Shelly Gamlielly
Yerus Mandfro

Intro to Deep Learning Ex3

Practical Part

Transfer Learning

Describe the test accuracy as well as the architecture of the MLP you ended up using.

Minimal Classifier

This classifier was provided to us, it has only one fully connected layer, to classify an image into one of 10 classes.

Next, we describe the test accuracy using the encoder with fixed length. Meaning, we train only the minimal classifier, and use the latent vector that the encoder extracts as input to the classifier.

Encoder fixed weights

training accuracy - 100%
test accuracy - 71%

The classifier has the highest test f1 score on the digit 1 - 0.91
The classifier has the lowest test f1 score on the digit 5 - 0.38
The classifier shows behavior of overfitting.

Train

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         7
           2       1.00      1.00      1.00         4
           3       1.00      1.00      1.00         7
           4       1.00      1.00      1.00         4
           5       1.00      1.00      1.00         4
           6       1.00      1.00      1.00         5
           7       1.00      1.00      1.00         4
           8       1.00      1.00      1.00         4
           9       1.00      1.00      1.00         7

    accuracy                           1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50
```

Test

```
              precision    recall  f1-score   support

           0       0.75      0.86      0.80       980
           1       0.85      0.98      0.91      1135
           2       0.67      0.64      0.66      1032
           3       0.69      0.83      0.76      1010
           4       0.56      0.75      0.64       982
           5       0.57      0.29      0.38       892
           6       0.74      0.82      0.78       958
           7       0.78      0.76      0.77      1028
           8       0.81      0.59      0.68       974
           9       0.62      0.50      0.55      1009

    accuracy                           0.71     10000
   macro avg       0.70      0.70      0.69     10000
weighted avg       0.71      0.71      0.70     10000
```

Now, let's compare to the results when training the classifier with an encoder architecture.

Composition of encoder and smaller MLP

training accuracy - 100%
test accuracy - 68%

This model has lower accuracy compared to the model with the encoder fixed weights.

The classifier has the highest f1 score of test on the digit 1 - 0.87
The classifier has the lowest f1 score of test on the digit 5 - 0.44

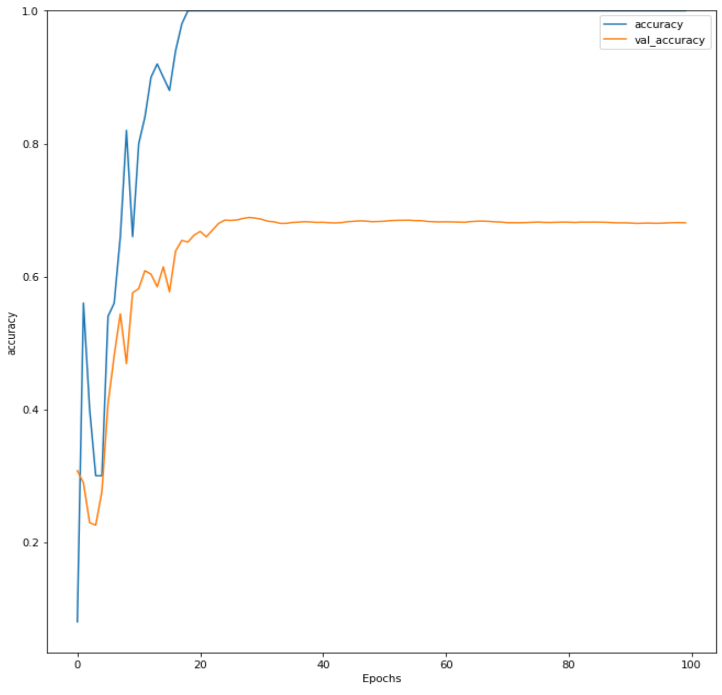
Train

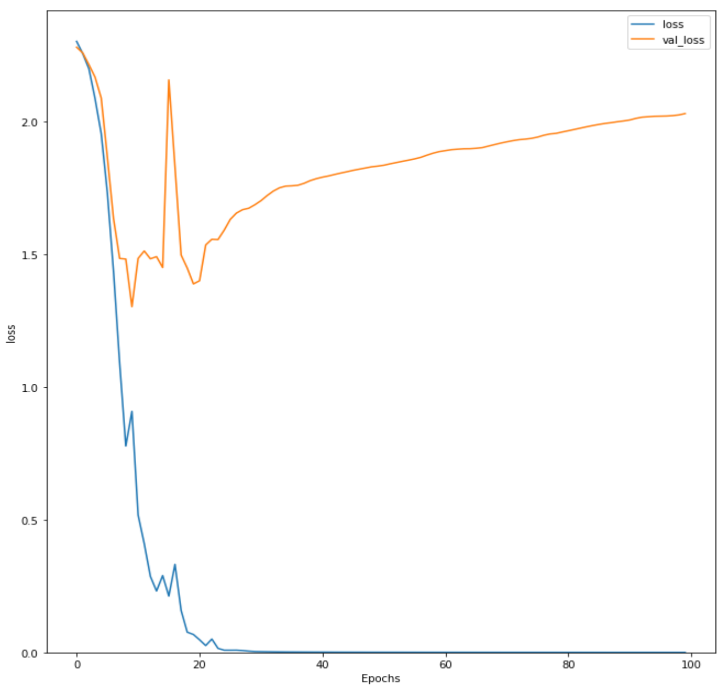
```
             precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         7
           2       1.00      1.00      1.00         4
           3       1.00      1.00      1.00         7
           4       1.00      1.00      1.00         4
           5       1.00      1.00      1.00         4
           6       1.00      1.00      1.00         5
           7       1.00      1.00      1.00         4
           8       1.00      1.00      1.00         4
           9       1.00      1.00      1.00         7

    accuracy                           1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50
```

Test

```
              precision    recall  f1-score   support

           0       0.86      0.68      0.76       980
           1       0.80      0.96      0.87      1135
           2       0.55      0.63      0.59      1032
           3       0.72      0.80      0.76      1010
           4       0.67      0.76      0.71       982
           5       0.59      0.35      0.44       892
           6       0.67      0.77      0.72       958
           7       0.65      0.75      0.69      1028
           8       0.73      0.43      0.54       974
           9       0.59      0.60      0.60      1009

    accuracy                           0.68     10000
   macro avg       0.68      0.67      0.67     10000
weighted avg       0.68      0.68      0.67     10000
```

The graphs below show the accuracy and loss of the training set and test set.
Notice that there is overfitting - the train loss decreases with each epoch, the test loss decreases at the first epochs, but then increases from epoch 25.
Moreover, the test accuracy does not improve from epoch 40.



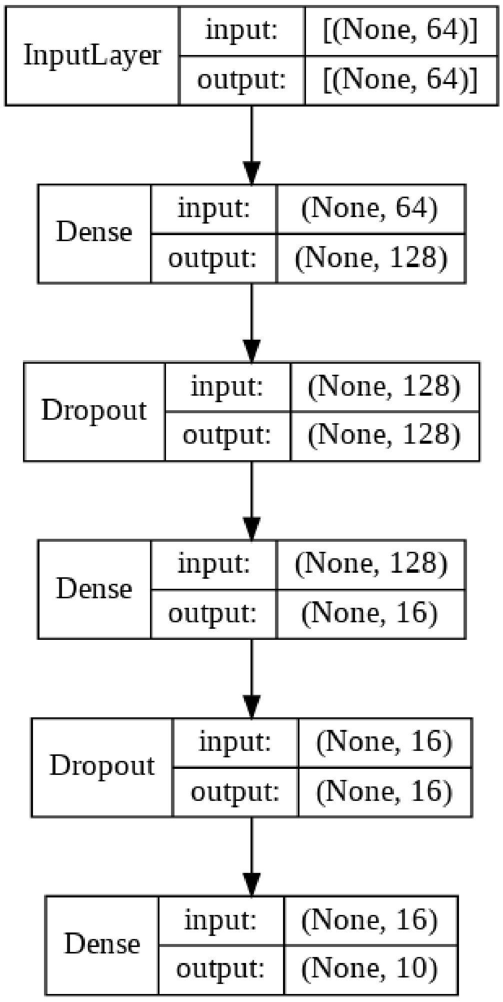


Small MLP

The input of the network with the fixed weights of the encoder is *train_code* which represents the latent space created by the encoder. The dimension of the latent space is 64, therefore the input shape is 64. We use the pre-trained encoder to extract important features from the image, as mentioned the features vector has a lower dimension - 64. And according to those features the classifier predicts which digit the vector represents.

We used drop up layers to avoid overfitting and used EarlyStopping to avoid overfitting/underfitting. Since too many epochs can lead to overfitting of the training dataset, and too few may result in an underfit model. Early stopping allowed us to specify an arbitrary large number of training epochs (100) and stop training once the model performance stops improving on the test dataset.

The architecture we chose is shown below:



Encoder fixed weights

Train accuracy - 100%
Test accuracy - 72%

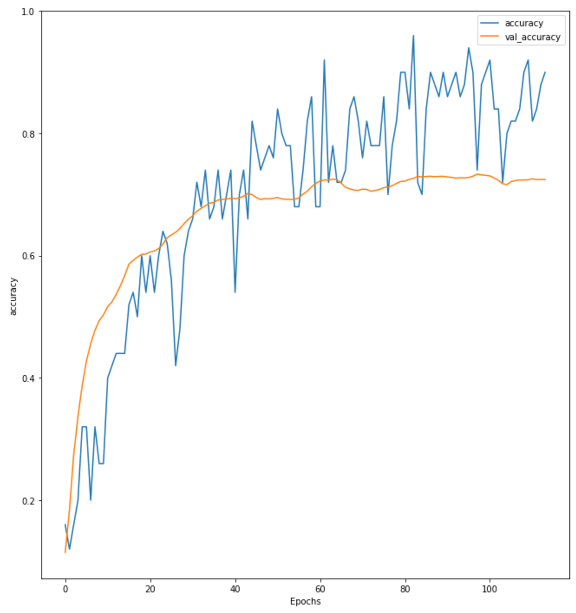
The classifier has the highest f1 score of test on the digit 1 - 0.85
The classifier has the lowest f1 score of test on the digit 5 - 0.52

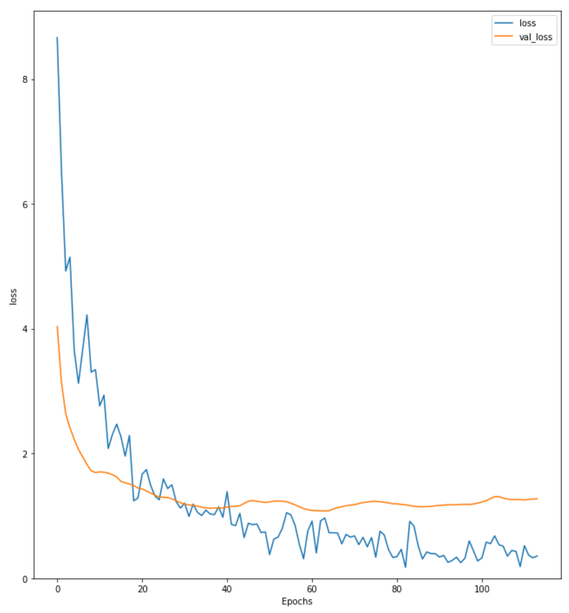
Train

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         7
           2       1.00      1.00      1.00         4
           3       1.00      1.00      1.00         7
           4       1.00      1.00      1.00         4
           5       1.00      1.00      1.00         4
           6       1.00      1.00      1.00         5
           7       1.00      1.00      1.00         4
           8       1.00      1.00      1.00         4
           9       1.00      1.00      1.00         7

    accuracy                           1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50
```

Test

```
              precision    recall  f1-score   support

           0       0.83      0.78      0.81       980
           1       0.75      0.99      0.85      1135
           2       0.68      0.66      0.67      1032
           3       0.67      0.84      0.74      1010
           4       0.58      0.76      0.66       982
           5       0.75      0.40      0.52       892
           6       0.70      0.87      0.77       958
           7       0.88      0.72      0.79      1028
           8       0.85      0.59      0.70       974
           9       0.68      0.56      0.62      1009

    accuracy                           0.72     10000
   macro avg       0.74      0.72      0.71     10000
weighted avg       0.74      0.72      0.72     10000
```





As you can see in the graphs above that described accuracy and loss of test set and train set, the loss of training set and test set decreases.

Composition of encoder and smaller MLP

This time we set up a classification network that is equivalent to the composition of the encoder and our smaller MLP and train it using the same number of labeled training examples (50). Once again the test accuracy is low compared to the model with fixed encoder weights. The reason for this difference could be the fact that we update the encoder weights with the classifier, since now they are combined.

This model achieves test accuracy of 60, after 36 epochs.
We used keras's Early Stopping, so if after 20 epochs the test loss doesn't improve, we stop training.

train accuracy - 100%
test accuracy - 60%

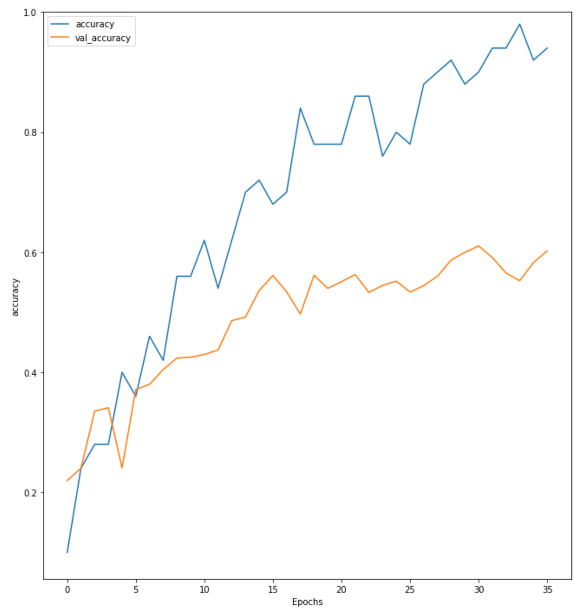
The classifier has the highest f1 score of test on the digit 1 - 0.83
The classifier has the lowest f1 score of test on the digit - 0.31

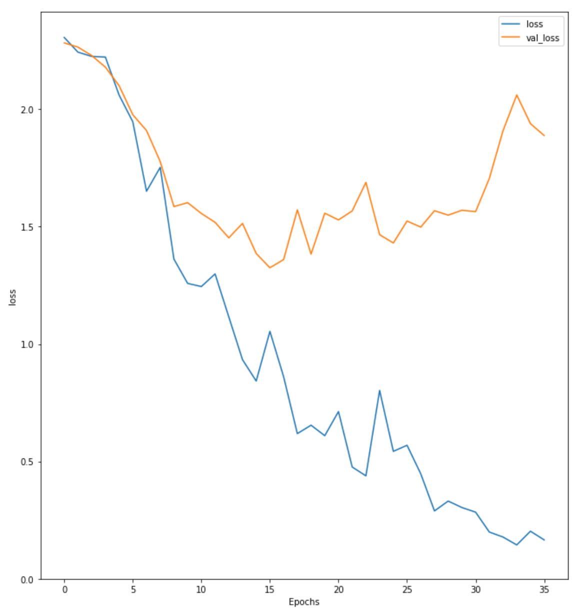
Train

```
             precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         7
           2       1.00      1.00      1.00         4
           3       1.00      1.00      1.00         7
           4       1.00      1.00      1.00         4
           5       1.00      1.00      1.00         4
           6       1.00      1.00      1.00         5
           7       1.00      1.00      1.00         4
           8       1.00      1.00      1.00         4
           9       1.00      1.00      1.00         7

    accuracy                           1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50
```

Test

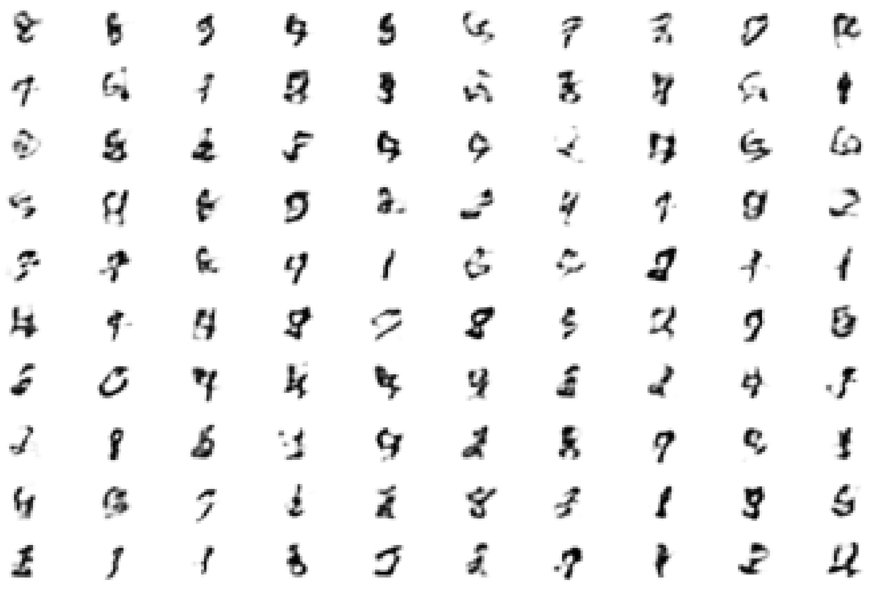
```
              precision    recall  f1-score   support

           0       0.77      0.67      0.72       980
           1       0.73      0.98      0.83      1135
           2       0.69      0.46      0.55      1032
           3       0.74      0.86      0.80      1010
           4       0.40      0.74      0.52       982
           5       0.49      0.32      0.39       892
           6       0.52      0.86      0.65       958
           7       0.61      0.48      0.54      1028
           8       0.81      0.20      0.31       974
           9       0.55      0.38      0.45      1009

    accuracy                           0.60     10000
   macro avg       0.63      0.60      0.58     10000
weighted avg       0.63      0.60      0.58     10000
```





Generative Adversarial Networks (GAN)

Show several sampled digits:



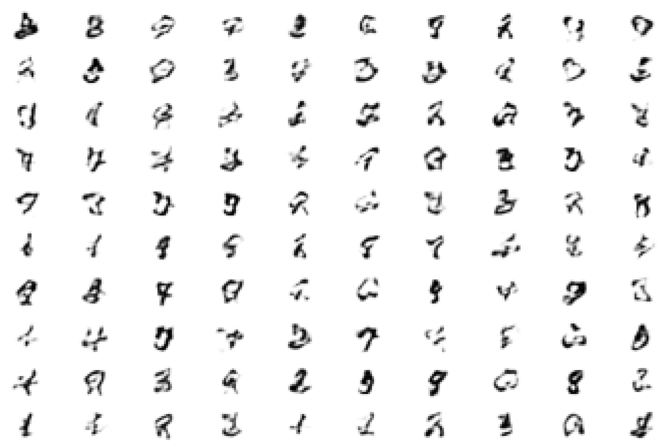
The generated images look reasonable, the model creates mostly valid digits that are close to reality. Compared with images produced by the AE, the images are more creative.

The dimensions of the digits are 32x32 - meaning 1024, reducing the dimension to 64 for AE or 32 for GAN has a very powerful impact.
The decoder could recustract 'well any digit from latent space it receives, if the digits are separated there. In the case of AE The encoder is expected to do the clustering , and when training the generator we want to create a separable latent space for digits.
Notice that during the training phase the AE gets as input an image from the training set with some gaussian noise, and outputs a restoration of the original noise-free image. While the GAN gets some random noise , and generates a real image from it.
The autoencoder is meant to learn an encoding network and decoding network. When an image is given to the encoder, it reduces the image to a compressed encoded form, which is then fed to the decoder. The encoder learns to find a compressed form of the input information, and the decoder learns to reconstruct the input from the encoded form. For certain generative problems GAN's are able to give more "realistic" outputs. Intuitively, GAN learns how to make an image look real in general, rather than how to memorise a set of images with the greatest accuracy.

In our case, all the decoder should do is to reconstruct the original image from the features vector it gets. The AE uses it's encoder to create this features vector and the decoder to reconstruct the original image. This is much easier than what the GAN is expected to do. The MLP generator's inputs are gaussian distributed random points from the latent space. Only the discriminator gets the images from the training set so it could predict whether this image is real or generated by the generator. The generator learns to generate real images by fooling the discriminator.

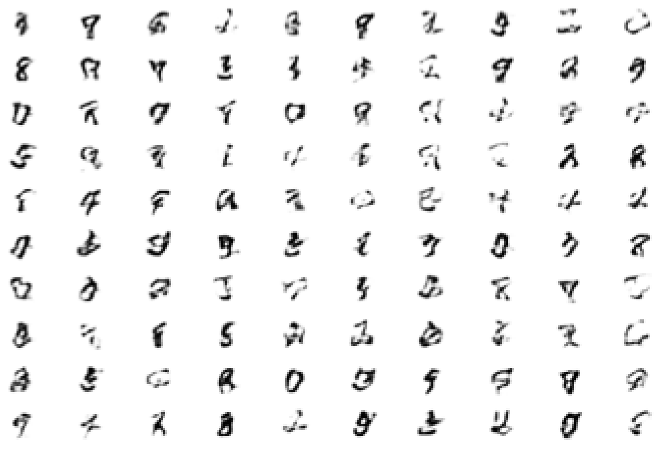
Below there are plots during the training of the GAN model ,as we can see it learned to generate different digits.

100 epochs :



80 epochs

60 epochs



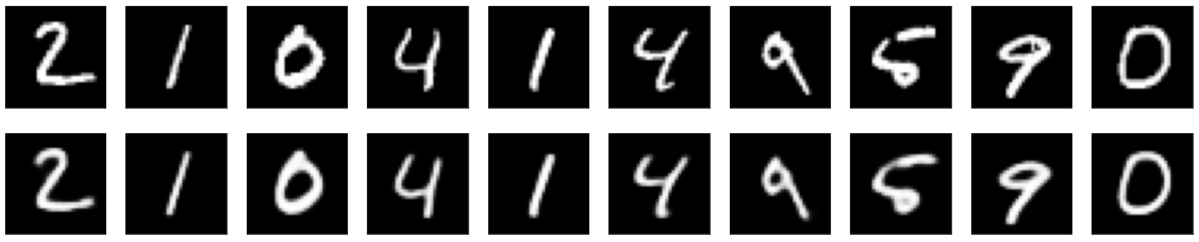
20 epochs

- Compare Generator with last Conv. layer and tanh activation and without :

40 epochs

40 epochs with tanh - no improvement

NeIn order to compare the GAN results with the AE, we plotted AE images where the upper row is the original images, and the lower are the reconstructed images:

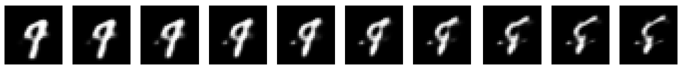


Perform interpolation

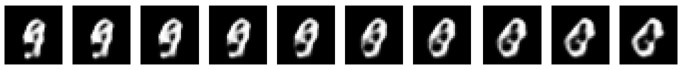
Show the results you get when interpolating in the AE's latent space compared to the ones in the GAN's latent space. Explain which strategy produces better results and why.

GAN:

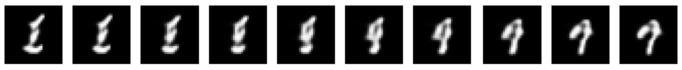
9->5



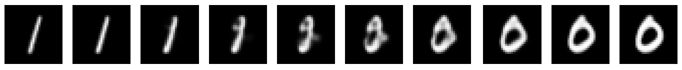
9->0



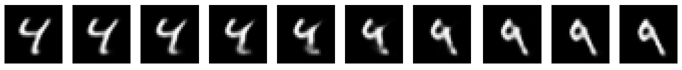
1->9



AutoEncoder
1->0



4->9



At first sight it seems like the autoencoder has better results. The images generated by the AE are more sharp and have a real structure of digits. The GAN images have a similarity to some digits but it is not very clear. Also the GAN sometimes confuses between digits (like 7 and 1), that could suggest that the digits on the GAN's latent space are not completely separated.Sp the AE images reconstructed from the encoded vectors of features performs better clustering compared to the generator.

On the other hand, GAN generates images from noise, so the images are more creative. The AE images generated from encoded vectors so they have more in common with the original images.

When plotting the interpolation between 2 different digits for example 1 and 0, we can see that at the middle of the AE interpolation there is a weird structure which does not match any digit. When examining the GAN interpolation we can see that all images between the right digit and the left digit match to a digit that the generator constructs. Notice that the generator had never seen any real image, it relies on the assumption that if it fools the discriminator then it is real.

Digit-Specific Generator

Describe your choice and show the variability of the images you get for a particular digit.

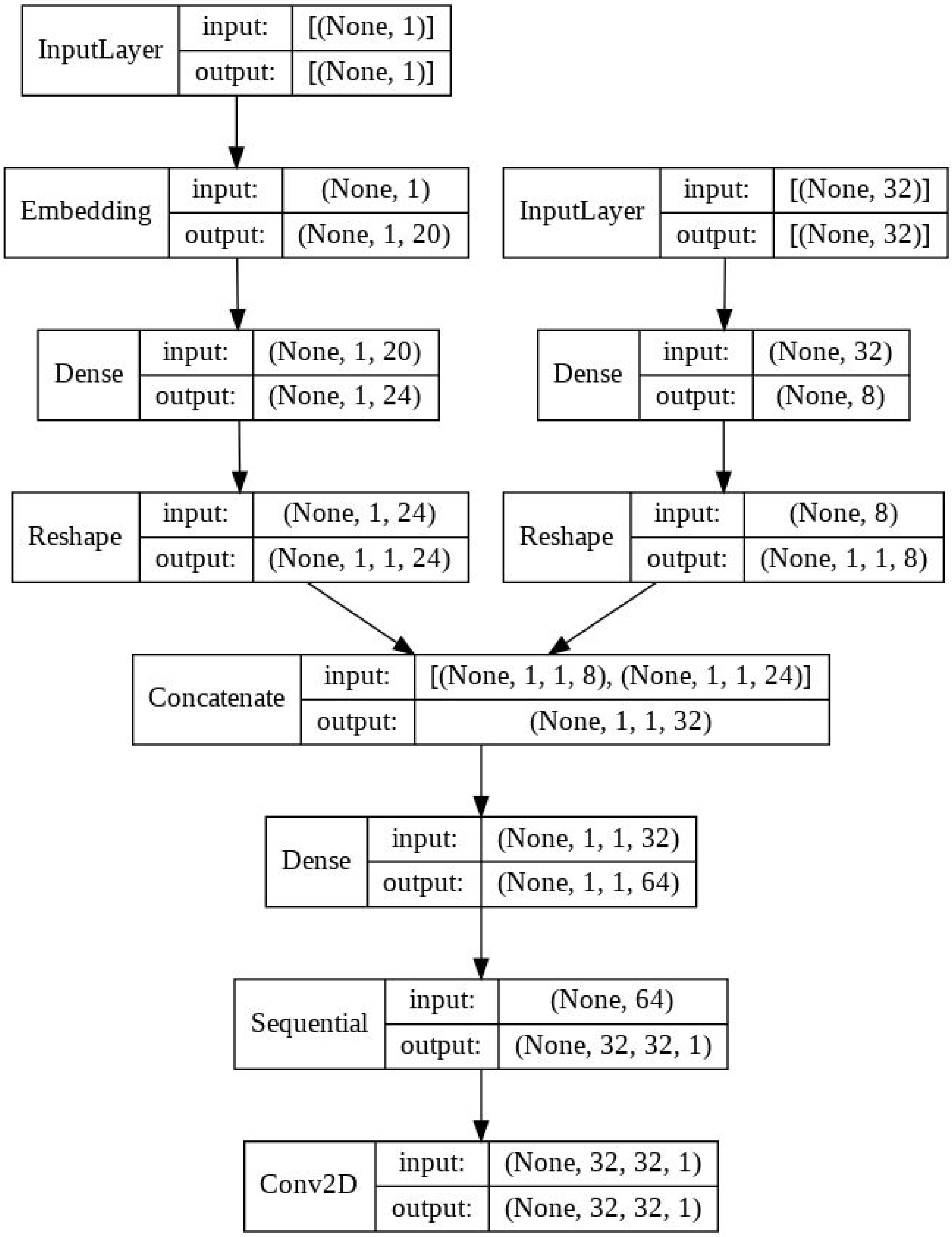
A GAN is an architecture that consists of a generator and a discriminator. The generator is responsible for generating new examples that are indistinguishable from real examples from the dataset. The discriminator is responsible for classifying a given image as either real or fake . A generator should be able to synthesize new images that resemble those of the data distribution by fooling the discriminator.

Our dataset has additional information that is a class label of a digit, and we want to use this information to tell the model which digit to generate. The GAN can be trained in such a way that both the generator and the discriminator models are conditioned on the class label.

We tried 2 different approaches of inserting the labels to the models. First ,we used the categorial labels we got from the first part, and secondly we used an embedding layer of size 20 and 65. The approach with the embedding had lower loss of the generator.Also using embedding size of 20 we trained the model 100 epochs, and when using embedding size 65 we trained the model 50 epochs.

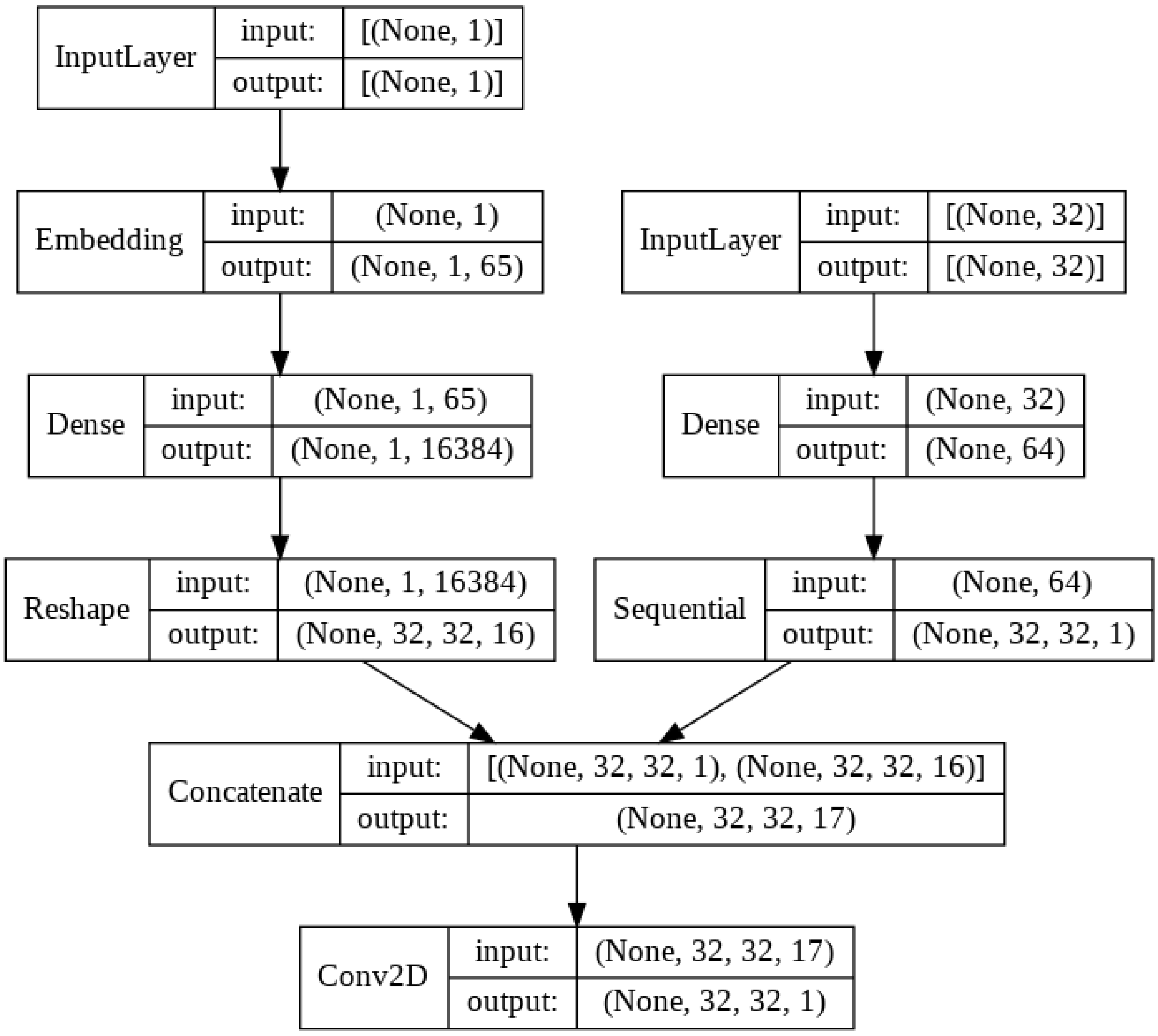
The embedding layer was followed by a fully connected layer that scales them to the size of the image. Later, we concatenate images and their corresponding labels and use this as input to the next layers both in the generator and the discriminator.

We tried 2 different architectures for the generator - one with embedding size 20 , where we concatenated the label with the features vector before inserting it to the decoder:



InputLayer | input: | [(None, 1)]
output: | [(None, 1)]
Embedding | input: | (None, 1)
output: | (None, 1, 20)
InputLayer | input: | [(None, 32)]
output: | [(None, 32)]
Dense | input: | (None, 1, 20)
output: | (None, 1, 24)
Dense | input: | (None, 32)
output: | (None, 8)
Reshape | input: | (None, 1, 24)
output: | (None, 1, 1, 24)
Reshape | input: | (None, 8)
output: | (None, 1, 1, 8)
Concatenate | input: | [(None, 1, 1, 8), (None, 1, 1, 24)]
output: | (None, 1, 1, 32)
Dense | input: | (None, 1, 1, 32)
output: | (None, 1, 1, 64)
Sequential | input: | (None, 64)
output: | (None, 32, 32, 1)
Conv2D | input: | (None, 32, 32, 1)
output: | (None, 32, 32, 1)

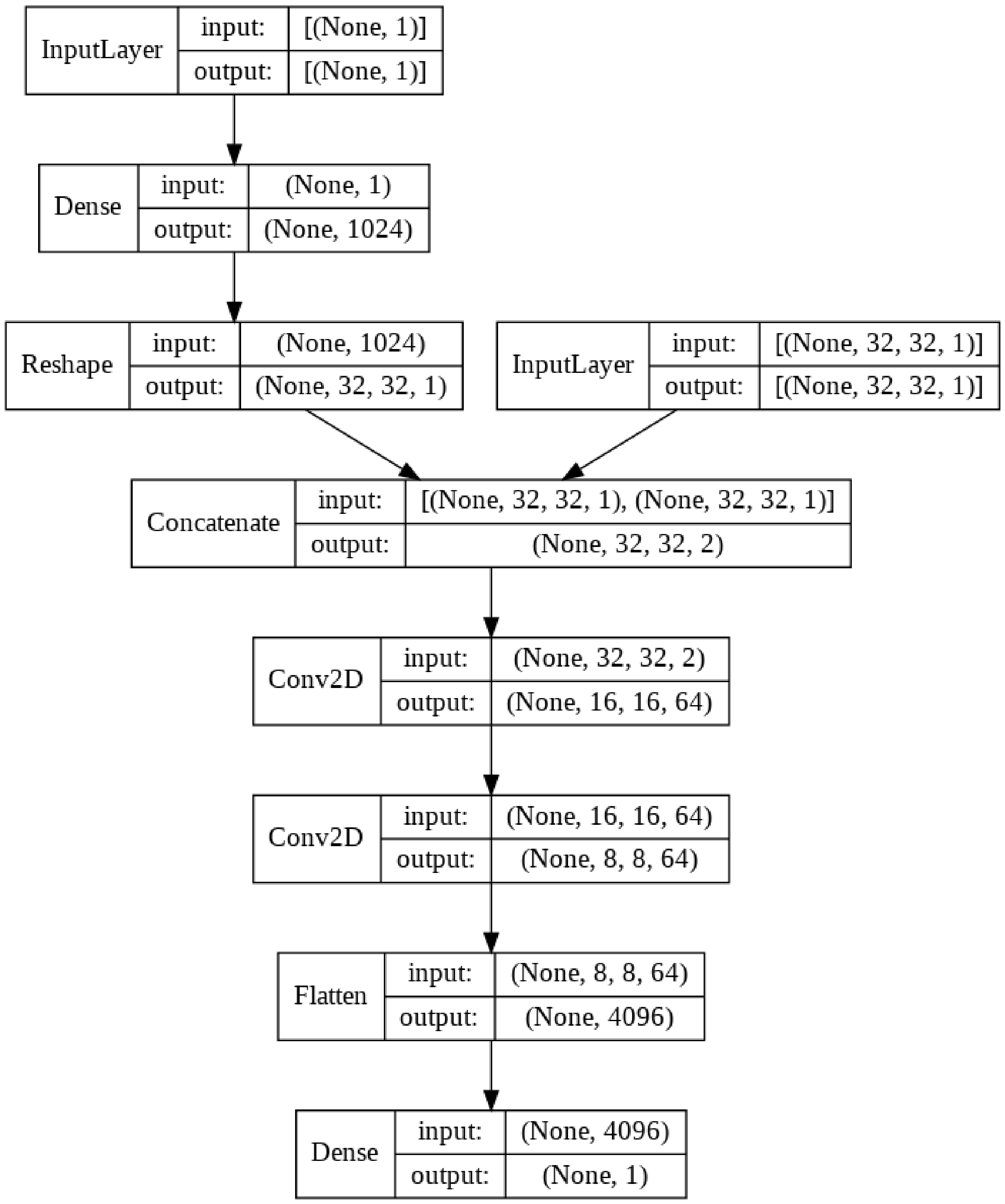
And the other concatenating after insert the features vector to the decoder :



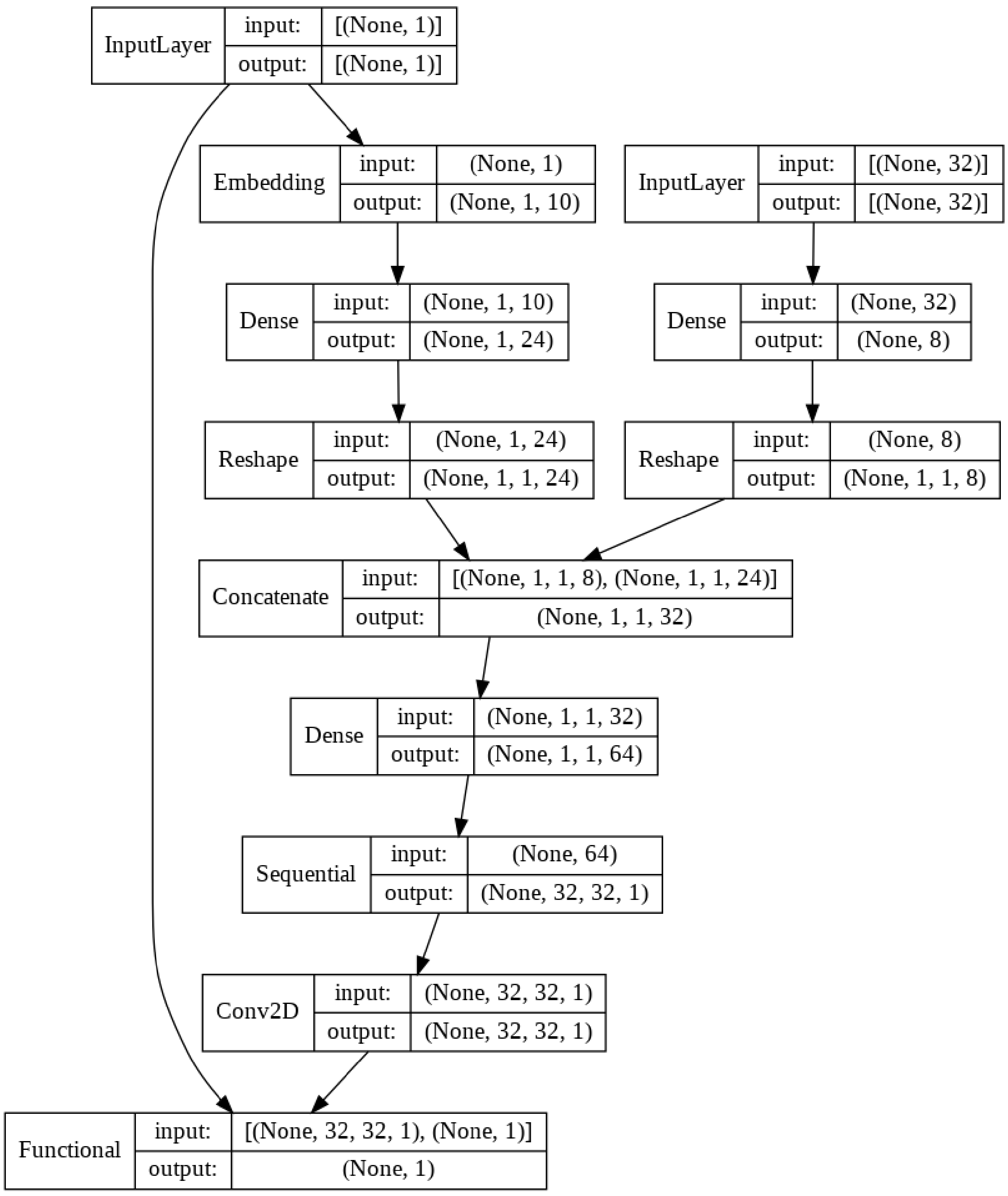
Using the first architecture yields better results.

The generator is updated to take the class label, so the point in the latent space is conditional on the provided class label. In the GAN model the fake images that are generated by the generator and their corresponding class labels are combined as input to the discriminator model. This allows the same class label input to flow down into the generator and down into the discriminator.

Discriminator architecture :



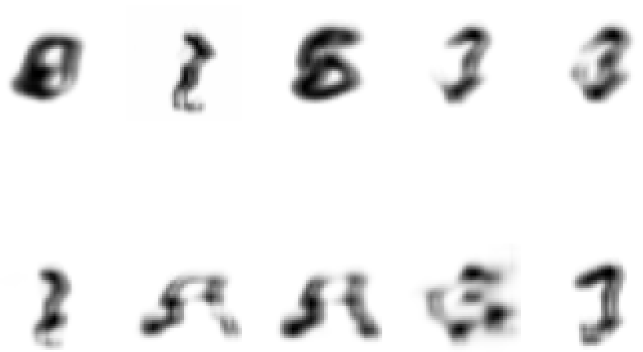
GAN architecture :



We also tried to add a Leaky Relu activation to the discriminator, that made the discriminator much stronger, so it's loss decreased and the generator's loss increased. The images that we got this way were sharper, but with no clear digit expect the digit 3.

The resulted images :

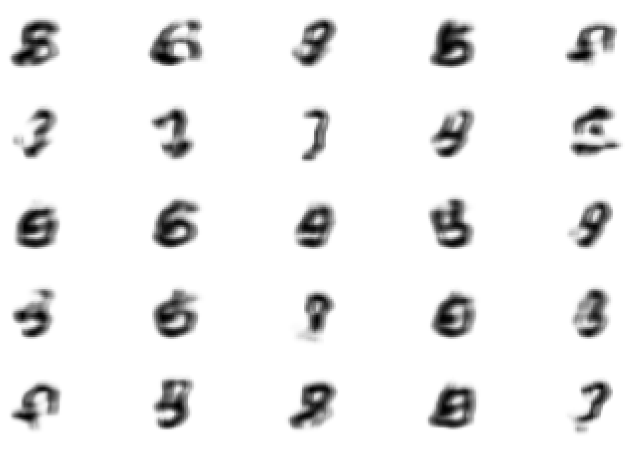
The example below is 10 images generated by the generator according to the digits 0 to 9 :



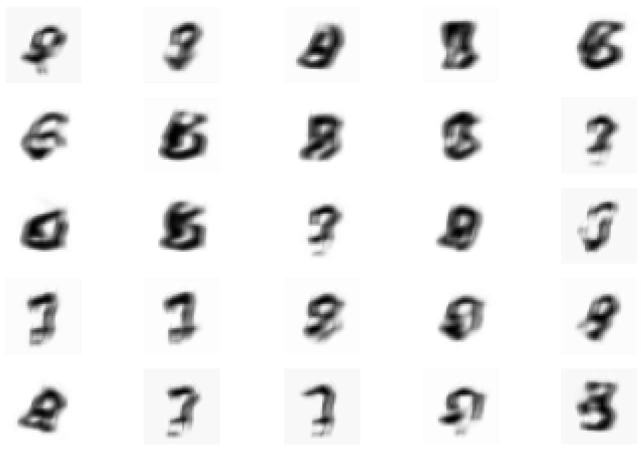
We expected each image to match a specific digit from 0-9 in this order.
As we can see above the generator managed to generate images that match 0,1,3, but wasn't successful when using other digits (it created 6 instead of 2 and 1 instead of 9).

The example belows shows 25 images of the chosen digit generated by the generator when concatenating **before** decoding :

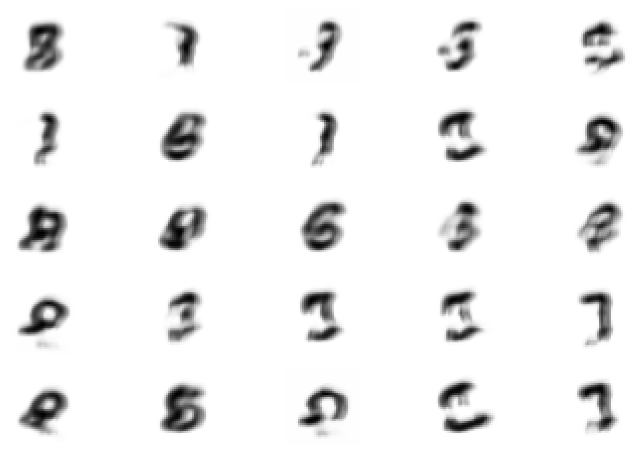
epoch 100 :



epoch 90 :

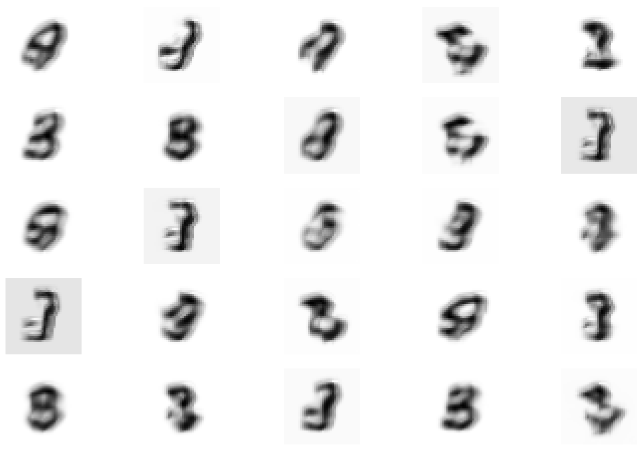


epoch 80：

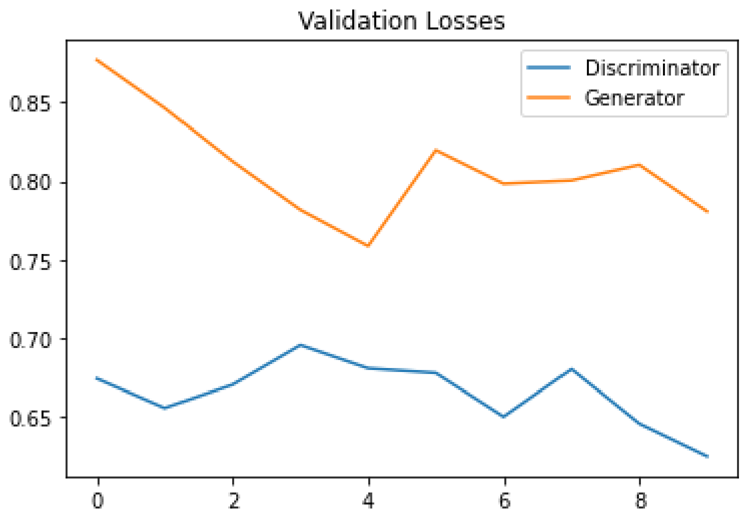


epoch 40：

epoch 20：



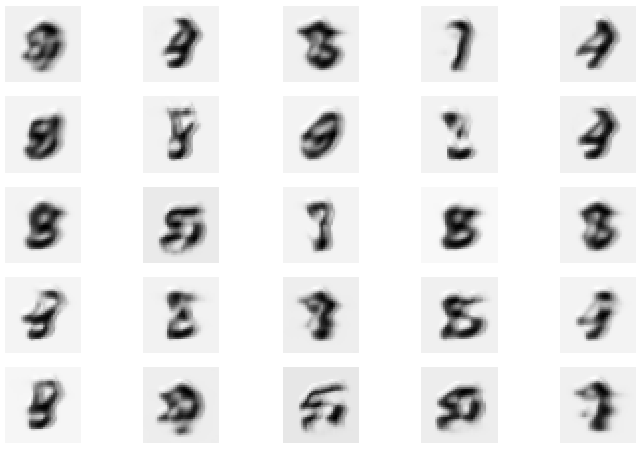
When plotting the loss of the discriminator and generator we got :



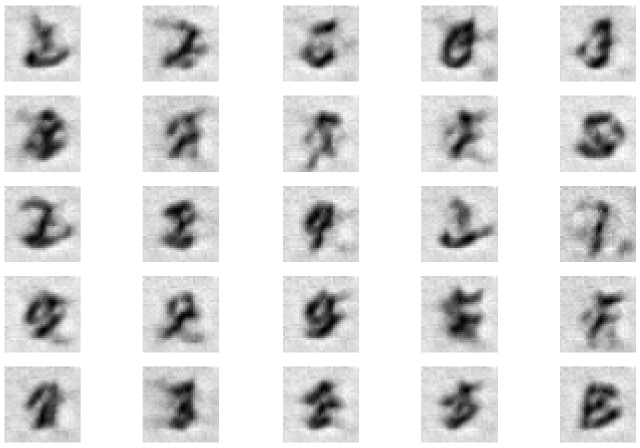
We calculated the loss after every 10 epochs, the discriminator loss is computer as the average loss of the discriminator on real images and fake images. As we expected we can see at the graph above the generator loss decreases after epoch 40 it increases a bit and decreases again. The discriminator loss increases from epoch 10 until epoch 30, but then it goes up and down, when at the end it decreases.

The results from epoch 30 are given below where the generator has the lowest loss and the discriminator has the highest.

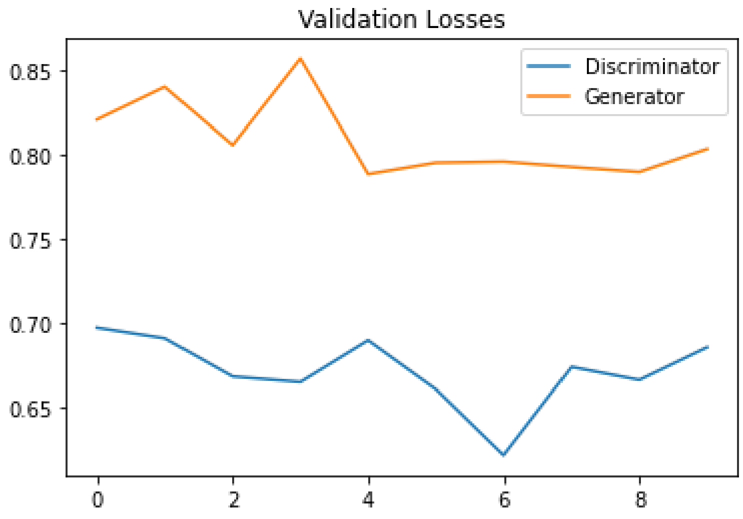
epoch 30 :



The example below is **25** images generated by the generator when concatenating **after** decoding according to given labels.



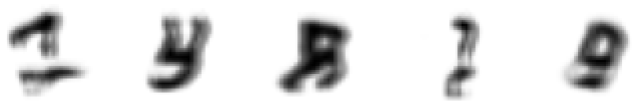
The graph below shows the losses when the generator was built in such a way that we concatenate labels with the image, after decoding:

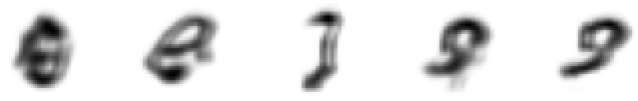


Now we can examine the generated images according to one specific digit, we compared between the generators we build with concatenating before and after decoding:

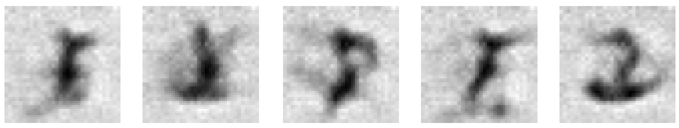
The label input is `0` :

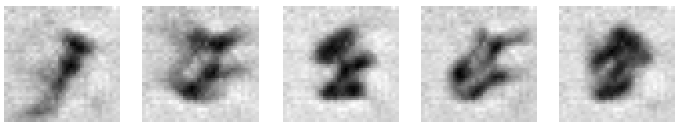
Before Decoding:





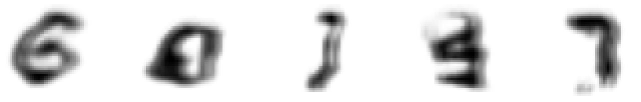
After Decoding:

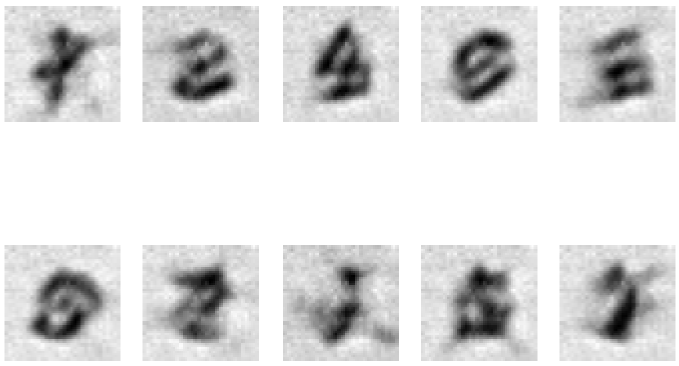




The label input is `1` :

Before Decoding:





Theoretical Questions:

1. Transformers

CNNs relate nearby values (e.g., pixels) when extracting features at each layer. A transformer layer can relate every pair of tokens. Explain how you'd feed an image to a BERT-like arch. such that it will be able to mimic the action of a convolution operator. Explain how this architecture can be used to mimic an FC layer (or an MLP).

We would like to insert the data correctly into the transformer so that it has the capability to mimic the action of a convolution operator,relate nearby values and mimic FC layer.

A transformer is invariant to promotion but the FC layer and the CNN are not. CNN assumes that all the relations in every part of the image are the same, so it analyzes each area in the image in the same way. It relies on the geometric distances, so there is a uniform structure when analyzing the data.

The BERT network consists of stacked transformers and is able to get inputs with variable length, makes all items to interact, and does not consider the order. As each pixel in an image simultaneously flows through the Transformer's encoder/decoder stack, the model doesn't have any sense of order. One possible solution is to add a piece of information to each pixel about its position in the image. Encoding can be done using a k-dimensional vector that contains information about a specific position in an image, where it is not integrated into the model itself. We can feed the network with an input of a positional encoding as given in the lecture :

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

A characteristic of sinusoidal positional encoding is that it allows the model to attend relative positions effortlessly.

To make a transformer have a notion of time/space (as convolution) we will change the way the data is fed to the transformer in such a way that the transformer remains the same.
One approach could be the following - before inserting the image, we will divide it into windows with the size of the desirable convolution kernel. We will create from the image K windows ,so that K is the number of pixels in an image. Each window will have a pixel in such a way that the pixel corresponding to it will be in the center.

A BERT-like arch has for each input an embedding vector, therefore we will encode each window we mentioned above to be an embedding vector and insert it into the network. Inserting the input this way can mimic a convolution operator because the attention layer operates as usual and then the pointwise FC can operate on each window - that is exactly the same action that the convolution kernel would do.There will be exactly K outputs, those outputs make it possible to construct the image from the parts we applied the kernel on.

A transformer is a generalization of MLP since the network allows all the items to interact with each other. The self- attention layer in the transformer allows each item in the data to interact with all the others, and the Feed Forward network operates on each element independently, unlike the self-attention layer.

2. Image Translation

Generative networks can also be trained to operate over images as an input rather than noise vectors. This way the network can translate images from one class of images, denoted by "A", to another class "B". Consider the following three training strategies: Regular GAN-like approach, Cycle GAN, Conditional GAN.
Consider each of these cases and explain whether you expect it to succeed on each of the following datasets.

1. A paired dataset is one in which every image in class A has a counterpart in class B containing the same content but in a different visual style (photographs and hand-drawn images of the same scene). The correspondence could be very accurate (i.e., all the details in one image found in its counterpart).

2. Not so accurate pairing (the drawing could be a loose description of the scene).

3. An unpaired dataset simply contains images from both classes with no correspondence and no guarantee that the same scene will appear in both classes.

Explain whether you expect each strategy to benefit from "upgrading" its dataset (from 3 to 1)

Regular GAN-like

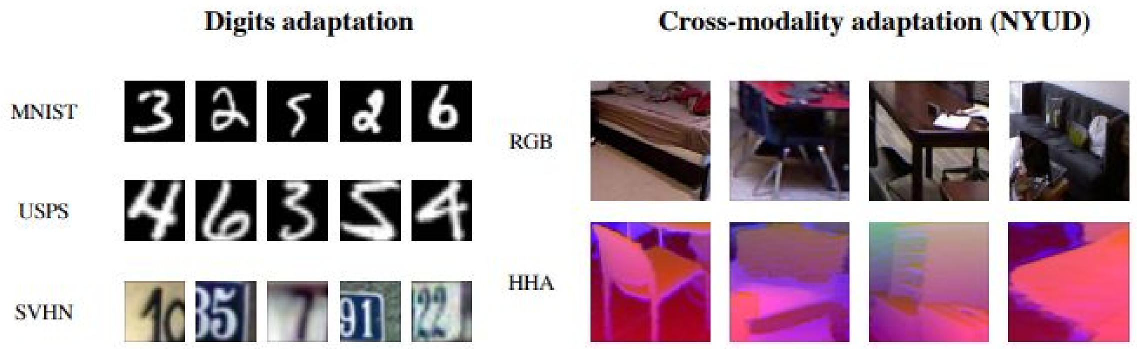
Regular GAN-like approach: feed the generator G with images from class A and while training the discriminator D to separate G's output from images from class B.

We train a generator G:X⟶Y such that the output $G(x) = \hat{y}$, x∈X , is indistinguishable from images y∈Y by an adversary trained to classify $\hat{y}$ apart from y. In theory, this objective can induce an output distribution over $\hat{y}$ that matches the empirical distribution $p_{data}(y)$. The optimal G there translates the domain X to a domain $\hat{y}$ distributed identically to Y.

⇒ this approach will work successfully on all the above listed datasets

However, such a translation does not guarantee an individual input x and output y that are paired up in a meaningful way - there are infinitely many mappings G that will induce the same distribution over .

Domain shift between source and target images could be :



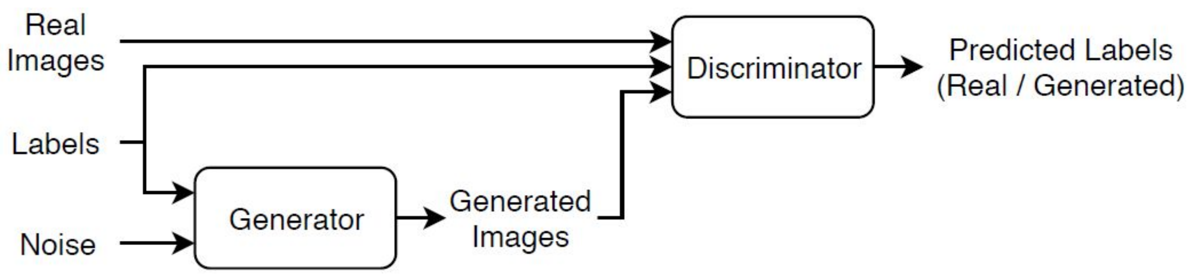
If the domain shift is too large, there is no guarantee that the generated images will be close to the target images.

Conditional GAN

In this strategy the discriminator network needs to distinguish between pairs of images (I,G(I)) where I belong to class A, and pairs of (I,J) where I, again, comes from class A, and J from class B. The generator G is therefore required to generate images that appear as if they came from class B.

This strategy works successfully in image to image translation tasks where sufficient amounts of paired training examples are available.

The generator receives input from group A and a suitable output from group B and produces an image that is supposed to "fool" the discriminator and make him think that the image is actually from group B and not a generated image.



This results in a generator that generates convincingly realistic data that corresponds to the input labels and a discriminator that has learned strong feature representations that are characteristic of the training data for each label.

The system requires a large set of input-output image pairs for training and therefore it will not work well on an unpaired dataset.

Cycle GAN

Cycle GAN will work successfully on unpaired dataset.

Cycle GAN is an image to image translation system that does not require paired examples. Specifically, where any two collections of unrelated images can be used and the general characteristics extracted from each collection and use it in the image translation process.

For example, Cycle GAN is able to take a large collection of photographs and a large collection of hand-drawn images of different scenes and translate specific images from one group to the other. The first GAN G_1 will take a images of photographs, generate hand-drawn images, which is provided as input to the second GAN G_2 , which in turn will generate images of photographs. The cycle consistency loss calculates the difference between the image input to G_1 and the image output by G_2 and the generator models are updated accordingly to reduce the difference in the images.

One major limitation of CycleGAN is that It works well on tasks that involve color or texture changes (day-to-night photo translations, photo-to-painting tasks like collection style transfer) however, tasks that require substantial geometric changes to the image, such as cat-to-dog translations, usually fail.

Another limitation of CycleGAN is that it only learns one to one mappings, i.e. the model associates each input image with a single output image. Therefore an accurate paired dataset and not so accurate paired dataset CycleGAN will not work so successfully because the output of the model will not necessarily be the same or similar to the corresponding output provided in the dataset for this input.