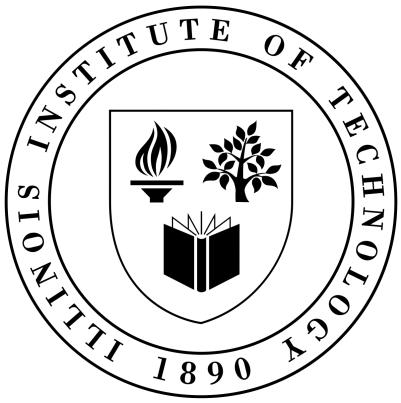

CS 512

Project Proposal

Authors:
Jean Haberer – A20496422
Kemen Goicoechea - TODO

Supervisor:
Agam Gady

Spring 2022

1 Description of the problem

Through this project, we will create a good representation of a number's image in a low dimensional space. We wanted to analyze the way our model perceives the numbers as an image. We are interested to see what differences our model would see between two numbers and how it comes from one number to another. Our goal is to display transition animations between numbers and and variations between different number's writing.

To achieve this goal, we wanted to use auto-encoders and convolution neural network such as our support papers suggested. According to our main research paper, auto-encoders using convolution layers have the ability to reduce drastically the dimension of an input without loosing too much information. Such auto-encoders have a lot of different usages. As the paper [1] and [2] shows, we can use them to classify unlabelled data (using VAE for example), generate images (using the decoder) and reduce the size of an image (using the encoder) for instance.

In this project, we will have a new approach based on those papers. We will try to use one kind of auto-encoder to reduce the dimension of an MNIST image to a meaningful low dimension equal to 11. 10 elements determining the number displayed and 1 representing a variation parameter. This variation parameter can be considered as the 'font' in which the number is written.

2 Methods

2.1 Idea behind the method

Because we want to have a meaningful latent space, we thought of training our auto-encoder in a certain way. We would first train the encoder to classify correctly the numbers, then fix the encoder weights responsible for the classification. This way we keep our meaning behind the output of the encoder. We then train the entire auto-encoder allowing it to figure out the best way to take into account this 'variation' or 'font' parameter on its own, this will also train the decoder. We will finally be able to generate a number image from its value and a 'font' value.

2.2 The auto-encoder model

2.2.1 Encoder

Following a simpler version of the best architecture for MNIST classification, the encoder is composed of 3 layers of 2D Convolutions with 3 Batch Normalization layers, 2 Max Pooling layers and finally the first output of size 10. All of the weights from these layers will be fixed during the training of the whole auto-encoder. We then have a Normalization layer followed by a dense layer to finally have our second output, the 'variation' parameter. This parameter will be trained by the auto-encoder.

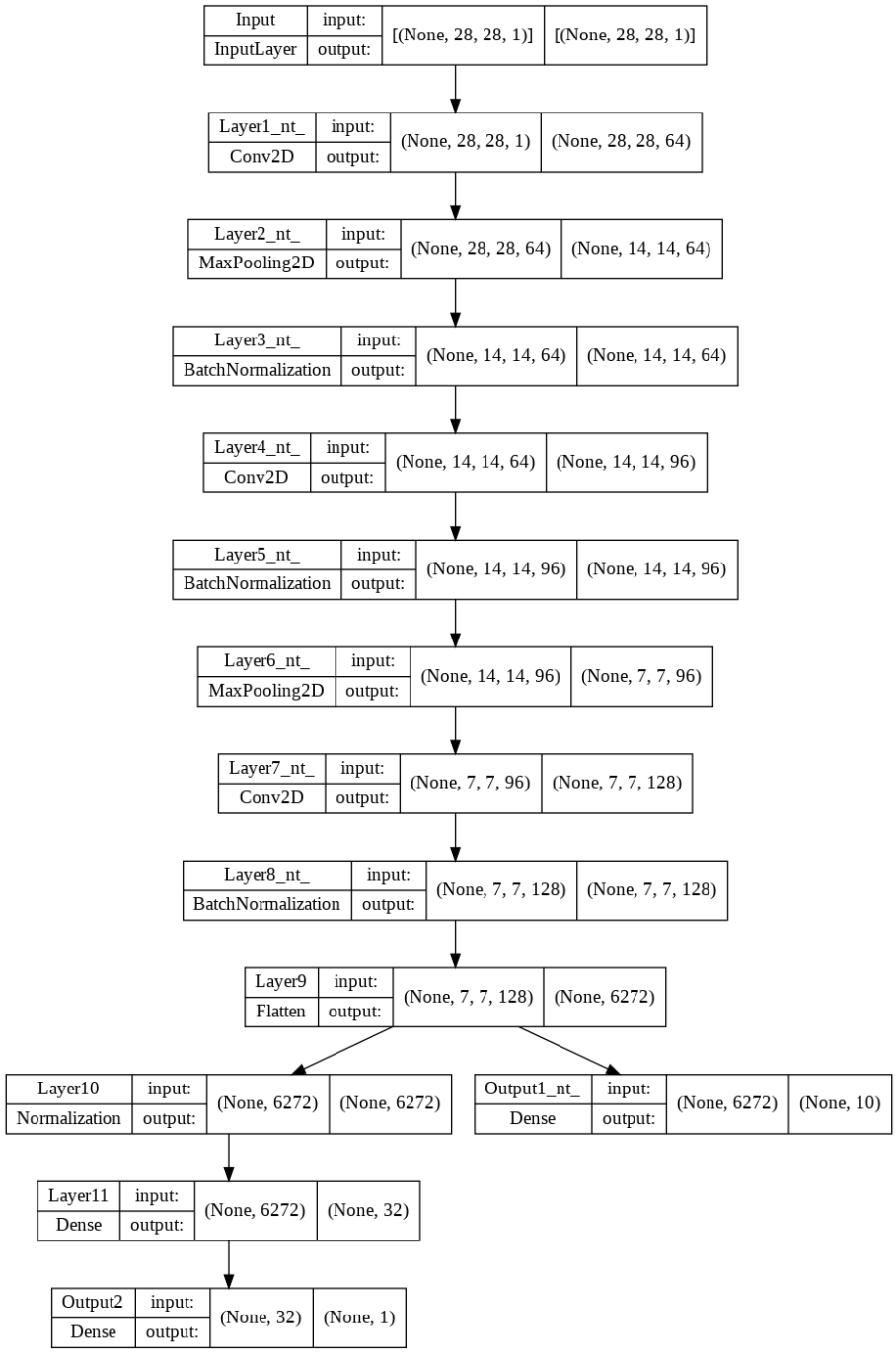

Figure 1: Encoder architecture

2.2.2 Decoder

The decoder follows a very similar architecture to the encoder, but in the opposite direction.

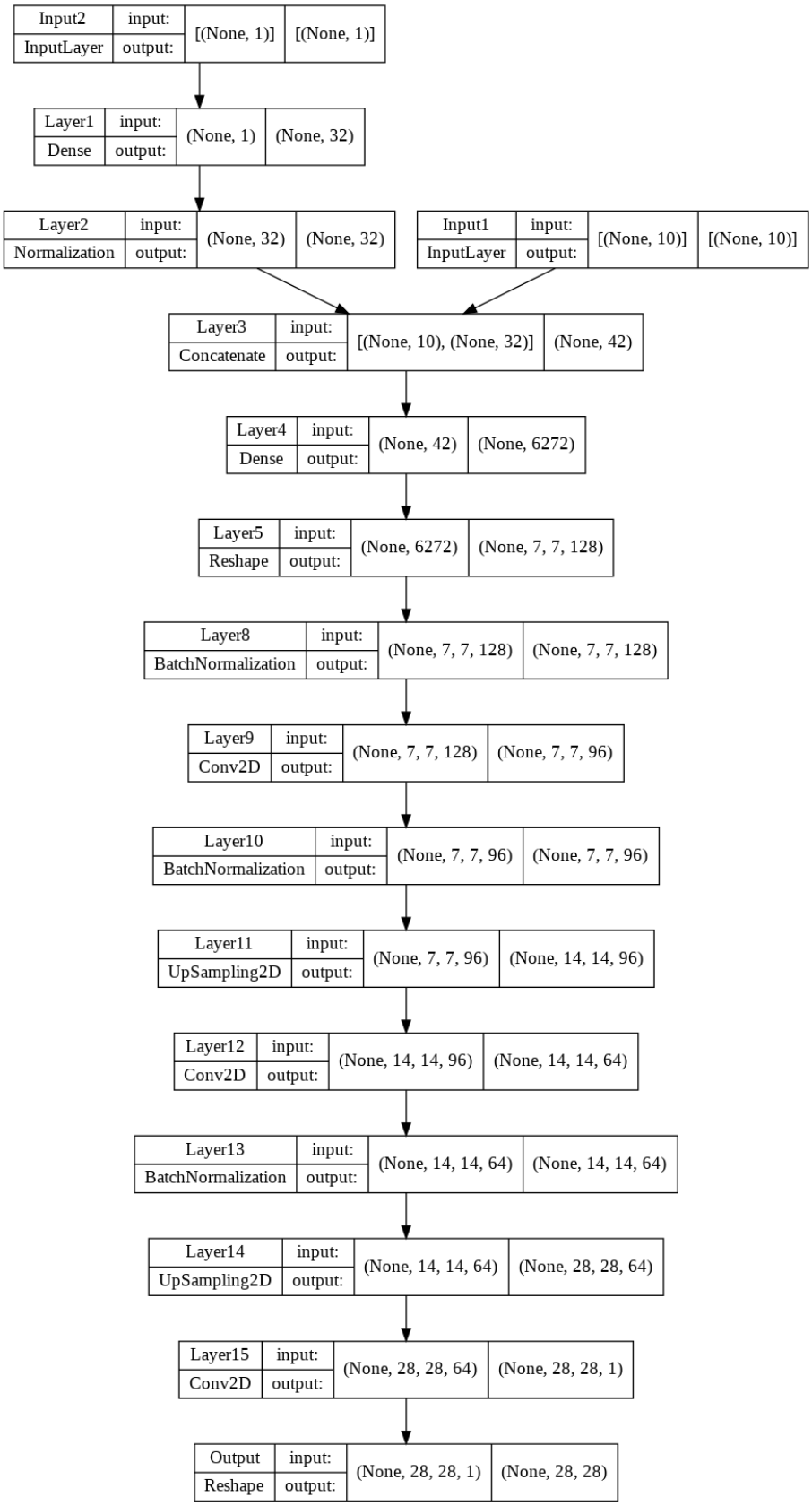

Figure 2: Decoder architecture

2.2.3 Whole Auto-Encoder

The auto-encoder is simply composed of the encoder, followed by the decoder. As you can see there is our latent space in between composed of a first vector of size 10 and a unique number as second output from the encoder.

The encoder is composed of 430,827 trainable parameters, the decoder is composed of 436,961 trainable parameters and finally the whole auto-encoder is composed of 637,730 trainable parameters. There are 243,820 non trainable parameters that actually represent the pre-trained parameters of the encoder.

The majority of the parameters in the auto-encoder (more than 50%) are directly related to the 'variation'/'font' parameter.

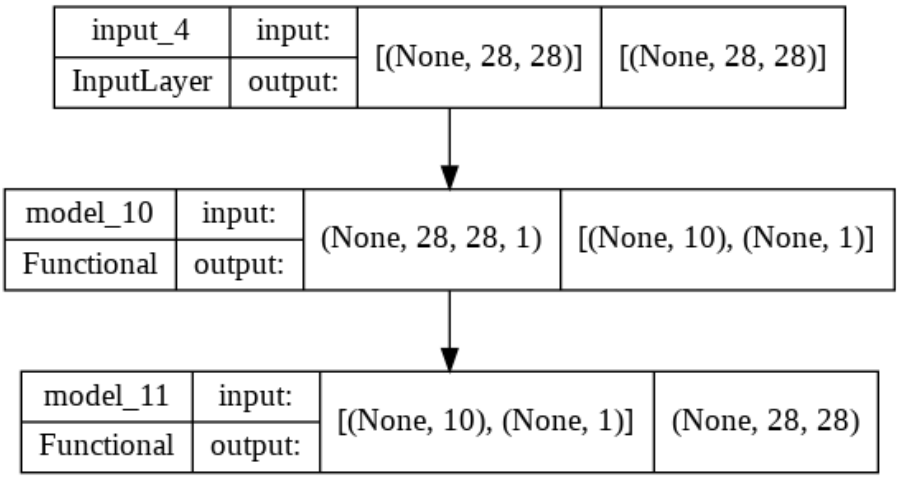

Figure 3: Auto-encoder architecture

2.3 Training the model

2.3.1 Dataset

We used the MNIST dataset to train our auto-encoder. Its strength is its large amount of pre-processed labeled examples.

Set	Size	Label
Train	55,000	{1,...,9}
Validation	10,000	{1,...,9}
Test	5,000	{1,...,9}

Table 1: MNIST Dataset used

2.3.2 Training the encoder

As you can see in our notebook (please read instruction at the end of this report), 3 epochs are more than enough to get a high accuracy for the classification task -

98.49% on the validation dataset. During this training, we don't train the second part of the encoder responsible for the 'variation' parameter. After the training, we fix all the weights responsible for the classification task.

2.3.3 Training the auto-encoder

We can now use the Binary Crossentropy to train the whole auto-encoder by giving to it the images from the training set as input and output. This way, the model will try to minimize the difference between the input image and the output image. With only 25 epochs, we already get a satisfying result with a loss value of 0.1832.

We can see in the notebook (and on the next figures) the evolution of the training. 2 inputs are given to the encoder to generate an image of a 1 at each epoch. The first being a one hot encoded vector [1,0,0,0,0,0,0,0,0,0] and the second the variation parameter with a value of 0.5.

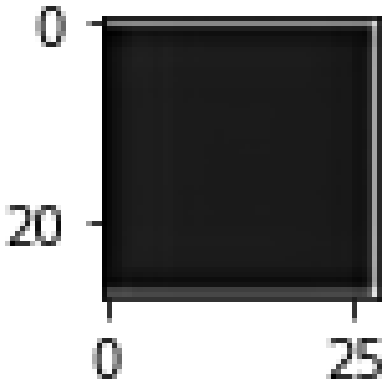

Figure 4: Epoch 1

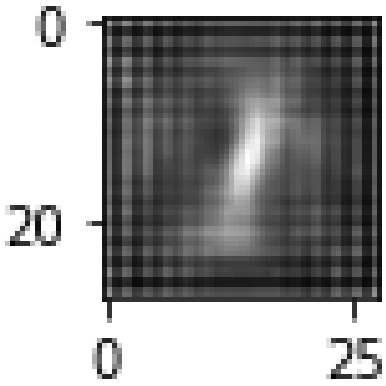

Figure 5: Epoch 2

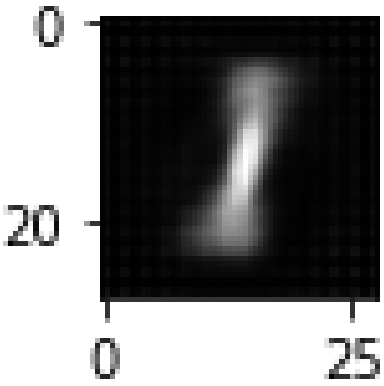

Figure 6: Epoch 3

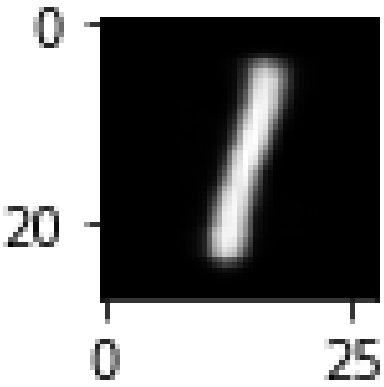

Figure 7: Epoch 8

3 Results

3.1 Generating number animations

Now that the entire auto-encoder is trained, we can use it to explore the numbers. There are a lot of ways to explore the results, we wanted to produce graphical visualization of what the auto-encoder learnt through the training.

The first thing we can construct is an image showing the smooth transformations in between the numbers. To do that, we are generating images with the decoder part of the AE.
More precisely, we are building vectors that correspond to transitions between numbers and feed them to the decoder. For example, in our model, if we consider x as the progress of a transition from 1 to 2. The image would be encoded as follow:

$$[0, x, 1-x, 0, 0, 0, 0, 0, 0, 0] \tag{1}$$

We can then use the 'variation' or 'font' parameter to encode a type of writing, from 0 to 1.
This is what we have done to build the following figure. You can see horizontally the transformation between each numbers from 0 to 9, and vertically the transformation between variations of writing.

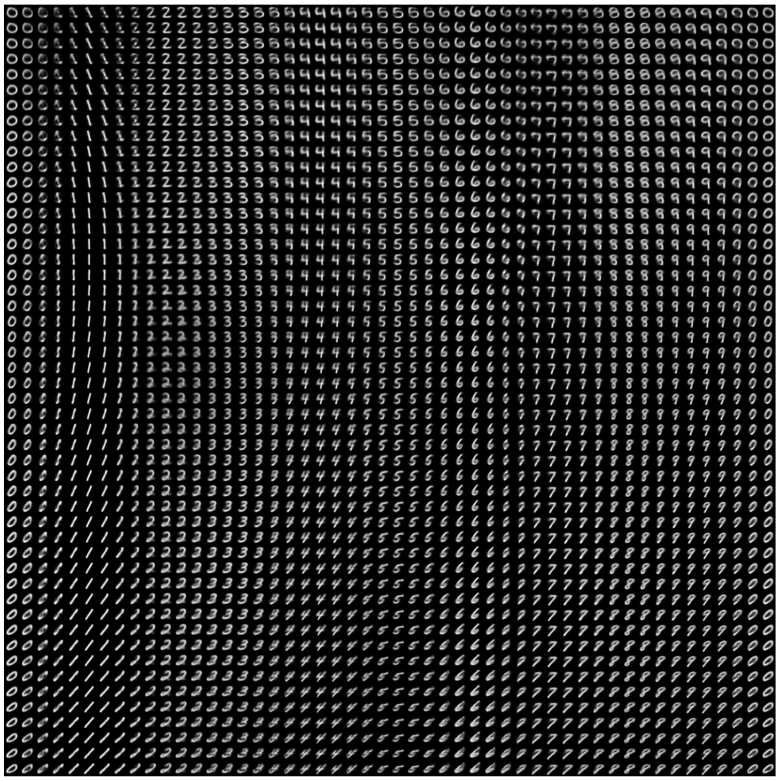

Figure 8: Number (horizontally) vs Variations (vertically from 0 to 1)

A second way of observing transition is by generating a video. With similar methods, we generated 2 particularly expressive videos showing both transformation from variation to variation and from number to number. You can download all the results here.

Figure 9: Screenshot of one of the video generated

You will find in one of the video a header composed of two sliders. The blue one corresponds to the current writing variation index (from 0 to 1) and the red one corresponds to the current number being displayed (from 0 to 9).

TODO: more detailed results

TODO: update result images (not up to date)

3.2 Interpretation

With our result, we can conduct some interpretations. The way the model understand the 'variation'/'font' parameter is particularly interesting. It seems that it is responsible for scaling, rotating, and positioning the number in the image, as well as adding features to the number's writing. For example, we can see in the Figure 8 that the models can see the number 2 as two really different writing:

TODO: Montrer des exemples de difference entre 2 ecriture de plusieurs nombres, un tableau?

TODO: Autre interpretations?

TODO: Afficher mieux les resultats?

3.3 Improvement and future work

This fascinating project shows only the tip of an iceberg. There is a lot more that we can do with similar auto-encoder architectures.
First to improve our model, we could actually use 2 dimensions for the 'variation' parameter to get more accurate results.

Future work could consist in exploring other dataset, more complex, with a similar model.

We could find application for this model in various fields like lossy compression. Indeed, if we could train a similar network on a big dataset composed of object, we could generate an image composed of different objects by encoding the objects present in the image along with some variations like position, rotation, color, etc. At the end we would be able to compress simple images with only a few bytes and then regenerate them back with the help of decoder.
TODO

4 Program Instructions

All the code we wrote is included in a python notebook in our personal CS512 repository. You should be able to run the entire notebook easily, cell by cell. You do not need to train the model on your computer, you can import the models' weights by running the corresponding cell in the notebook.

Make sure to download the models through this link. Unzip the file, and place it into the project folder prior to run the notebook. You can always edit the paths in the first cell of the notebook if needed.

You can find all the results at this link.
TODO: See dans le project presentation PDF if on a oublié qqchose, peut etre rajouter plus des references aux research papers

5 References

[1] Gabriel B. Cavallari, Leonardo Sampaio Ferraz Ribeiro and Moacir Antonelli Ponti.
Unsupervised representation learning using convolutional and stacked auto-encoders: a domain and cross-domain feature space analysis, 2018;
arXiv:1811.00473.

[2] Dor Bank, Noam Koenigstein and Raja Giryes.
Autoencoders, 2020;
arXiv:2003.05991.