Benjamin Scialom
A20432063

Computer Vision : Homework 4

(CS-512)

I. Problem statement

In this homework, we need to implement a basic convolutional network for classification with « Tensorflow or Keras ». All the tests have to use a GPU framework.
Basically, after building, training and evaluating a CNN, the goal is to classify numbers of the MINST dataset as even or odd number. In other words, the program CNN I have to implement has to learn features extraction and classification.
To put it in a nutshell, the CNN will have to be test on a picture uploaded in the program required.

II. Proposed solution

My solution is to implement the CNN using « **KERAS** » and « Google Colab ». I have a MacBook Pro and, so, don't have access to the GPU. Moreover, « Google Colab is much more powerful than a computer because it uses a strong GPU.
Once I had my model, I was able to work on my computer for the predicating part. I just dowloaded it as a json file.

The first part of the homework is to train the CNN on the MINST data-set. That is to say, to build the CNN and apply the required parameters to it (number of layers, size of the pooling, dropout rate, cross entropy loss, gradient descent optimization, learning rate and epochs).

The initial (default parameters) are :

- layer 1 : 32 filters of kernel 5x5
- layer 2 : 64 filters of kernel 5x5
- pooling by a factor 2
- dropout : 40%
- loss function : cross entropy
- optimizer : gradient descent
- learning rate : 0.001
- epochs : 5

I used for the first layer « relu » and « softmax »for the second layer as activation functions for the second layer because it was more coherent regarding the provided model.

At this point, the parameters are not optimized enough so we have to find how to make the model more efficient. Thus, in the second part, the goal is to find the right parameters in order to make it more efficient and without outfitting the model.

In that way, the following parameters have to be changed :

- the network architecture : number of layers and organization of the layers
- the receptive field and stride parameters
- optimizer and loss function
- dropout, learning rate, number of filters and number of epochs
- adding batch and layer optimization
- weight initializer (Xavier, He)
- features for pretrained model

Finally, I used my pre-trained custom CNN on the image of a handwritten digit (image contains 1 digit only). Of course, some basic image processing can be done before using the CNN to see what it can change if we modify the image.

III. Implementation details

1. Issues and solutions

- Tune parameters on « Tensorboard » involved errors that I couldn't dealt with, so, I moved to « Keras » which is much more high level and understandable for me.

- Linked « Tensorboard » and « Google Colab » was difficult at the beginning but much more easier after looking at few things on the internet. One solution is to create a pipe between the application « ngrok » to « tensorboard ». Of course, since I used keras, I had to add the « tensorboard » callbacks for « Keras » in my code too.

- I spent much time understanding how the layers were linked to parameters, so, I had to look for more details on internet to understand to what parameters correspond, for instance « stride ».

- More generally, one of the main issue is to understand how the libraries (tensorflow and eras) work. In other the set up was complex. I used keras because it was much more soft than tensorflow.

- It was impossible for me to have the recall and the precision by using keras. So I can have the graph and the value for that. I didn't find any solution on the web for that.

- I had a bitwise pre-processing because it improves the labelling process in part 3.

2. Instructions to run the program

First, the goals of the **« cnn.py »** code is to load the MINST data-set, to create a model according to the specified values, to train this model and to make it evaluates the distinction of even and odd number.

Second, the python file **« cnn_test.py »** has multiple role. It loads the pretrained model create, train and evaluate previously and then ask the user to provide an image in order to predict the label of the image (even or odd). Note that pre-processing are applied to the image before labeling.

To make it work, you just have to run the code, then, give the path of your picture as it is recommended in the terminal. The final result will be display on both a picture and in the terminal.

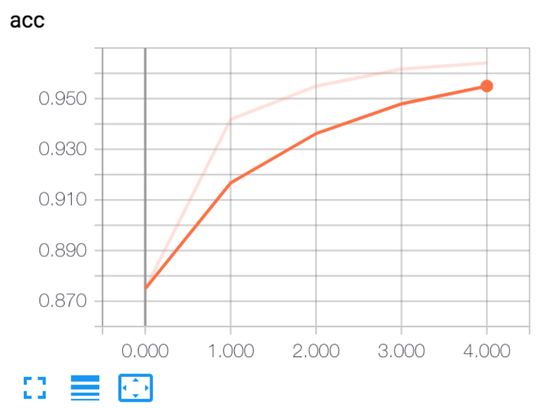
There is an issue with the exit but you can test several number without any issue. It's just the exit that doesn't work, so, you have to push it to stop manually.

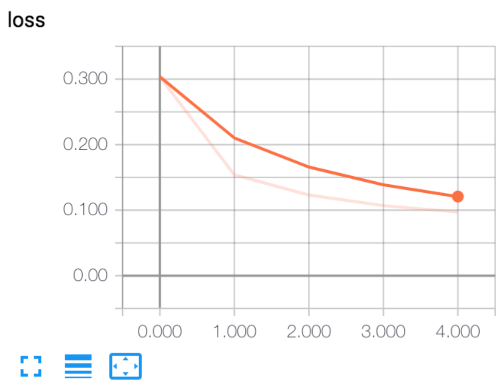
For each epochs : required metrics

IV. Results and discussion

1. Deliverable 1 :

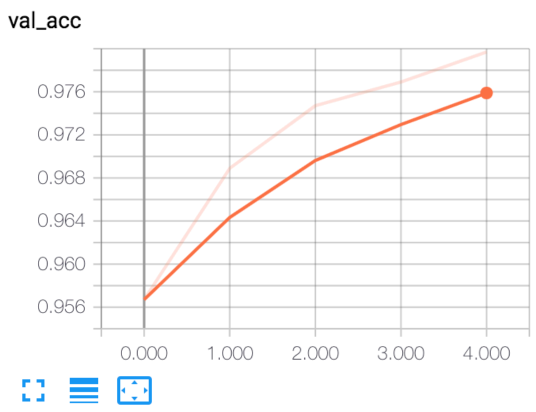
- **Training :**

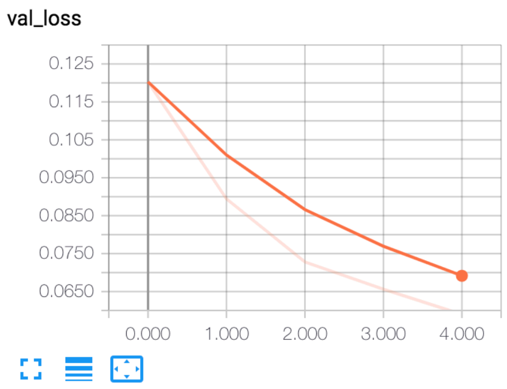




As we can see on the graphs above , the **training accuracy** is **0.9641** at the 5th epoch.
and the **training loss** is about **0.0972** at the 5th epochs.
It is not bad but there are rooms for improvements.

- **Evaluation :** (val <=> eval, it like this if I use keras on colab to have tensorboard)





```
Train on 60000 samples, validate on 10000 samples

Epoch 1/5
loss: 0.3035 - acc: 0.8749 - val_loss: 0.1203 - val_acc: 0.9567
Epoch 2/5
loss: 0.1541 - acc: 0.9418 - val_loss: 0.0894 - val_acc: 0.9689
Epoch 3/5
loss: 0.1232 - acc: 0.9549 - val_loss: 0.0728 - val_acc: 0.9746
Epoch 4/5
loss: 0.1069 - acc: 0.9617 - val_loss: 0.0656 - val_acc: 0.9769
Epoch 5/5
loss: 0.0972 - acc: 0.9641 - val_loss: 0.0590 - val_acc: 0.9797
[0.05897679421380162, 0.9797]
```

Default metrics results

As we can see on the graphs above, the **evaluated accuracy** is **0.9797** at the 5th epochs and the **evaluated loss** is 0.0590.
It is not bad but there are rooms for improvements.

2. Deliverable 2 :

As you can see, in the first part, the results (loss and accuracy) both for the training and the evaluation are good but not optimum.
In this part, the goal is to tune different parameters in order to make it stronger.

Drop rate : A Simple Way to Prevent Neural Networks from Overfitting. Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly.

```
Train on 60000 samples, validate on 10000 samples

Epoch 1/5
loss: 0.2875 - acc: 0.8843 - val_loss: 0.1816 - val_acc: 0.9371
Epoch 2/5
loss: 0.1513 - acc: 0.9479 - val_loss: 0.1120 - val_acc: 0.9658
Epoch 3/5
loss: 0.1069 - acc: 0.9644 - val_loss: 0.0906 - val_acc: 0.9713
Epoch 4/5
loss: 0.0871 - acc: 0.9708 - val_loss: 0.0734 - val_acc: 0.9759
Epoch 5/5
loss: 0.0764 - acc: 0.9746 - val_loss: 0.0639 - val_acc: 0.9789
[0.06386050489544869, 0.9789]
```

drop rate = 0.01

```
Train on 60000 samples, validate on 10000 samples

Epoch 1/5
loss: 0.6591 - acc: 0.6168 - val_loss: 0.5159 - val_acc: 0.8296
Epoch 2/5
loss: 0.5754 - acc: 0.6936 - val_loss: 0.4326 - val_acc: 0.8741
Epoch 3/5
loss: 0.5385 - acc: 0.7197 - val_loss: 0.4001 - val_acc: 0.8976
Epoch 4/5
loss: 0.5080 - acc: 0.7464 - val_loss: 0.3483 - val_acc: 0.9076
Epoch 5/5
loss: 0.4900 - acc: 0.7565 - val_loss: 0.3210 - val_acc: 0.9194
[0.32096313781738284, 0.9194]
```

drop rate = 0.99

We can conclude that the drop rate should not be too high. I have notice that there is not much change until a drop rate of 0.8 but it declines after.

Learning rate :it is defined in the context of optimization, and minimizing the loss function of a neural network.

```
Train on 60000 samples, validate on 10000 samples

Epoch 1/5
loss: 0.3270 - acc: 0.8603 - val_loss: 0.1999 - val_acc: 0.9316
Epoch 2/5
loss: 0.1863 - acc: 0.9319 - val_loss: 0.1344 - val_acc: 0.9546
Epoch 3/5
loss: 0.1382 - acc: 0.9514 - val_loss: 0.1033 - val_acc: 0.9655
Epoch 4/5
loss: 0.1143 - acc: 0.9601 - val_loss: 0.0806 - val_acc: 0.9737
Epoch 5/5
loss: 0.0999 - acc: 0.9658 - val_loss: 0.0693 - val_acc: 0.9767
[0.06925134192705154, 0.9767]
```

learning rate = 0.01

We can conclude that the learning rate don't change a lot the performance in terms of loss and accuracy. However it has an influence on the time.

I forgot to change my learning rate for the rest of the simulation. I am sorry for that.
So keep in mind that the default value of the learning rate wasn't the right one for the rest. It won't change any of my conclusions since the value was 0.003 for the learning rate at this time. The loss is maybe just a little bit over-evaluated.

Epochs : In Deep Learning, an epoch is a hyper-parameter which is defined before training a model. One epoch is when an entire dataset is passed both forward and backward through the neural network only once.

```
Train on 60000 samples, validate on 10000 samples

Epoch 1/10
loss: 0.3141 - acc: 0.8675 - val_loss: 0.1866 - val_acc: 0.9357
Epoch 2/10
loss: 0.1720 - acc: 0.9384 - val_loss: 0.1193 - val_acc: 0.9630
Epoch 3/10
loss: 0.1270 - acc: 0.9571 - val_loss: 0.0907 - val_acc: 0.9713
Epoch 4/10
loss: 0.1052 - acc: 0.9636 - val_loss: 0.0745 - val_acc: 0.9758
Epoch 5/10
loss: 0.0916 - acc: 0.9688 - val_loss: 0.0652 - val_acc: 0.9790
Epoch 6/10
loss: 0.0828 - acc: 0.9716 - val_loss: 0.0608 - val_acc: 0.9799
Epoch 7/10
loss: 0.0774 - acc: 0.9731 - val_loss: 0.0546 - val_acc: 0.9822
Epoch 8/10
loss: 0.0721 - acc: 0.9759 - val_loss: 0.0507 - val_acc: 0.9833
Epoch 9/10
loss: 0.0668 - acc: 0.9773 - val_loss: 0.0478 - val_acc: 0.9831
Epoch 10/10
loss: 0.0649 - acc: 0.9775 - val_loss: 0.0483 - val_acc: 0.9827
[0.04826309872306883, 0.9827]
```

epochs = 10

We can conclude that increasing the number of epochs results in better result. However it takes more time because we train more the model.

Add a layer with 256 filters :

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
loss: 0.3503 - acc: 0.8504 - val_loss: 0.1838 - val_acc: 0.9381
Epoch 2/5
loss: 0.1729 - acc: 0.9364 - val_loss: 0.1075 - val_acc: 0.9642
Epoch 3/5
loss: 0.1256 - acc: 0.9547 - val_loss: 0.0796 - val_acc: 0.9711
Epoch 4/5
loss: 0.1036 - acc: 0.9630 - val_loss: 0.0673 - val_acc: 0.9774
Epoch 5/5
loss: 0.0889 - acc: 0.9689 - val_loss: 0.0562 - val_acc: 0.9806
[0.05620251962766051, 0.9806]
```

With a third layer

We can conclude that adding a layer make the model stronger so it could be a good thing. However it involves more computation and, so, more time.

Number of filters in the layer two layers :

```
Epoch 1/5
loss: 0.2623 - acc: 0.8959 - val_loss: 0.1416 - val_acc: 0.9546
Epoch 2/5
loss: 0.1337 - acc: 0.9542 - val_loss: 0.0976 - val_acc: 0.9682
Epoch 3/5
loss: 0.1010 - acc: 0.9667 - val_loss: 0.0730 - val_acc: 0.9763
Epoch 4/5
loss: 0.0849 - acc: 0.9711 - val_loss: 0.0622 - val_acc: 0.9805
Epoch 5/5
loss: 0.0760 - acc: 0.9746 - val_loss: 0.0555 - val_acc: 0.9821
[0.0555326691865921, 0.9821]
```

64 filters 1st layer
128 second layer

We can conclude that the bigger number of filters we have, the better are the results.
I tried with lower number and the model is less efficient than the basic one.

Optimizer : Optimization algorithms are used in optimizing a Neural Network.

```
Train on 60000 samples, validate on 10000 samples

Epoch 1/5
loss: 0.1090 - acc: 0.9573 - val_loss: 0.0372 - val_acc: 0.9874
Epoch 2/5
loss: 0.0455 - acc: 0.9838 - val_loss: 0.0267 - val_acc: 0.9907
Epoch 3/5
loss: 0.0346 - acc: 0.9879 - val_loss: 0.0219 - val_acc: 0.9926
Epoch 4/5
loss: 0.0278 - acc: 0.9906 - val_loss: 0.0193 - val_acc: 0.9933
Epoch 5/5
loss: 0.0245 - acc: 0.9914 - val_loss: 0.0173 - val_acc: 0.9942
[0.017347140110074544, 0.9942]
```

Adam optimizer

We can conclude that the Adam optimizer is more efficient that the one by default.

Batch size : A convolutional neural network (CNN) doesn't process its inputs one-at-a-time: to increase throughput, it will process the data in batches.

```
Train on 60000 samples, validate on 10000 samples

Epoch 1/5
loss: 0.4600 - acc: 0.7891 - val_loss: 0.3396 - val_acc: 0.8602
Epoch 2/5
loss: 0.3345 - acc: 0.8609 - val_loss: 0.2782 - val_acc: 0.8941
Epoch 3/5
loss: 0.2860 - acc: 0.8853 - val_loss: 0.2419 - val_acc: 0.9131
Epoch 4/5
loss: 0.2491 - acc: 0.9045 - val_loss: 0.2072 - val_acc: 0.9270
Epoch 5/5
loss: 0.2207 - acc: 0.9167 - val_loss: 0.1835 - val_acc: 0.9367
[0.18348954063653947, 0.9367]
```

Batch size = 500

```
Epoch 1/5
loss: 0.2594 - acc: 0.8937 - val_loss: 0.1317 - val_acc: 0.9560
Epoch 2/5
loss: 0.1284 - acc: 0.9557 - val_loss: 0.0815 - val_acc: 0.9724
Epoch 3/5
loss: 0.0953 - acc: 0.9670 - val_loss: 0.0659 - val_acc: 0.9783
Epoch 4/5
loss: 0.0817 - acc: 0.9714 - val_loss: 0.0540 - val_acc: 0.9807
Epoch 5/5
loss: 0.0714 - acc: 0.9750 - val_loss: 0.0461 - val_acc: 0.9840
[0.046121957134082917, 0.984]
```

Batch size = 50

We can conclude that the more little is the batch size, the better it is for the efficiency of the model.

Stride : The filter convolves around the input volume by shifting one unit at a time. The amount by which the filter shifts is the stride. Stride is normally set in a way so that the output volume is an integer and not a fraction.

```
Train on 60000 samples, validate on 10000 samples

Epoch 1/5
loss: 0.1821 - acc: 0.9359 - val_loss: 0.0892 - val_acc: 0.9703
Epoch 2/5
loss: 0.0801 - acc: 0.9720 - val_loss: 0.0568 - val_acc: 0.9814
Epoch 3/5
loss: 0.0631 - acc: 0.9785 - val_loss: 0.0471 - val_acc: 0.9847
Epoch 4/5
loss: 0.0549 - acc: 0.9817 - val_loss: 0.0436 - val_acc: 0.9849
Epoch 5/5
loss: 0.0506 - acc: 0.9830 - val_loss: 0.0483 - val_acc: 0.9831
[0.04829322890844196, 0.9831]
```

stride = (1,1)

We can conclude that the smaller the stride is, the better are the efficiency of the model. I also tried a stride (5,5) and the results were worst than the default set up. By taking a high value for stride parameters, we shift more the filter and so are making a bigger approximation, so, the conclusion is coherent

Initializer :

```
Train on 60000 samples, validate on 10000 samples

Epoch 1/5
loss: 0.2688 - acc: 0.8895 - val_loss: 0.1409 - val_acc: 0.9569
Epoch 2/5
loss: 0.1386 - acc: 0.9510 - val_loss: 0.0919 - val_acc: 0.9725
Epoch 3/5
loss: 0.1059 - acc: 0.9638 - val_loss: 0.0739 - val_acc: 0.9772
Epoch 4/5
loss: 0.0905 - acc: 0.9692 - val_loss: 0.0618 - val_acc: 0.9806
Epoch 5/5
loss: 0.0797 - acc: 0.9731 - val_loss: 0.0556 - val_acc: 0.9821
[0.05558638058155775, 0.9821]
```

He initializer

The He initializer is better than the initializer that was used by default.

Batch normalization :

```
Epoch 1/5
loss: 0.1508 - acc: 0.9478 - val_loss: 0.0489 - val_acc: 0.9832
Epoch 2/5
loss: 0.0673 - acc: 0.9762 - val_loss: 0.0504 - val_acc: 0.9825
Epoch 3/5
loss: 0.0555 - acc: 0.9802 - val_loss: 0.0366 - val_acc: 0.9862
Epoch 4/5
loss: 0.0494 - acc: 0.9826 - val_loss: 0.0331 - val_acc: 0.9883
Epoch 5/5
loss: 0.0431 - acc: 0.9848 - val_loss: 0.0299 - val_acc: 0.9901
[0.02985524965962395, 0.9901]
```

with batch normalization

The batch normalization has a huge impact on the performance of the model. It makes the mode super accurate and with few loss.

Loss function : A loss function is a measure of how good a prediction model does in terms of being able to predict the expected outcome.

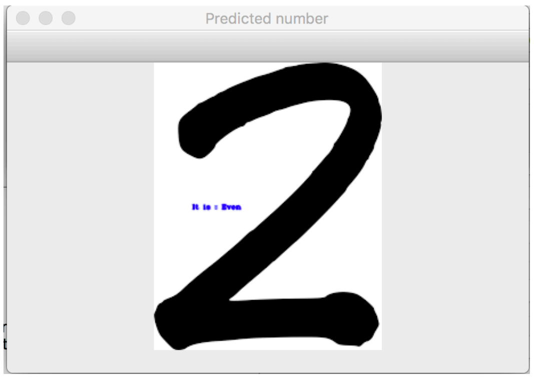
```
Train on 60000 samples, validate on 10000 samples

Epoch 1/5
loss: 0.1245 - acc: 0.8298 - val_loss: 0.0818 - val_acc: 0.8999
Epoch 2/5
loss: 0.0788 - acc: 0.9010 - val_loss: 0.0591 - val_acc: 0.9324
Epoch 3/5
loss: 0.0607 - acc: 0.9273 - val_loss: 0.0463 - val_acc: 0.9484
Epoch 4/5
loss: 0.0498 - acc: 0.9431 - val_loss: 0.0383 - val_acc: 0.9597
Epoch 5/5
loss: 0.0430 - acc: 0.9499 - val_loss: 0.0323 - val_acc: 0.9657
[0.03231380733177066, 0.9657]
```

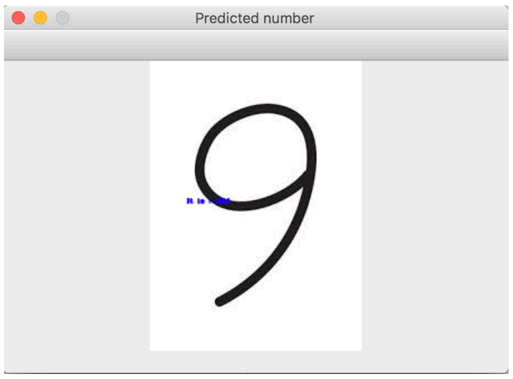
loss function = mean square error

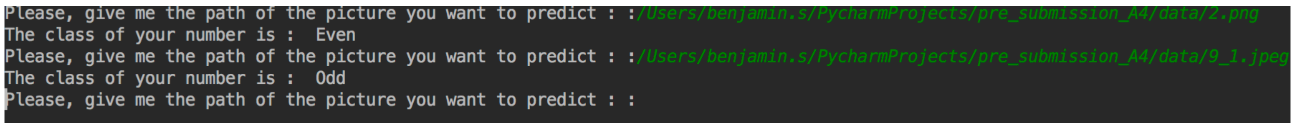
Apparently, the loss function I chose is a little bit less efficient than the one by default.

3. <u>Deliverable 3 :</u>

On the different picture below, you can see that the algorithm for prediction works well.
The pre-processing is essential because it makes the detection easier with the model.







```
Please, give me the path of the picture you want to predict : :/Users/benjamin.s/PycharmProjects/pre_submission_A4/data/2.png
The class of your number is :  Even
Please, give me the path of the picture you want to predict : :/Users/benjamin.s/PycharmProjects/pre_submission_A4/data/9_1.jpeg
The class of your number is :  Odd
Please, give me the path of the picture you want to predict : :
```

V. References

- https://www.quora.com/What-does-stride-mean-in-the-context-of-convolutional-neural-networks
- https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/
- http://cs231n.github.io/convolutional-networks/
- https://www.quora.com/What-does-stride-mean-in-the-context-of-convolutional-neural-networks
- https://keras.io/