LINMA2472

homework 3 GAN and CNN

Julien Thonnard 06441800
Corentin Vermeulen 25271800

1 Train a GAN on MNIST for image generation

1.1 The GAN model

A Generative Adversarial Networks (GAN) is composed of two networks. The first one is the **generator** that aims at generating the target data from a latent vector, in our case it generates an image of a digit if the dataset is MNIST or an image of a letter is the dataset is EMNIST. We did not modified the reference architectures.
The **generator**'s architecture is the following:

First the input layer with input size: [batch size (32) ,latent_vector size (1,100)];
Then we have the hidden layers:

Hidden Layer	Activation	N parameters	Outputs
Linear		316,736	[Batch_size, 1, 3136]
2D transposed convolution		36,928	[Batch_size, 64, 14, 14]
Batch normalization	Relu	128	[Batch_size, 64, 14, 14]
2D transposed convolution		18,464	[Batch_size, 32, 14, 14]
Batch normalization	Relu	64	[Batch_size, 32, 14, 14]
2D transposed convolution		4,624	[Batch_size, 16, 14, 14]
Batch normalization	Relu	32	[Batch_size, 16, 14, 14]
2D transposed convolution		145	[Batch_size, 1, 28, 28]

Finally the Output layer with hyperbolic tangent activation that outputs [batch size (32), image_size(28,28)].

The second network of the GAN is a discriminator which aims at differentiating real images coming from the data set and fake images coming from the generator.
The **discriminator**'s architecture is the following:

First the input layer with input size: [Batch_size (32), image_size(28,28)]
Then the hidden layers

Hidden Layer	Activation	N parameters	Outputs
2D convolution	Relu	320	[Batch_size, 32, 14, 14]
Dropout2d		0	[Batch_size, 32, 14, 14]
2D convolution	Relu	18,496	[Batch_size, 64, 14, 14]
Dropout2d		0	[Batch_size, 64, 14, 14]
2D convolution	Relu	73,856	[Batch_size, 128, 14, 14]
Dropout2d		0	[Batch_size, 128, 14, 14
2D convolution	Relu	147,584	[Batch_size, 128, 7, 7]
Dropout2d		0	[Batch_size, 128, 7, 7]

Finally the linear output layer with 6273 parameters which outputs [Batch_size (32),1].
We used the same architectures for MNIST and EMNIST data.

1.2 Training the GAN

We started to train our model on the MNIST database because it is smaller in size and has less classes than EMNIST. Indeed, MNIST contain only 70000 images of 10 classes and EMNIST contains more than 145000 images of 26 classes.
Nevertheless, we also trained and tested a second model on the EMNIST data set to produce our final messages for the second part of this project.

We ran the code on Google Colab to use the power of available GPU and increase the efficiency of the model. In term of parameters, we choose to keep the reference parameters:

- Batch size = 32
- Number of epoch = 5 for MNIST and 10 for EMNIST
- Adam parameters: lr_gan = 0.0002 and betas_gan = (0.5, 0.999)

Below we can see on the first line some generated digits and the discriminator decision as title. The second line are images from the training data set. In this case the discriminator is 100% right. We can clearly see a difference between the generated digits and the real ones even if we can deduce the class of it.

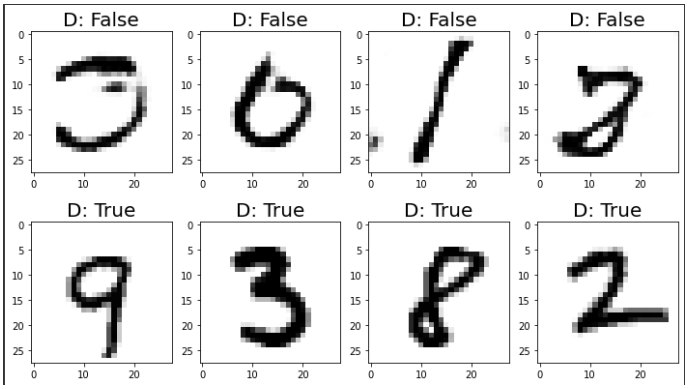

Figure 1: First line are fake images, second line are images from training data

1.3 Error metrics during training

On the figure below we observe that the training loss and accuracy stabilize after two epochs. The accuracy of the discriminator for test data is around 0.3 which is not that bad since a perfect GAN would lead to a 0.5 discriminator accuracy.
Not many variation is observed after the second epoch therefore we could stop the training after three epochs if we want to spare some training time. Indeed each epoch took around 45 seconds with Google's GPU to execute on MNIST data set (total training time for 5 epochs = 215 seconds).

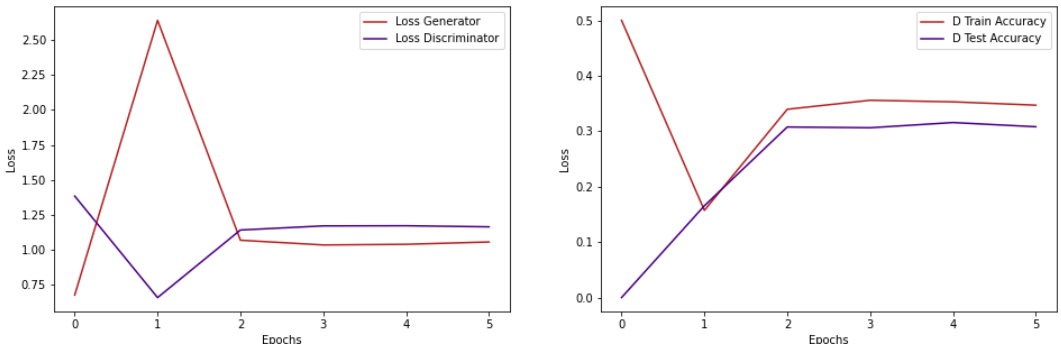

Figure 2: Evolution of the Loss and Accuracy functions at each epoch on MNIST

1.4 Improve your GAN model

As we say above, the train and test isn't very fast and the accuracy could be improved. So it may be interesting to tune our model.

- First tuning parameter: Number of epoch.
  If we reduce the numbers of epochs it will decrease the running time but won't significantly decrease the quality of the results. As we can see on the plots the training is quite constant after 2-3 epochs.
- Second tuning parameter: Number of hidden layers.
  If we want to go faster we could also reduce the number of hidden layers which may lead to a decrease in the accuracy and the generation quality. On the other hand we could increase the number of hidden layers to get better results.
- Third tuning parameter: Batch size.
  Increasing the batch size will decrease the variance between the learning gradients.

2 Train a CNN on MNIST for image classification

2.1 Your CNN model

The CNN aims at finding for a given image the right label (digit class for MNIST or letter class for EMNIST).
We used the reference architecture which is the following:
The input layer takes an input size of [batch_size (32) , image_size (28,28)].
Then the hidden layers:

Layer	N parameters	Outputs
Conv2d-1	416	[32, 16, 28, 28]
ReLU-2	0	[32, 16, 28, 28]
MaxPool2d-3	0	[32, 16, 14, 14]
Conv2d-4	12,832	[32, 32, 14, 14]
ReLU-5	0	[32, 32, 14, 14]
MaxPool2d-6	0	[32, 32, 7, 7]

Finnaly a linear output layer with 15,690 parameters which output a [batch_size , 10] which correspond to the probabilities to be a given class for each batch element.

2.2 Training your CNN

As for the GAN we first trained the model on the MNIST data set for the same reasons as in 1.2. Later we trained another model with same architecture on the EMNIST data set to be able to produce the final message.
We ran the code on Google Colab to use the power of available GPU and increase the efficiency of the model. In term of parameters, we choose to keep the reference parameters for the CNN:

- Batch size = 32
- Number of epoch = 15
- Adam parameters: lr_gan = 0.01 and betas_gan = (0.9, 0.999)

2.3 Error metrics during training

As we see on the figure below, the accuracy increases quickly on the first 3 epochs, then it starts to stagnate around 0.95.
At the same time the loss decreases rapidly and stagnates after 3 epochs. We observe that the training and testing lines are quite close for the accuracy. First we thought that the model was also training on the testing data but even after checking that the `model.train()` and `model.eval()` function were well present we still got the same results.

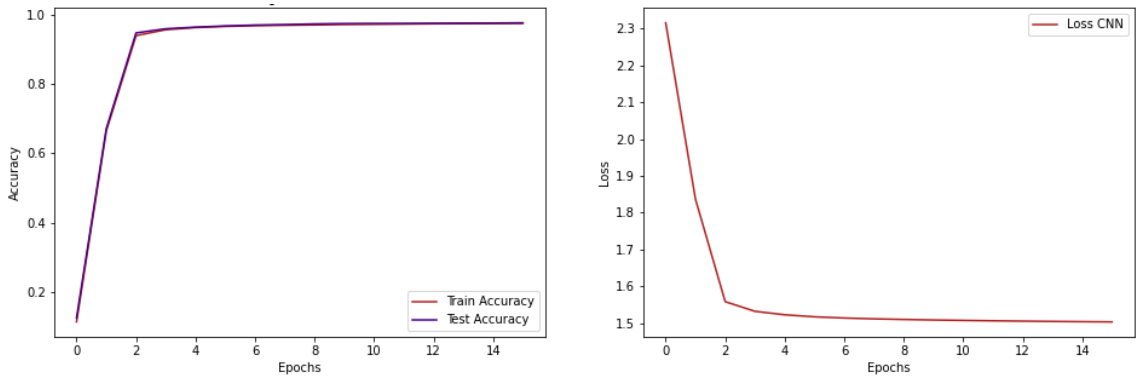

Figure 3: Evolution of the Accuracy and Loss functions at each epoch for MNIST data set

2.4 Convolutional layers

Convolutional neural networks are widely used in image classification because they allow to do a classification quickly take the image structure into account.
Those models use convolutional layers in their architecture. They can be seen as layers of neurons who read the image in input. Their function is to simplify the image to reduce his size while keeping some relevant structure. Those layers are preferred to dense linear layers because they focus not only on one pixel as a dense linear layer would do but it also take the neighbour of the pixel into account. So in the end, they have a more local approach where dense linear layers have a global approach.

GAN with convolutional layers are called Deep Convolutional Generative Antagonist Network (DCGAN).

2.5 Improve your CNN model

The techniques we can use to improve some criterion in our CNN model are very similar with those we propose in the section 1.4.

- First tuning parameter: Number of epoch.
  Our model don't need 15 epoch to train so decrease this number will also decrease our running time. As we see for the GAN model, 2-3 epochs is enough to have a constant value as it is show in the plots.
- Second tuning parameter: Number of hidden layers.
  We could increase or decrease the number of hidden layers to have better results or to go faster as we said before.
- Third tuning parameter: Batch size.
  Increasing the batch size will decrease the variance between the learning gradients.

3 Explore the latent space

3.1 Combine your generator and your classifier

We had here to generate latent vectors to produce images with the trained generator and then to classify the generated images with the CNN. We got acceptable results for the digit generation but the class is not obvious for all the images. For the letters generation it was more complicated. Indeed the EMNIST data set contains lower case and upper case letter which deacrease the precision of the letters generation.
The probability threshold for selected images was 0.995. Lower threshold gave uglier results.

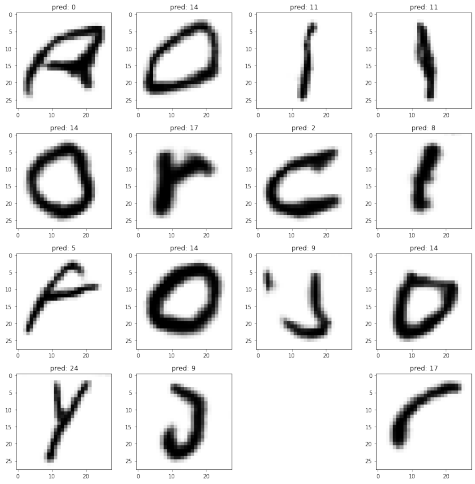

Figure 4: Generated letters, probability threshold=0.995

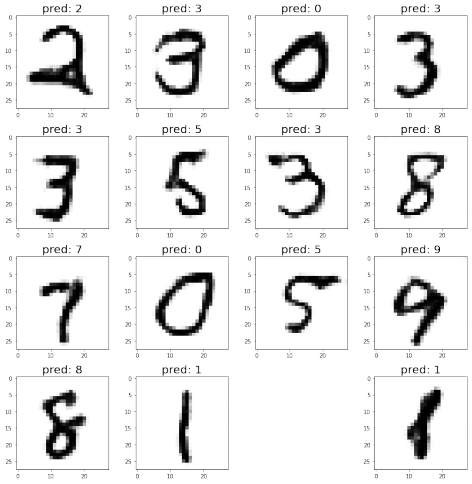

Figure 5: Generated digits, probability threshold=0.995

3.2 Distribution of classes

For the letters, before the selection with the treshold, all the classes are represented but not equally. The M-class and W-class are the two least represented classes.On the other hand, C, L, O, R and Y are the most represented classes. It seems that the algorithm has more facilities to represent those classes than the two others. After the selection, we don't have M anymore and only a few W and Z. But B, O and Y are still the most represented classes. In global the distribution between the classes are the same than before the selection

For the digits, it's quite different. The distribution before the selection is almost uniform with the least represented class for the 6 and the most represented class is the 8. However, the distribution is totally different after the selection. The 8 is still the most represented digit but the 2 is also very well represented. For the least represented value we have still the 6 but the 4 is also badly represented. The distribution changes after the selection.

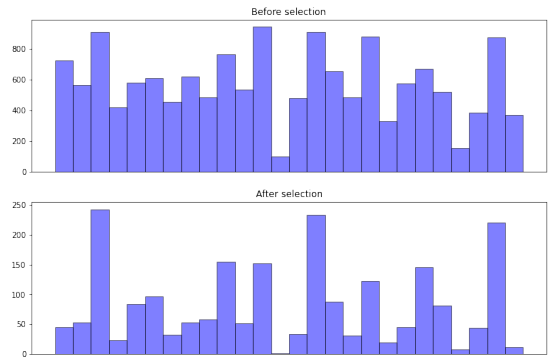

Figure 6: Distribution between classes for EMNIST

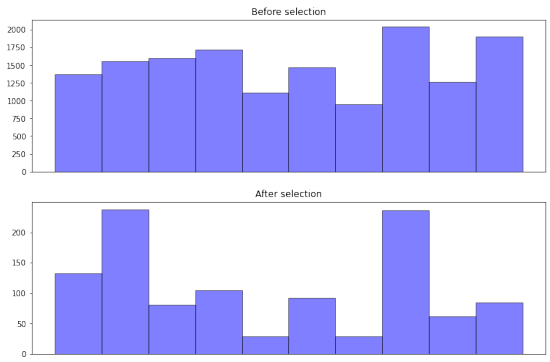

Figure 7: Distribution between classes for MNIST

3.3 Mean of your latent vectors

The results with the mean of the latent vector are better. It seems logical because we take the mean of the vector (with good classification probability) within the same class. Only D and M are still not well represented but as we say before, they are not well represented after the selection with the threshold.

Figure 8: Generates means letters

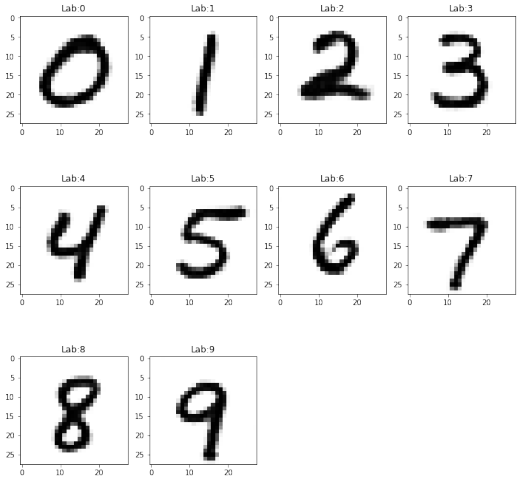

Figure 9: Generates means digits

3.4 Norms and distances

When we compute the norm of the mean latent vectors we always get values around 2 or 3. The others random latent vectors have norms around 9 or 11.
We then observe a big gap between them. We can explain this by the fact that the mean vector are the mean of the selected vectors after the probability vector and that only some features of the latent vectors are important to produce a digit. For example to produce a 1 we have a vector full of 0 and only some higher values in on the 40-50 and 80-90 features. The norm of such a vector will be smaller since most of the features are close to zero.

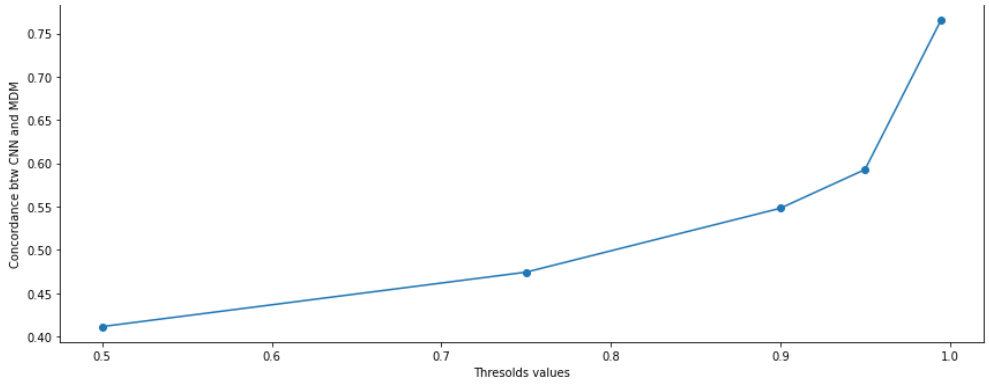

Figure 10: Evolution of the concordance between CNN and MDM with different threshold for MNIST

On the figures above we see that CNN and MDM agree only on highly probable classification. As the certainty increase, the two methods agree on the classification.

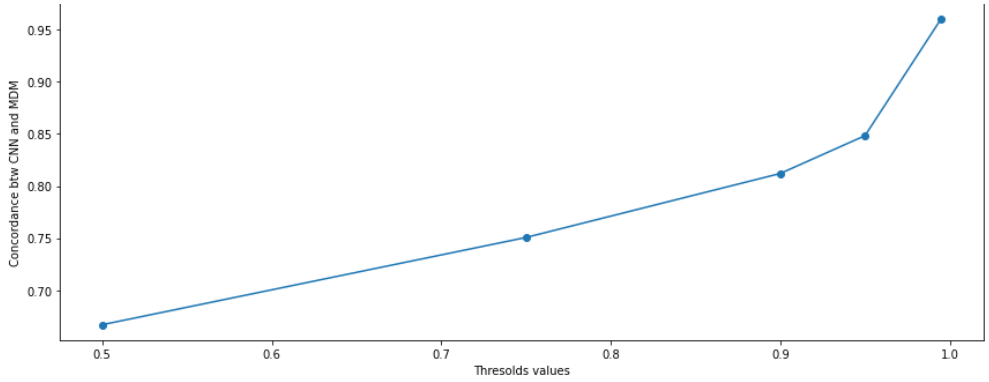

Figure 11: Evolution of the concordance between CNN and MDM with different threshold for EMNIST

3.5 Interpolation for exploration

To make linear interpolation we just used the following formula: $(((n-t)/(n-1))*X_0+((t-1)/(n-1))*X_N)$ where n is the number of points we want in the interpolation, t is the index of the interpolated point, X_0 is the first point and X_N is the last point. We generalize this for a vector of m dimension. We see the results below with n=10. You will find in the folder animated .gif about these interpolations which are easier to visualize.

3.5.1 Within the same class

We use X_0 and X_N vectors from the same class.

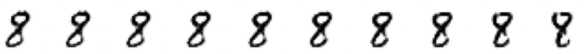

Figure 12: Interpolation between two "8"

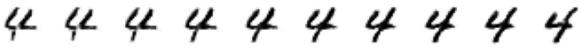

Figure 13: Interpolation between two "4"

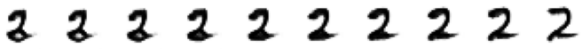

Figure 14: Interpolation between two "2"

3.5.2 Between different classes

We used X_0 and X_N vectors from different classes. You will find in the folder an attached .gif with interpolation going from 0 to 9 passing through each digit.

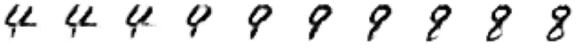

Figure 15: Interpolation from a 4 to an 8

3.6 Noise for exploration

Noise is interesting if we want to combine multiple generated images in order to produce a text. In that case we don't want all the images with same label to be the same images, we want some variation. The variation induced with a sigma of 0.1 is not notable and 0.7 and 0.9 are to high, the produced images loose in quality. Therefore we decided to choose a variance of 0.3.

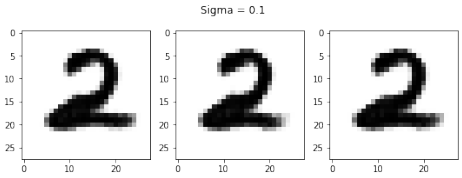

Figure 16: Sigma = 0.1

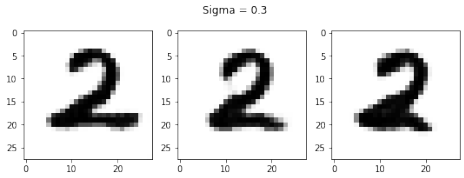

Figure 17: Sigma = 0.3

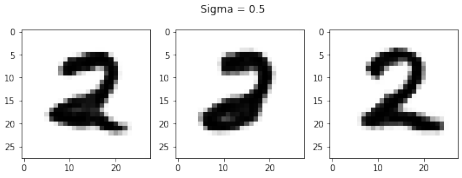

Figure 18: Sigma = 0.5

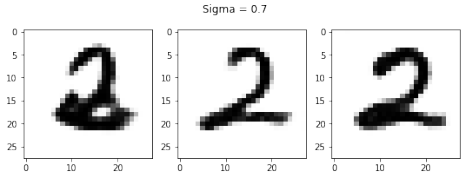

Figure 19: Sigma = 0.7

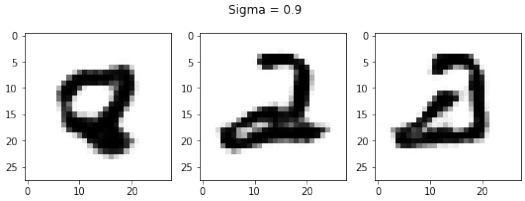

Figure 20: Sigma = 0.9

4 Create what you want

4.1 Mandatory: Open message

Here you can find the message we decided to create with the algorithm. The idea behind it was to simplify the correction by offering you directly a final grade for this project. I hope you enjoy the effort.
It is necessary to note that the two databases didn't contain some of the characters we used as the "\". The solution we decided to use is to put the L letter instead.

Figure 21: Wished grade for Julien

Figure 22: Wished grade for Corentin

Moreover, as you may have noticed, the title and other elements of the report have been directly written by the algorithm in order to link the report with its content.