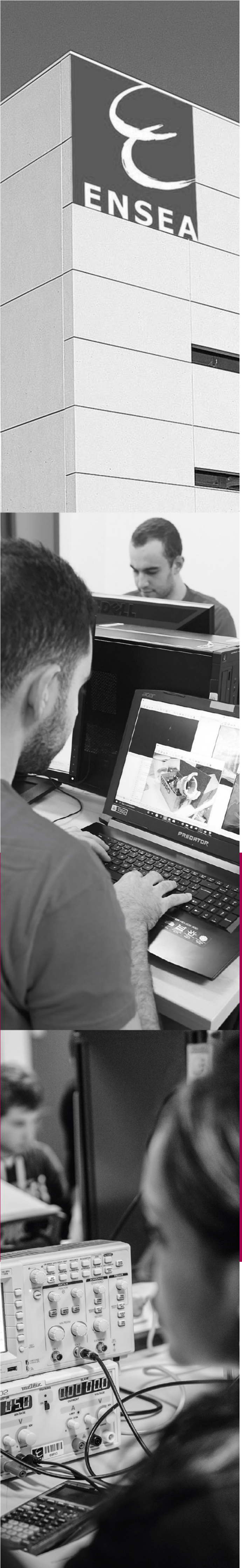


ENSEA

Beyond Engineering

IMAGE COMPRESION - AUTOENCODER

Lab report

2nd year student

Erwan BROUDIN

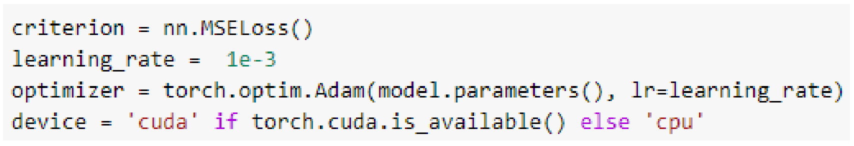
Friday, 14 March 2023

Contents

1 Introduction

1.1 The different values when we train an AI model

We do not modify the following values during this lab. These different values can be modified in some cases, and we could have gone further and done so to observe the differences this would have caused.



```
criterion = nn.MSELoss()
learning_rate =  1e-3
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

Our constants

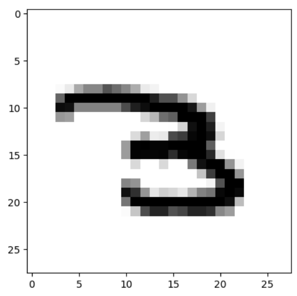
In addition, we have created three functions that allow us to avoid having to recopy the same large block of code (DRY method!):

1. **trainingAndValidationLoop** : The training/validation loop
2. **plotTrainingAndValidationLoss** : The drawing of loss curves
3. **testingLoop** : A test call

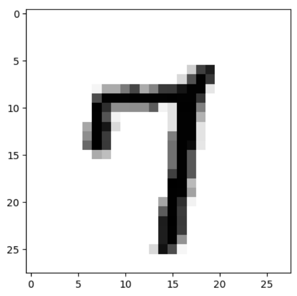
Each of these functions has many optional arguments that we could change if necessary.

1.2 Presentation of MNIST

MNIST is a dataset containing 28*28 images, which are handwritten numbers in black on a white background. The target values are the values of the written numbers. Here are two examples of data from this dataset:



An image from the MNIST dataset

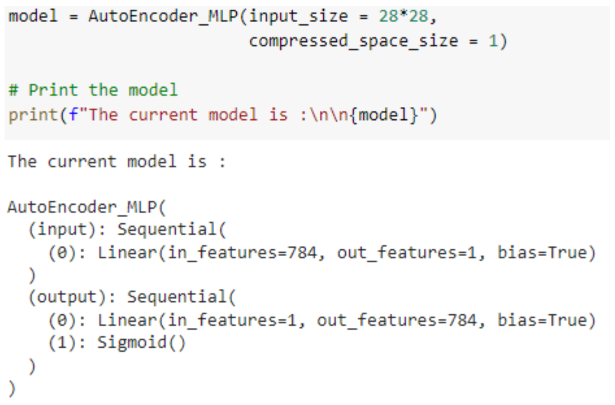


Another image from the MNIST dataset

2 AutoEncoder - Multi Layer Perceptron

2.1 Instantiation

We have coded the AutoEncoder_MLP class which allows us to compress images. This autoencoder consists of a single encoding layer (linear), a decoding layer (linear) and an activation layer (a sigmoid). After encoding the class, we can display the model and we have :



```
model = AutoEncoder_MLP(input_size = 28*28,
                        compressed_space_size = 1)

# Print the model
print(f"The current model is :\n\n{model}")
```

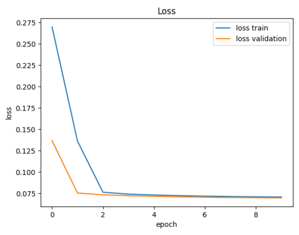
```
The current model is :

AutoEncoder_MLP(
  (input): Sequential(
    (0): Linear(in_features=784, out_features=1, bias=True)
  )
  (output): Sequential(
    (0): Linear(in_features=1, out_features=784, bias=True)
    (1): Sigmoid()
  )
)
```

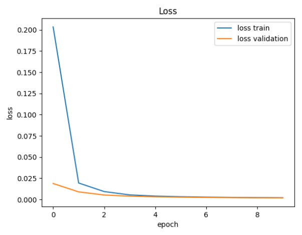
An example of AutoEncoder_MLP, for a compressed_space_size of 1

2.2 Loss results

We have created several models of this autoencoder and we get results that are similar (although they have differences!) In all cases, we notice that the loss decreases well. The larger the compressed_space_size, the faster our model "learns" and the loss is much lower. Here are two loss curves:



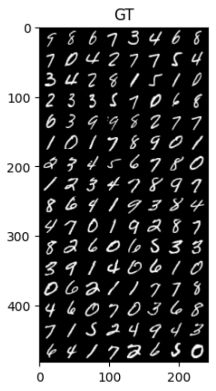
loss curves for compressed_space_size = 1



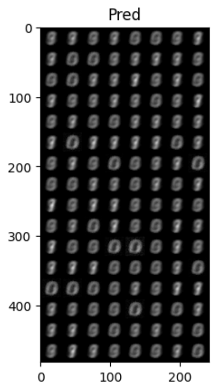
loss curves for compressed_space_size = 128

The loss does indeed decrease, but not to the same values: after 10 epochs, the loss train is 0.070 for a compressed space size of 1, while for compressed_space_size = 128, this loss train is only 0.02 (about three times less). This creates a difference in the image reconstruction during the testing part.

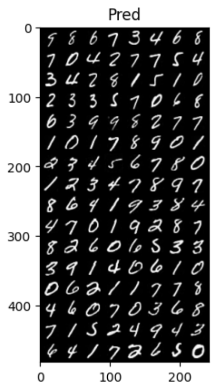
2.3 Reconstruction on a test part



Original set



Reconstructed set, when compressed_space_size = 1

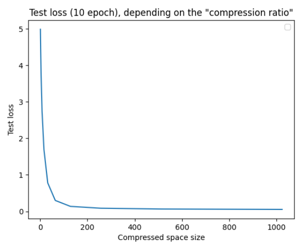


Reconstructed set, when compressed_space_size = 128

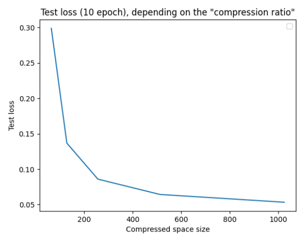
2.4 Conclusions about MLP compression of mnist elements

When the value of compressed_space_size is too low, we see a blurred effect. The model would have needed more training to get a good result. This is consistent with the fact that we require compression over a much larger space, and therefore higher compression, which is not as simple as compressing very lightly.

We have plotted the test loss we get as a function of the compressed space size. We do get the following decreasing curve :



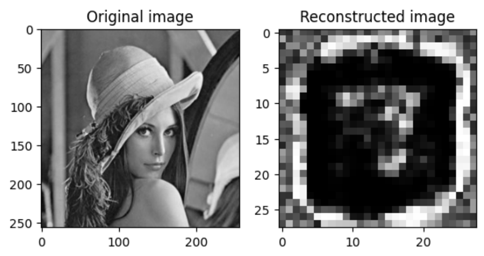
Test loss, depending on the compression ratio



Same curve, with a focus on big value of compression space size.

2.5 Compressing an image that has nothing to do with it

We can try to see what would happen if we use the last trained model (in our case the MLP autoEncoder with compressed space size = 512) on another type of data: an image taken from the internet. We do this on the image we used in the first lab and here is the result:



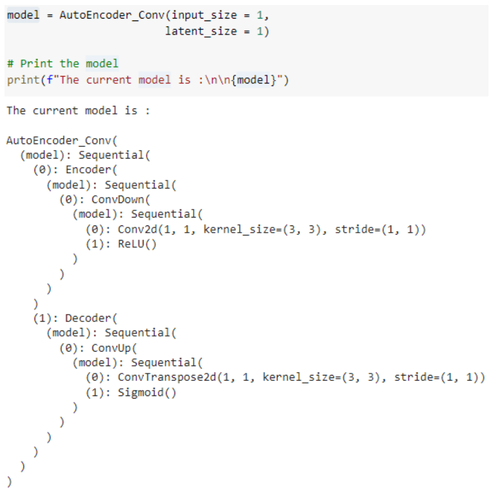
Other type of data

The result is completely inconsistent: the reconstructed image is not at all what we originally had. This result is predictable. Our model is trained on numbers, and in shades of grey. It is not trained at all to compress images of this type!

3 AutoEncoder - Convolutional Layer Style

3.1 Instantiation

We are now working on another mode of AutoEncoder, which will be composed of Conv2D and ReLu layers for the encoder, and ConvTranspose2D, ReLu and a sigmoid for the Decoder part.



```
model = AutoEncoder_Conv(input_size = 1,
                         latent_size = 1)

# Print the model
print(f"The current model is :\n\n{model}")
```

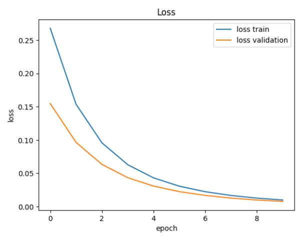
```
The current model is :

AutoEncoder_Conv(
  (model): Sequential(
    (0): Encoder(
      (model): Sequential(
        (0): ConvDown(
          (model): Sequential(
            (0): Conv2d(1, 1, kernel_size=(3, 3), stride=(1, 1))
            (1): ReLU()
          )
        )
      )
    )
    (1): Decoder(
      (model): Sequential(
        (0): ConvUp(
          (model): Sequential(
            (0): ConvTranspose2d(1, 1, kernel_size=(3, 3), stride=(1, 1))
            (1): Sigmoid()
          )
        )
      )
    )
  )
)
```

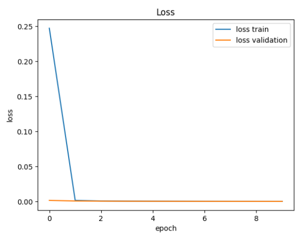
An example of AutoEncoder_Conv, for a latent_size of 1

3.2 Loss results

We can notice that, in the same way as before, increasing the value of the latent_size allows - for the same number of epochs - to have a better reconstruction rate. Here is the loss curve that we obtain for two values of latent_size :



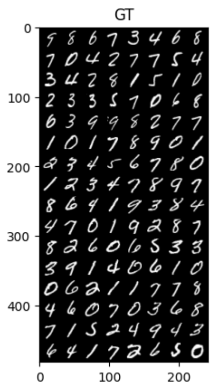
loss curves for latent_size = 1



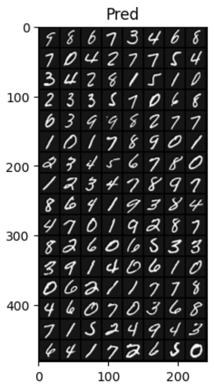
loss curves for latent_size = 128

For latent_size = 1, the training loss is - after 10 epochs - of 0.009, while for latent_size = 1 the loss is 0.00009 (i.e. 100 times less !). We get much more interesting results than before in terms of loss.

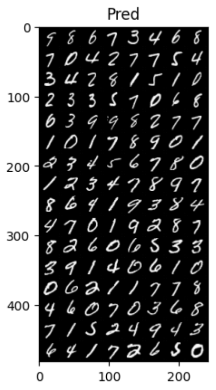
3.3 Reconstruction on a test part



Original set



Reconstructed set, when latent_size = 1



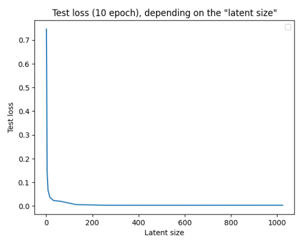
Reconstructed set, when latent_size = 128

This is not as obvious as before, but zooming in on the images shows a better reconstruction for a large value of latent_size. We also notice that the loss goes from 0.55 to 0.005 by increasing the latent_size from 1 to 128 (which is consistent with the previous training loss curve).

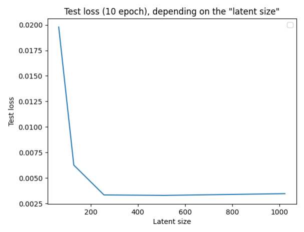
3.4 Conclusions about Conv compression of mnist elements

When the value of latent_size is too low, we do not see a blurred effect.

We have plotted the test loss we get as a function of the latent size. We do get the following decreasing curve :



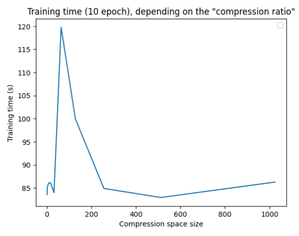
Test loss, depending on the latent size



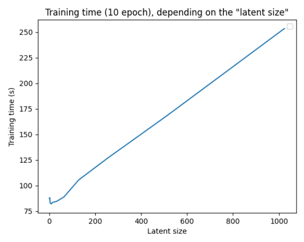
Same curve, with a focus on big value of latent size.

However, this great improvement in loss is not without cost. Let's look at the time it takes to train our models with the two previous AutoEncoder types.

3.5 Comparison of training times



Training time for AutoEncoder_MLP



Training time for AutoEncoder_Conv

We note that the training time for the MLP model is relatively constant (although the peak for n=16 and for n=32 is not explained). This time is consistently less than 90 seconds. However, for the convolutional model, the training time is almost linear, and for a latent size of 1024, this time is 4 minutes.
Thus, even if we can prefer the convolutional model at first, we will have to take into account this training time and perhaps make a compromise on the latent_size which does not need (in our case) to be so large.

3.6 Deeper Model : Adding a layer

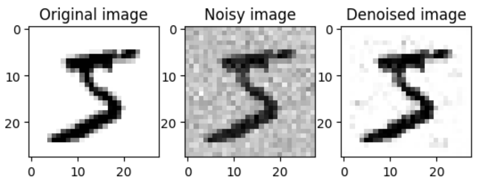
In the instantiation of our model, we could have added more layers on encoding and decoding. So that's what we did. Here is a table that summarises the different results we get for one and two convolutional layers (at encoding and decoding):

	1 layer		2 layer	
Latent size	64	128	64	128
Training time	91,8	107,3	137,6	147,64
Last training loss	0,0003	0,0001	0,0001	0.000027
Test loss	0,019	0,006	0,007	0.0016

We notice that the training time is much longer by adding layers, which is consistent with the fact that there are more meshes in our neural network. At equivalent latent size, the loss is lower. Thus, increasing the number of layers allows us to have a better model in a smaller number of iterations, but this requires more time and therefore resources.

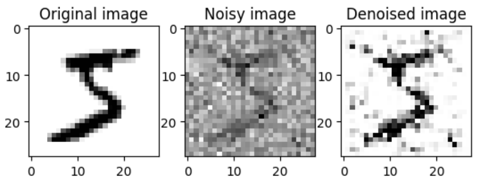
4 Noise suppression

It is possible, after training a model, to use it for noise suppression. This is what we have done in this section and here is the result we get when the noise is not too important:



An example of a denoised image using an AutoEncoder_ConvTwoLayers model

It was indeed possible to reconstruct the original image. This can be explained by the fact that when the model compresses the image, it "knows" that the number is written in black on a white background. It therefore has no interest in keeping all the grey pixels and encodes them as white pixels.
However, if the noise becomes too large, the model has difficulty reconstructing the original image because there are large areas of noise that it doesn't know are not giving us information. This is what happens when too much noise is added:



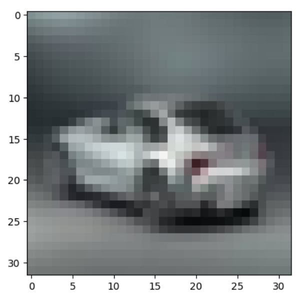
Even noisier image, denoised using an AutoEncoder_ConvTwoLayers model

The model has more difficulty in reconstructing the original image because there are many very dark grey areas which appear to the model to be information about the number.

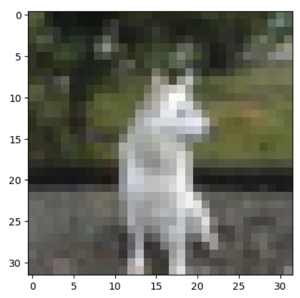
5 Getting some colors

5.1 Presentation of CIFAR10

CIFAR10 is a dataset containing images of size 32*32, and in colour! These images represent various elements as we can see during the test part: cars, animals, etc. Here are two examples of data from this dataset:



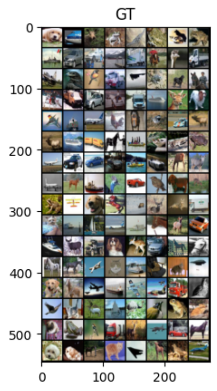
An image from the CIFAR10 dataset



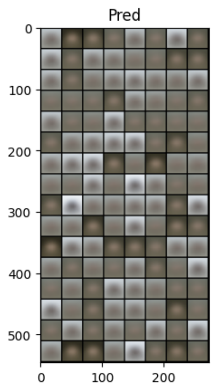
Another image from the CIFAR10 dataset

5.2 What needs to change ?

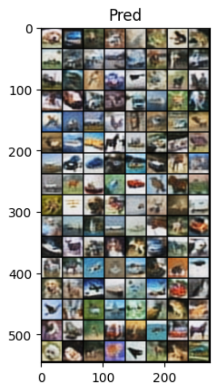
We only have to think about readjusting the size of our vectors: for an MLP, the input size is now 32*32*3, and for the convolutional model, we now have 3 channels.
We have carried out some tests and the results are consistent with previous ones. Increasing the compression space size gives a more accurate result at the end of the training of the model,as we can see on those figures :



Original set



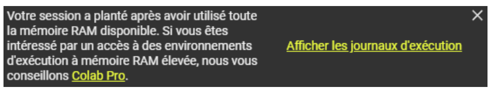
Reconstructed set, when compressed_space_size = 1



Reconstructed set, when compressed_space_size = 128

We notice, however, that for equivalent values, the constructions are slightly worse than when use MNIST numbers. This is also consistent with the fact that a colour (and slightly larger) image is harder to reconstruct than a black and white image.

We wanted to realize the loss curve during the test phase, as well as the time taken to execute this. However, we got an error because we have asked for more computing power than is available with the colab tools) :



Votre session a planté après avoir utilisé toute la mémoire RAM disponible. Si vous êtes intéressé par un accès à des environnements d'exécution à mémoire RAM élevée, nous vous conseillons Colab Pro.

Afficher les journaux d'exécution

Error :(

Of course, it would have been possible to find a roundabout way of constructing the curves, but we didn't take the time to do so (it took us 30 minutes to compile just a few models). So we didn't get these last curves, even though everything suggests that they are very close to those obtained previously.

6 Conclusion

To go further in this lab, we could have tried several things:

- Further increase the number of layers in the convolutional model
- Train for a larger number of epochs
- Test our code with other datasets (such as CIFAR100)
- Create a model to train to recognize the numbers / objects, to see if the compression / decompression allowed us to recognize the image as well

- Plot the test loss and time taken curves when using the models with the second dataset.

One point that we did not manage to calculate, is the real compression rate of the image. We see that it is effective because there is lost information (what we observe in particular during the denoising).