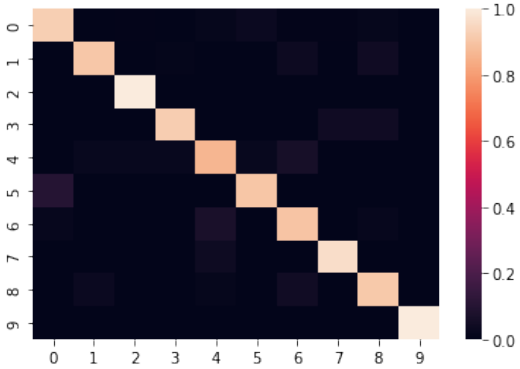
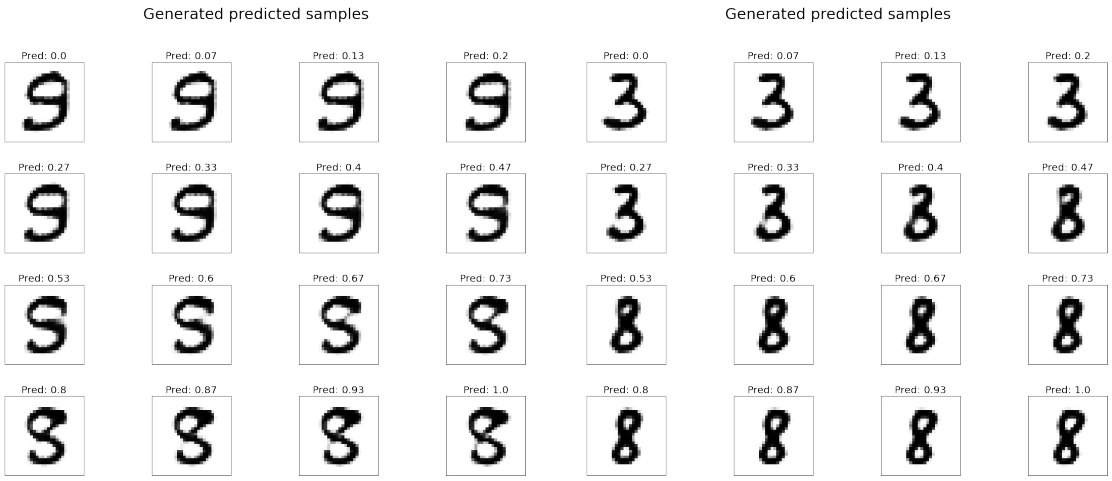


Figure 18: Percentage of random latent vectors with certainty threshold at 0.98 classified by MDM on the vertical axis that is also classified by CNN on the horizontal axis

For Figure 17 and Figure 18: pay attention to the scale on the left!

We see that the two classifiers agree more than 70% of the time even with low certainty. It is like there were defined clusters of the same label with the mean labeled vectors as centroïds. The running of the function described in section 3.7 with a very large "zoom" parameter seems to confirm this theory for very large latent vectors.

3.5 Interpolation for exploration



(a) Interpolation random 8 to random 9

(b) Interpolation mean feature vector 3 to mean feature vector 8

Figure 19: Interpolations

3.6 Noise for exploration

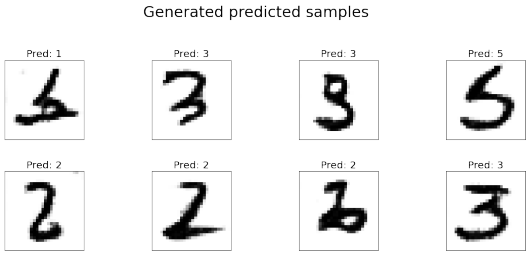


Figure 20: Mean feature vector label 3 with random Gaussian noise of std = 0.9

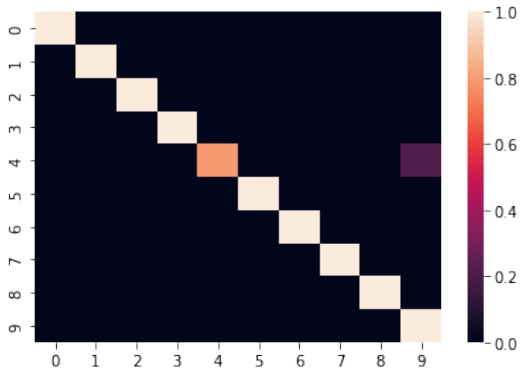


Figure 21: Percentage of random latent vectors plus a gaussian noise with a standard deviation of 0.1 with true label on the vertical axis that is classified by CNN with label on the horizontal axis

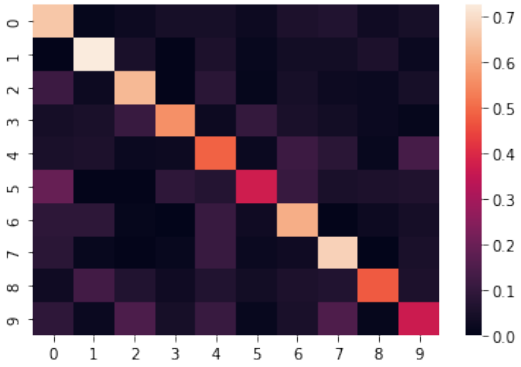


Figure 22: Percentage of random latent vectors plus a gaussian noise with a standard deviation of 0.9 with true label on the vertical axis that is classified by CNN with label on the horizontal axis

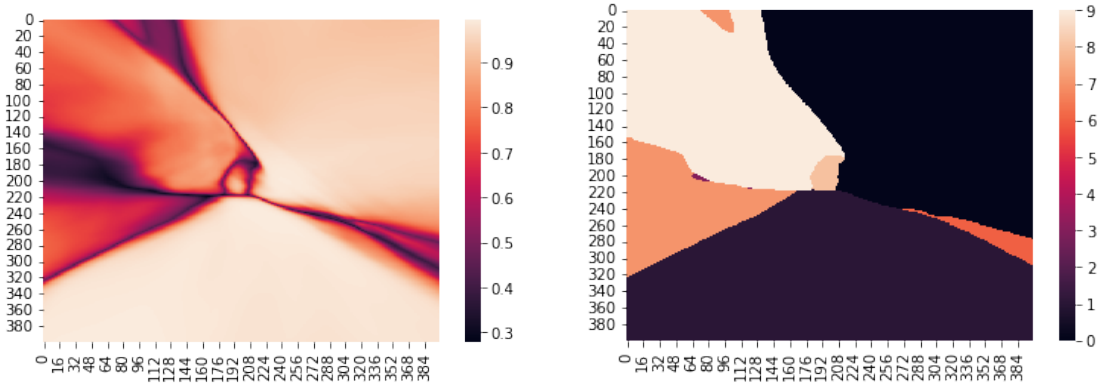
3.7 BONUS: Mode collapse

This part of the report does make reference to the last function at the end of the s̈ection3.ipynb; where we created a simple and brief algorithm that performs an exploration of a 2D plane in the latent space, which does show you a figure of the label's certainty and the label corresponding to the vectors located in those regions in the plane.

We highly encourage you to play with it and change the parameters as you wish.

To be good at fooling, the discriminator of gan, our classifier had to produce pictures with a clear label from random feature vectors. Nevertheless, because our classifier is continuous, it did have to produce intermediate pictures to go from one label to another. And due to its training with random label vectors, it will not put all the latent vectors with the same label on a specific determined side in the latent space.

What we think it can and will do is try to make the intermediate states of low uncertainty, the ones with no clear label, as little as they can. We can thus make the assumption that if we check the latent space we will have large plateaus of high certainty with squeezed cliffs of low uncertainty, and using our algorithm it is exactly what we observe.



(a) Certainty

(b) Label

Figure 23: 2D plane in the latent space

with parameters:
vec1 = mean_label[8]
vec2 = mean_label[1]
vec3 = mean_label[0]
quality = 400
dezoom = 5

include0 = True

4 Create what you want

For this section, please refer to the notebook "section_4.ipynb" and the "README" file.

References

[1] TORCH.NN, *PyTorch documentation*.

[2] TRAINING A CLASSIFIER, *PyTorch documentation*.

[3] How to Explore the GAN Latent Space When Generating Faces, *Machine Learning Mastery*.

[4] Improving Performance of Convolutional Neural Network!, *Medium*.

[5] 7 tricks to speed up the training of a neural network, *analyticsindiamag.com*.

[6] Kaveri A. Thakoor, Xinhui Li, Emmanouil Tsamis, Zane Z. Zemborain, Carlos Gustavo De Moraes, Paul Sajda, Donald C. Hood *Strategies to Improve Convolutional Neural Network Generalizability and Reference Standards for Glaucoma Detection From OCT Scans*. Trans. Vis. Sci. Tech. 2021;10(4):16.