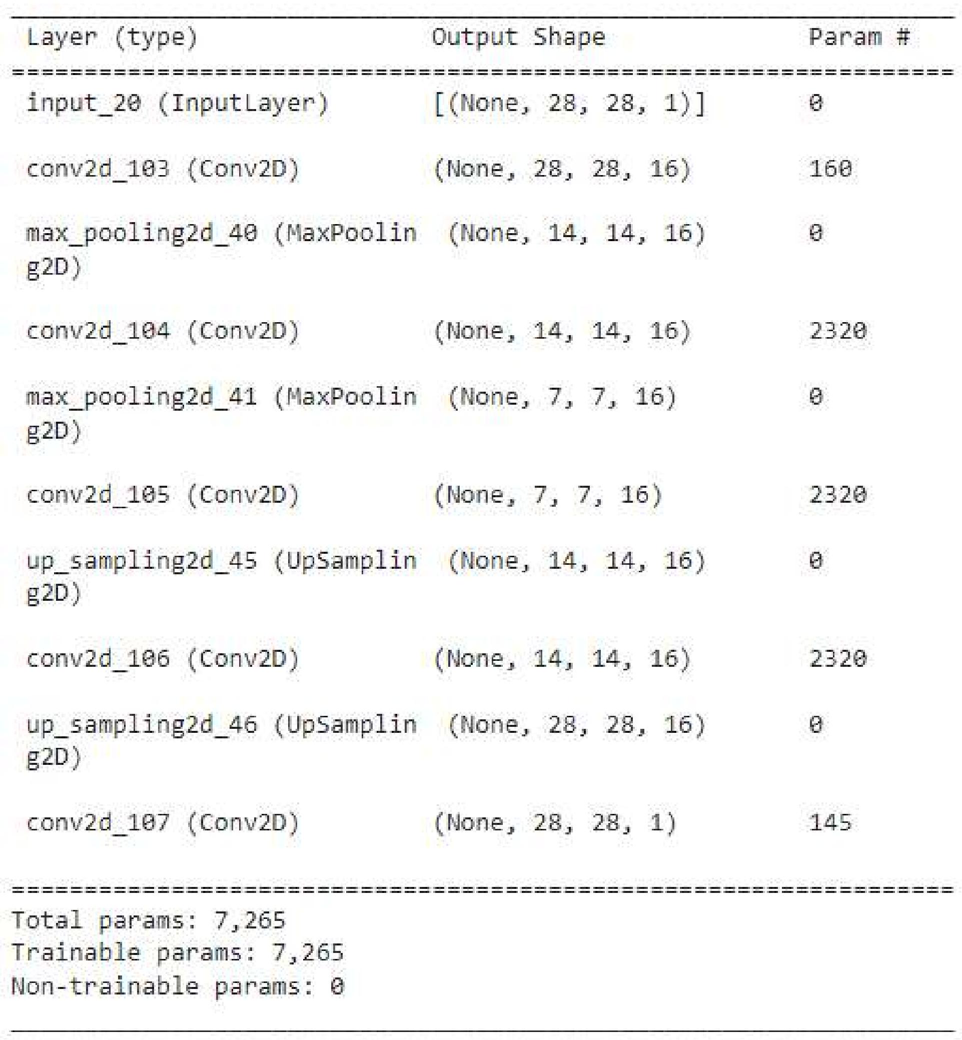
# ML-PW-12

**by Aurélien Héritier and Jean Nanchen**

## Task 1

*Provide the notebook your setup for training a convolutional autoencoder and for denoising the digits of the MNIST database.*

For the autoencoder, we used 2 convolutional layers to encode the input data (28x28 pixels image) to a 7x7 data (divised by 16). And we rebuilt the image from that. We modify the setup found in the blog explication (https://blog.keras.io/building-autoencoders-in-keras.html) to match the professor autoencoder structure needed.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_20 (InputLayer)       [(None, 28, 28, 1)]       0

 conv2d_103 (Conv2D)         (None, 28, 28, 16)        160

 max_pooling2d_40 (MaxPoolin  (None, 14, 14, 16)       0
 g2D)

 conv2d_104 (Conv2D)         (None, 14, 14, 16)        2320

 max_pooling2d_41 (MaxPoolin  (None, 7, 7, 16)         0
 g2D)

 conv2d_105 (Conv2D)         (None, 7, 7, 16)          2320

 up_sampling2d_45 (UpSamplin  (None, 14, 14, 16)       0
 g2D)

 conv2d_106 (Conv2D)         (None, 14, 14, 16)        2320

 up_sampling2d_46 (UpSamplin  (None, 28, 28, 16)       0
 g2D)

 conv2d_107 (Conv2D)         (None, 28, 28, 1)         145

=================================================================
Total params: 7,265
Trainable params: 7,265
Non-trainable params: 0
_________________________________________________________________
```

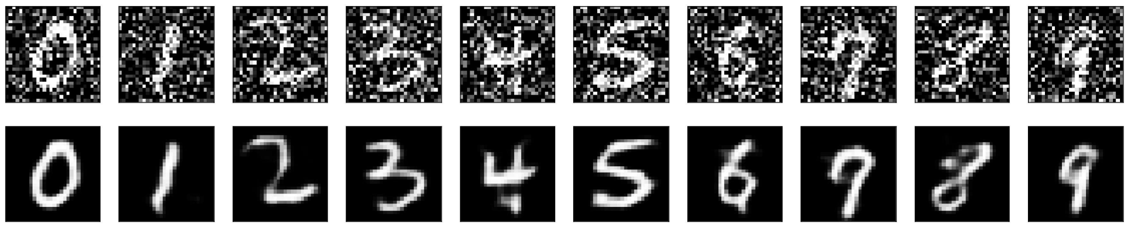
```
input_img = Input(shape=(28, 28, 1))  # adapt this if using `channels_first` image data format

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)

encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()
```

We configured our model to use a per-pixel binary crossentropy loss, and the Adam optimizer. Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models.

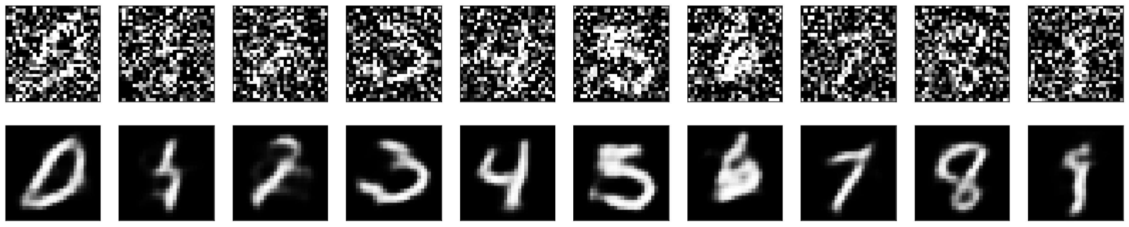
## Task 2

*Provide results showing some examples of the ten digits of the MNIST database before and after denoising for the three different levels of noise: noise factor = 0.5, 0.7, and 0.9*

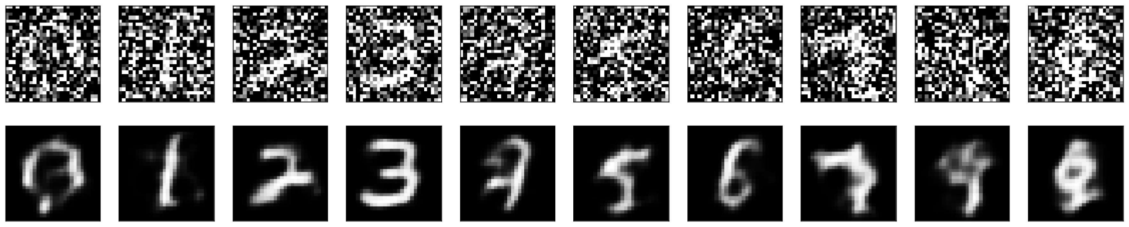
screen à 0.5, 0.7, 0.9



Noise = 0.5
All the digits have been decoded and are clearly visible after denoising.



Noise = 0.7
Digits are globally well decoded. In this case, the 2 and the 7 look the same.

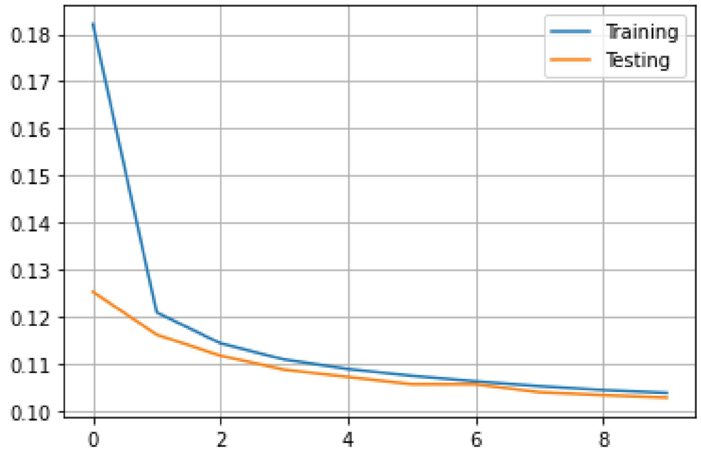


Noise = 0.9
The figures are moderately well decoded. The digit 4 looks like digit 7, digit 8. Digit 9 is not recognizable.

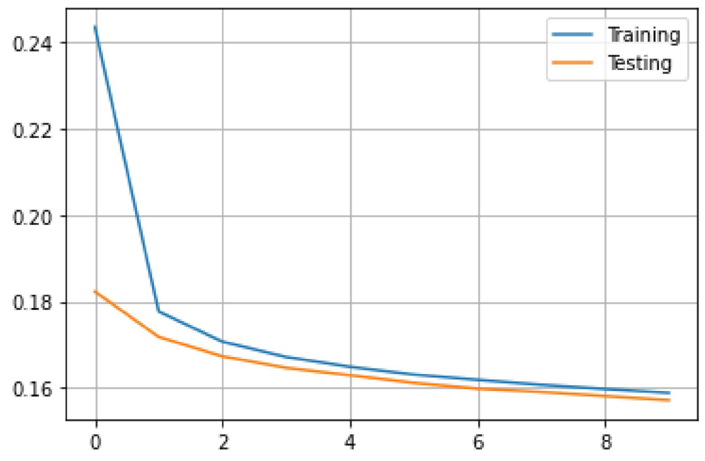
## Task 3

*Conclude based on the obtained results*

The more noise, the higher will be the loss rate. With a noise factor of 0.9 the loss rate is equal to 0.15. With a noise factor of 0.5 the loss rate is equal to 0.10
After a certain noise factor, it becomes hard to decode the digits correctly.



Noise factor = 0.5

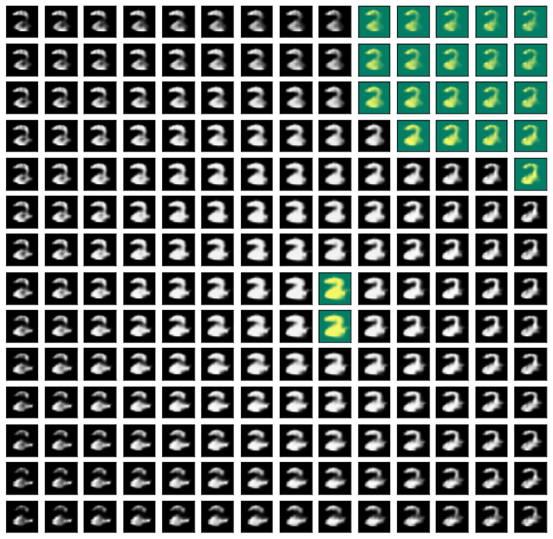
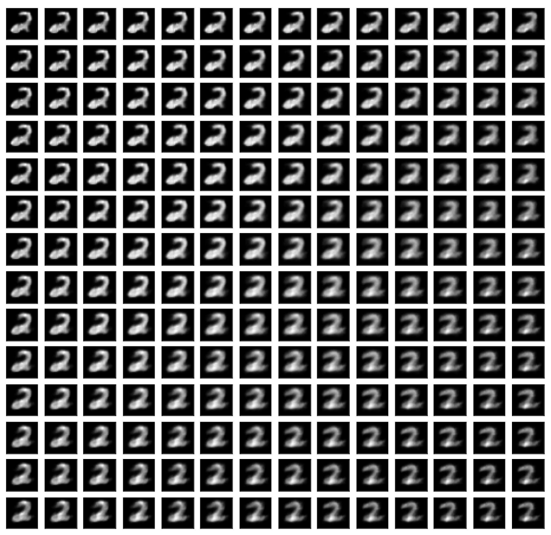


Noise factor = 0.9

## Task 4

*Regarding the data augmentation exercise, generate different synthetic digits (e.g., twos, threes and eights).*

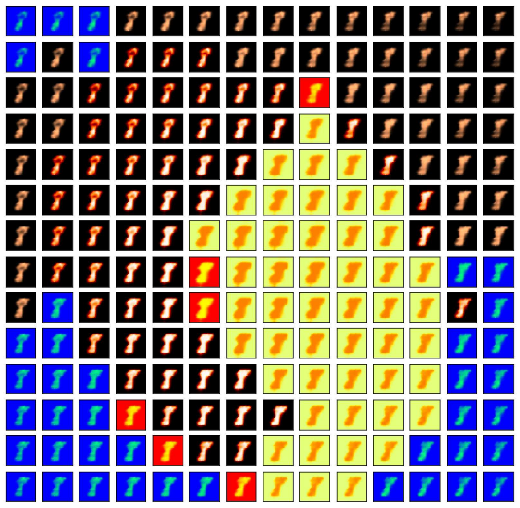
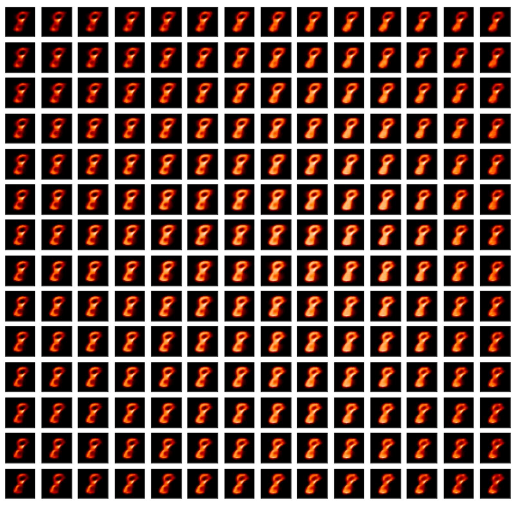
Test with 2 :

10 samples 100 samples

Some generated samples are not well classified. But not so much, only when the layent variables are extreme. And only when the input samples are not enough (10 samples).

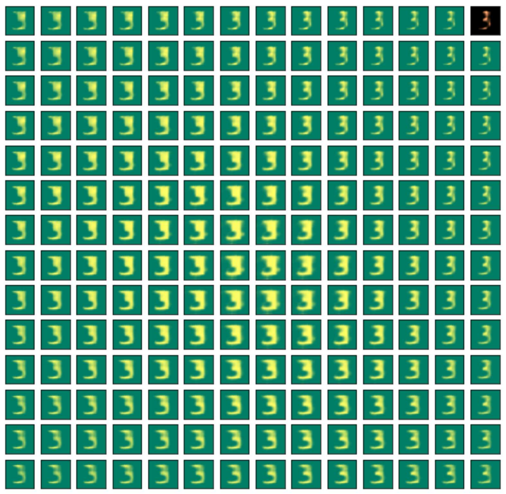
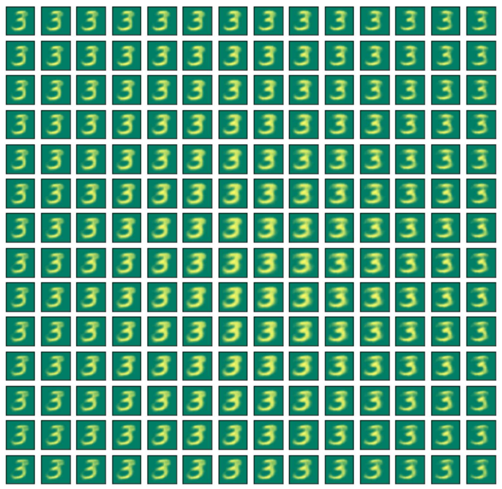
Test with 8 :

10 samples 100 samples

We see that the values that are not well classified in the 8 class is higher than with the 2 class. Only when the input samples are not enough (10 samples).

Test with 3 :

10 samples 100 samples

With 3, the generated values are well recognized. We think this is because the number 3 have some particular features that only him have.

*Comment about the diversity of the synthetic objects the autoencoder is generating. Is it possible to measure that diversity?*

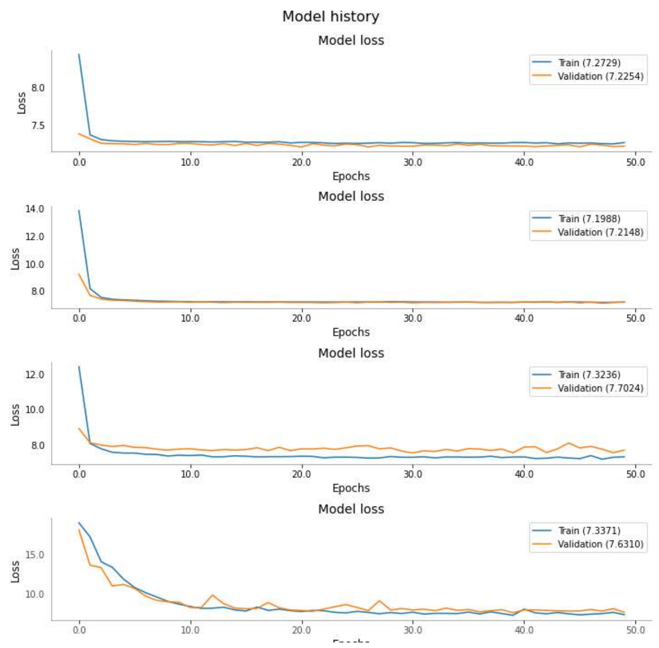
The numbers of patterns are not big in appearance. They are circa two main declinations for the number 2, with or without a loop on the bottom. But they are all different. When the amount of data used to train the autoencoder is higher, the generated outputs are more diverse in appearance.

It’s possible to measure the diversity by comparing the different image pixels by pixels. This algorithm can find the number of independent images. But in theory, the diversity (number of different images) are the same between all the amount of data used to train the autoencoder. The amount of data used will just amplify the difference between the features that are different between the pictures. But the amount of the features are theoretically the same.

## Task 5

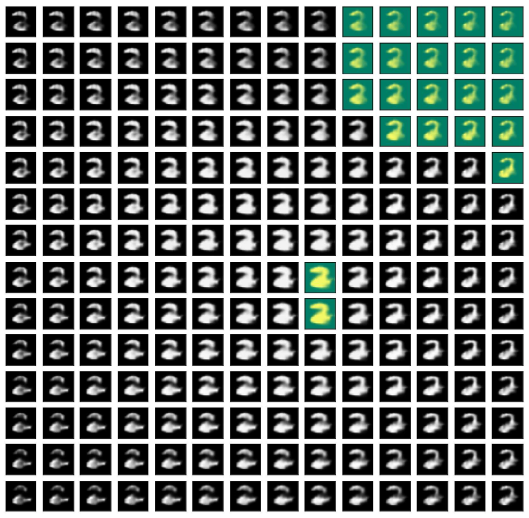
*Provide a summary of your experiments using the autoencoder for data augmentation and comment your results (e.g., modify the latent variables of an autoencoder that was trained with only 10 examples such that it generates digits that are no longer recognized by the pre-trained classifier).*

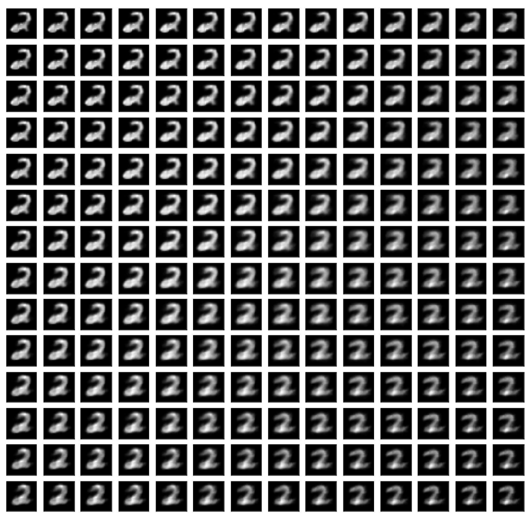
When changing the input training dataset for the autoencoder. (From the whole dataset, 1000 samples, 100 samples and 10 samples). We can see that when the dataset is bigger, the loss is lower. Which is important because it’s a key aspect of the variable autoencoder. It measures how different the reconstructed data are from the original data.



Model loss in function of epochs (from top to bottom) for the whole dataset, 1000 samples, 100 samples and 10 samples

When the latent variables are too extreme, and the training data are too small. The generated picture can be recognized as another number. It the image below, we see that the green numbers are recognized as a 3. Which is a fault. But this case appeared only with the trained autoencoder with only 10 samples.


10 samples


100 samples

In the other case with more samples to train the autoencoder, there are no generated mistakes.