ID : 0845034

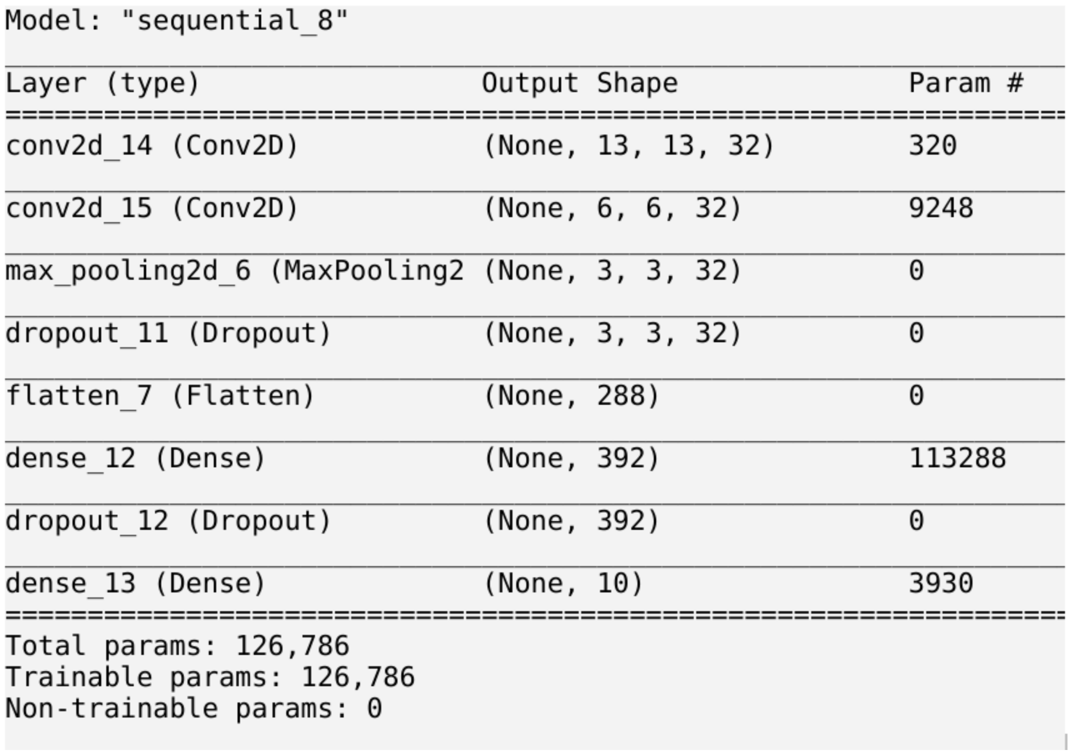
HOMEWORK 2 – DEEP LEARNING

I. Using Convolutional Neural Network for Image Recognition

I build a CNN network in order to classify the images from the MNIST dataset. I used the API Keras to implement it, to earn some time and to have optimized computation. Since my computer doesn't have any Graphic card, it would've been to long to implement it manually and to make it run on this kind of large dataset. Even using Keras, one epoch takes up to 30 second depending on the architecture of my CNN.

Here are the results of my CNN using the following architecture :

- 2 convolutionnal layers : filters number = 32, kernel size = 3, stride = (2,2), activation = relu
- 1 pooling layer : pooling size = (2,2)
- 2 dropout layers
- 1 flatten layer
- 2 dense layers : one dense has 392 units with relu activation, and the other one has 10 units with softmax activation.



```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 13, 13, 32)        320
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 6, 6, 32)          9248
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
dropout_11 (Dropout)         (None, 3, 3, 32)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_12 (Dense)             (None, 392)               113288
_________________________________________________________________
dropout_12 (Dropout)         (None, 392)               0
_________________________________________________________________
dense_13 (Dense)             (None, 10)                3930
=================================================================
Total params: 126,786
Trainable params: 126,786
Non-trainable params: 0
```

Figure 0. Architecture summary

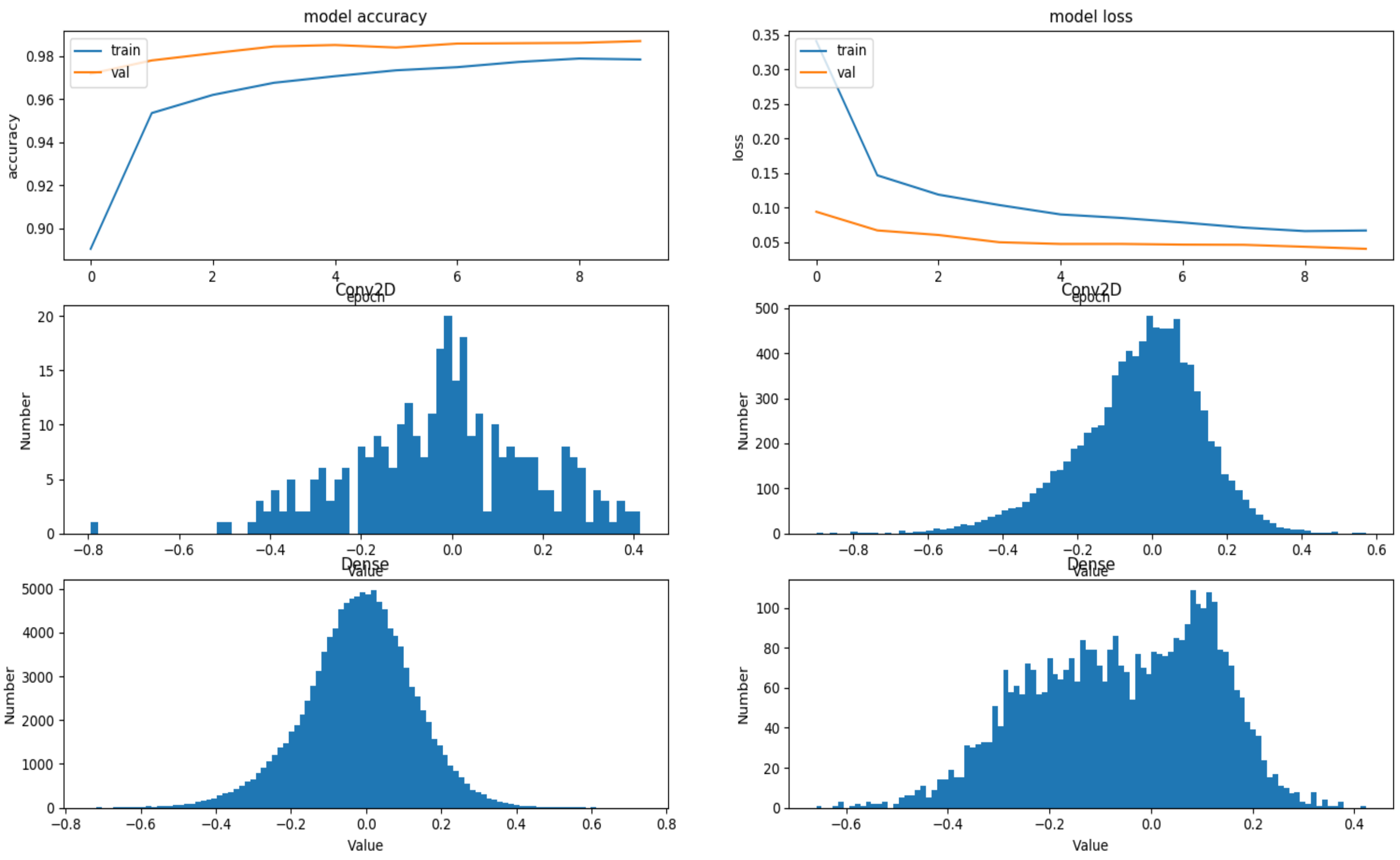


Figure 1. Results of the CNN, with weight details

Reducing the number of filters increase the error rate, and therefore decrease the accuracy. But it makes the computation faster. I tried to implement the same architecture with 8 filters instead of 32, the results are still very good, and the computation are a little faster. The following figure shows the weight with 8 filters for the convolutionnal layers, other parameters are the same.

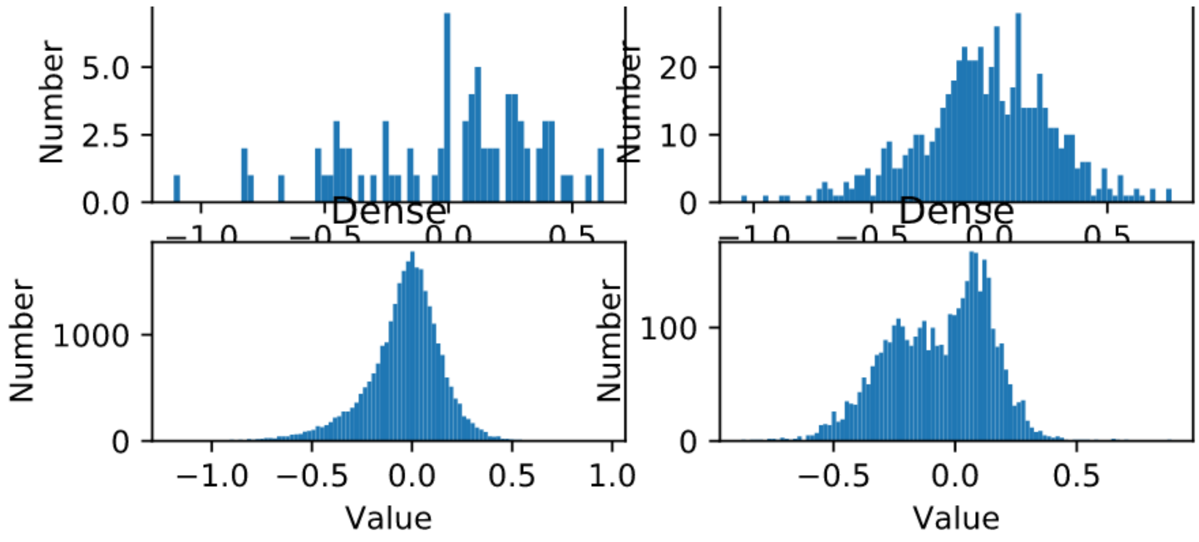


Figure 2. Weight repartition using less filters

On the following figure, you can see histograms of the weights, with stride increased to (4,4) instead of (2,2). The error is increasing as well, and the accruracy a bit lower. But as for the filters, it makes the computation slightly faster.

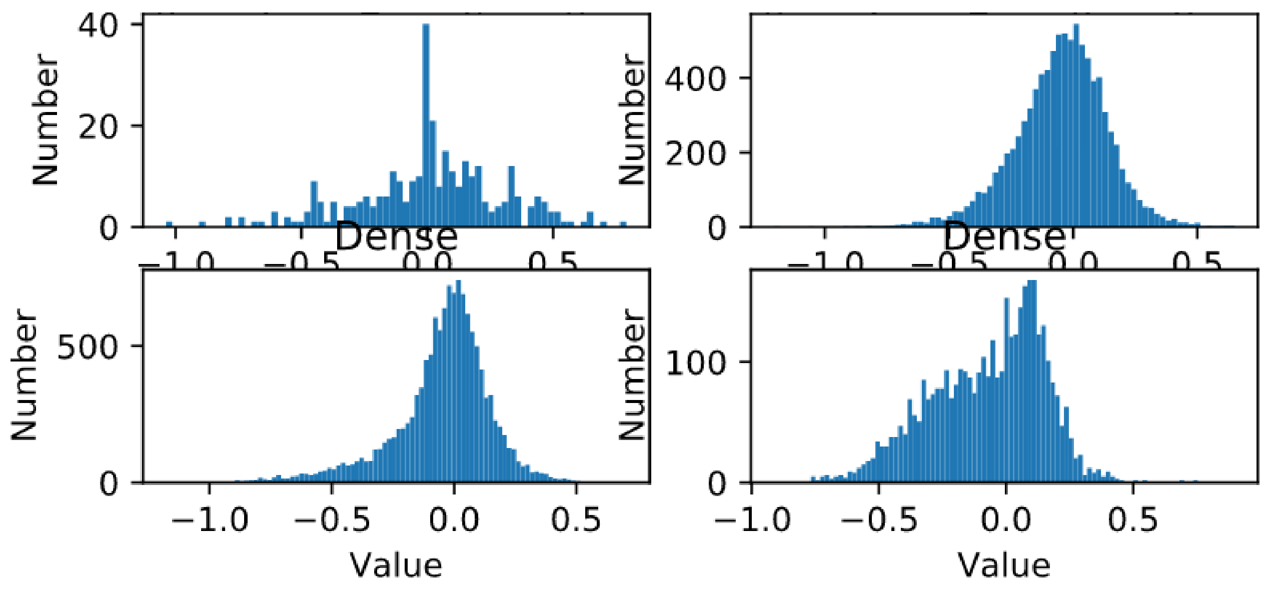


Figure 3. Weight repartition using a larger stride (4,4)

Example of missclassified images :

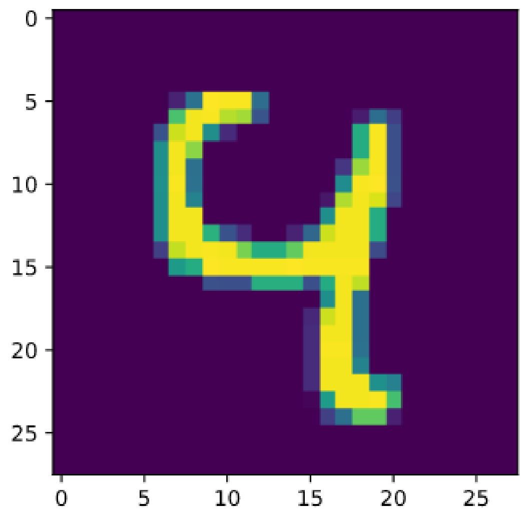


Figure 4. Predicted 4 but is actually 9

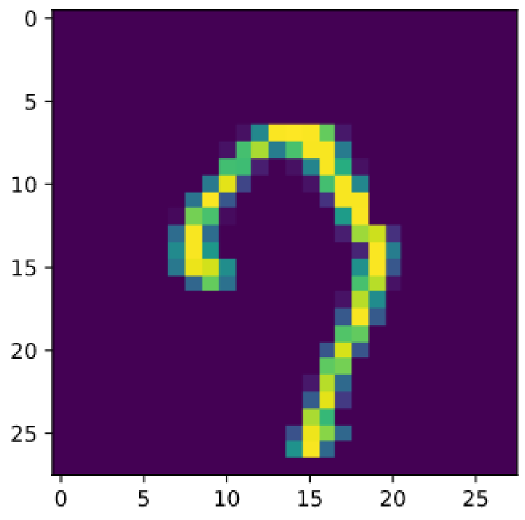


Figure 5. Predicted 7 but is actually 9

We see that for these missclassified numbers, there are legitimate doubt. Prediction of 4 can be explained by the fact that the top part of the digit is not closed. But the curve suggest that it will be a 9. So we see that the quality of the handwritten digits explain that our prediction is not perfect.

Aside from that, looking the prediction of our CNN for these images, the prediction (probability between 0 and 1) is very close between two numbers. For example, the left figure has around 0.9 for both 4 and 9, a tiny difference makes it predict 4 instead of 9, which suggest our CNN is hesitating between 4 and 9. It is the same for the right figure.

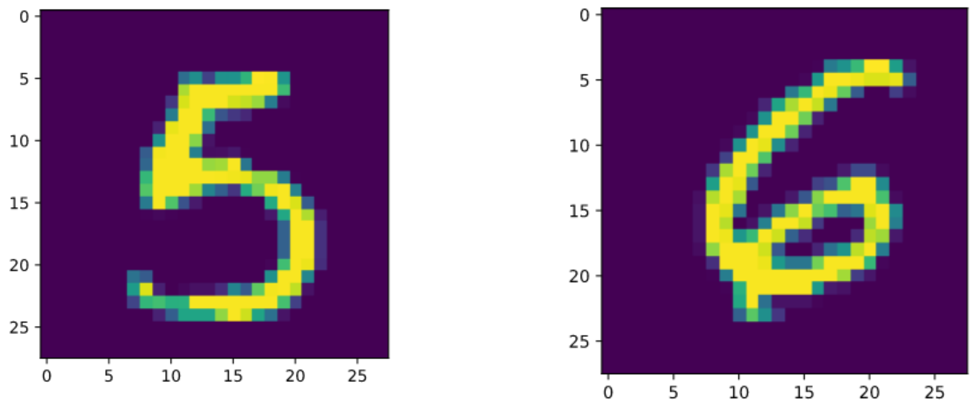
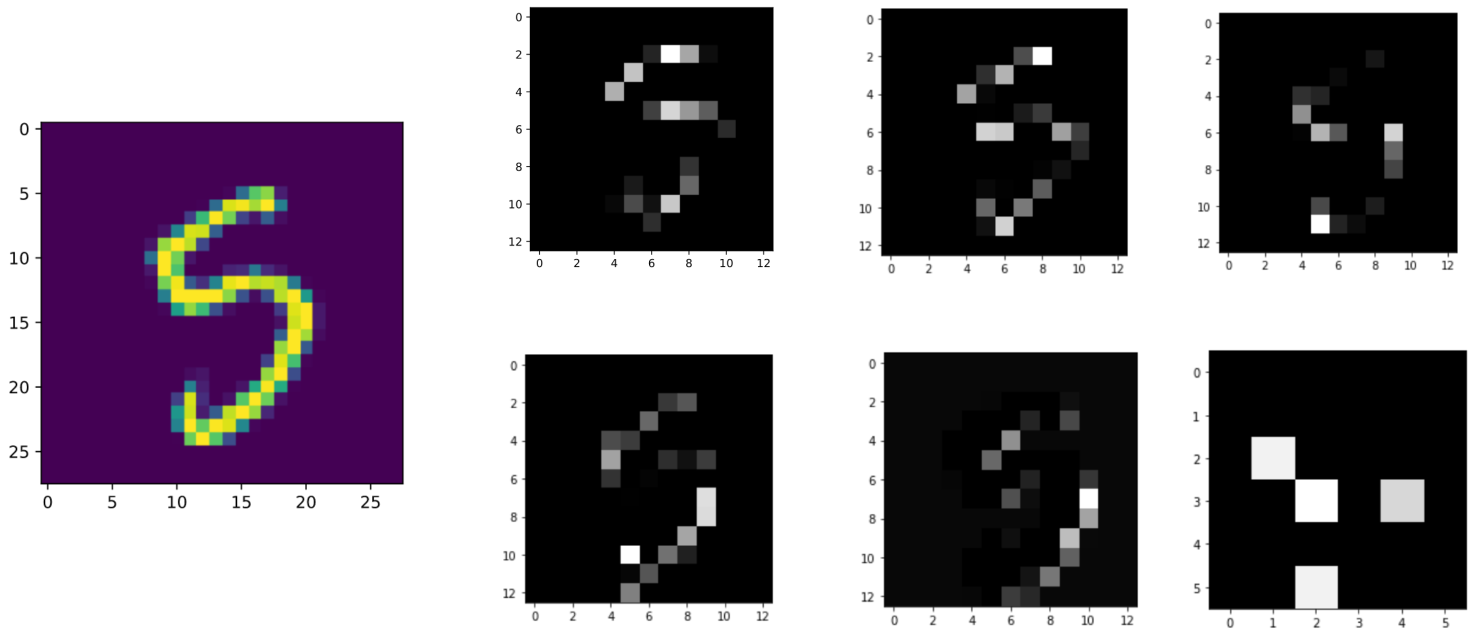


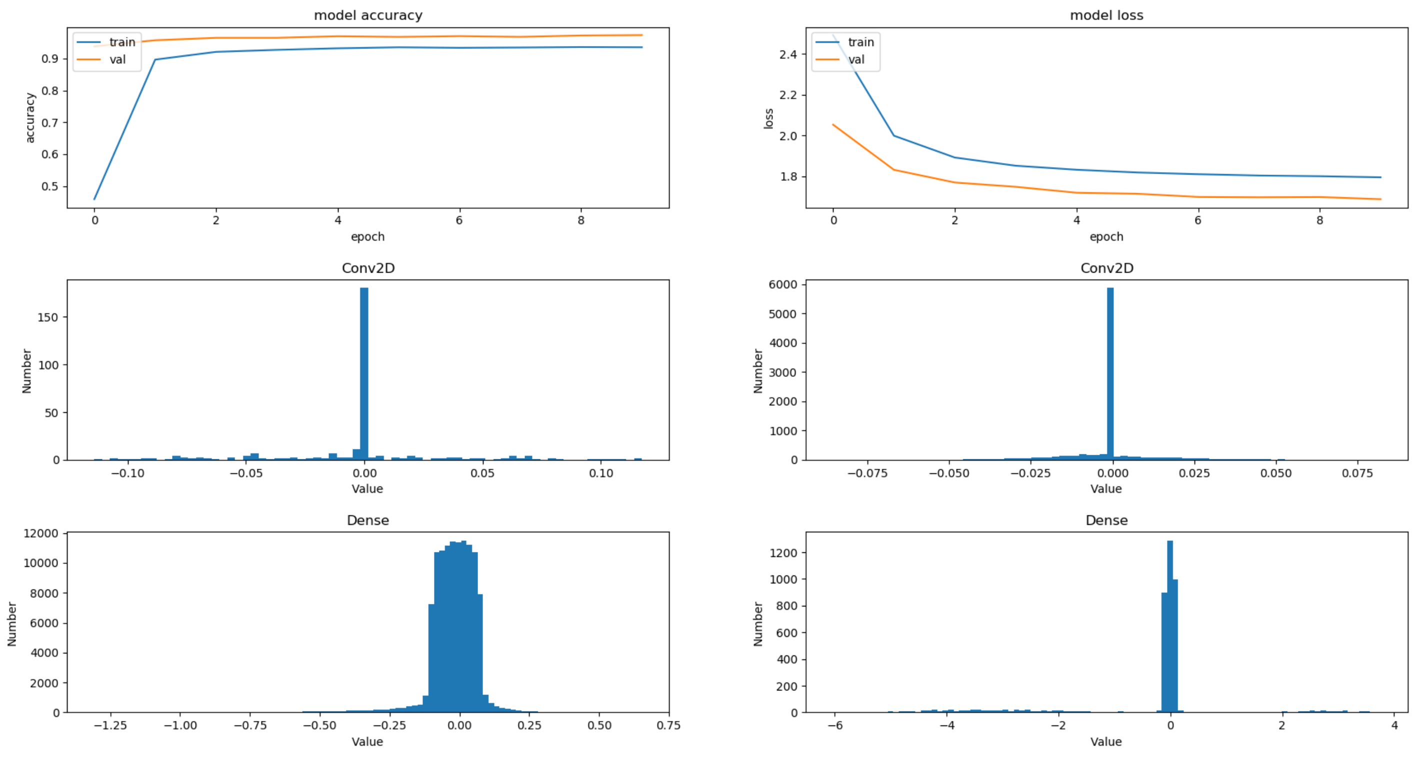
Figure 6. Right classified digits

Feature maps of a right classified number :



We can see that the first layer keeps a lot of information from the initial picture. We are still able to recognize the digits it refers to. But as we go to a deeper layer, we are getting more abstract images, until we can't recognize what is the digit. The figure carry no more informations about the aspect of the number, but more abstract information such as the curve, borders, corners and so on. It contains informations important in order to determine the class of the digit.

Adding the regularization :

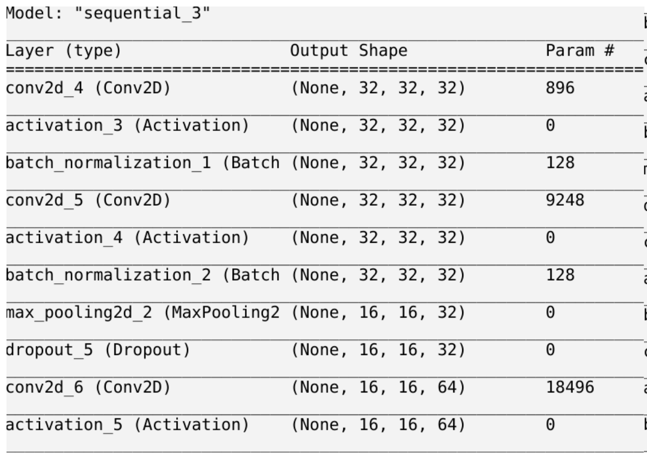


We observe that the weights are all closer to 0, which is the aim of regularization. Therefore, it worked well. The accuracy is still as high ! But the loss is higher, which is because the weight have been reduced and fit not as much as without regularization. Note that it avoid overfitting to regularize.

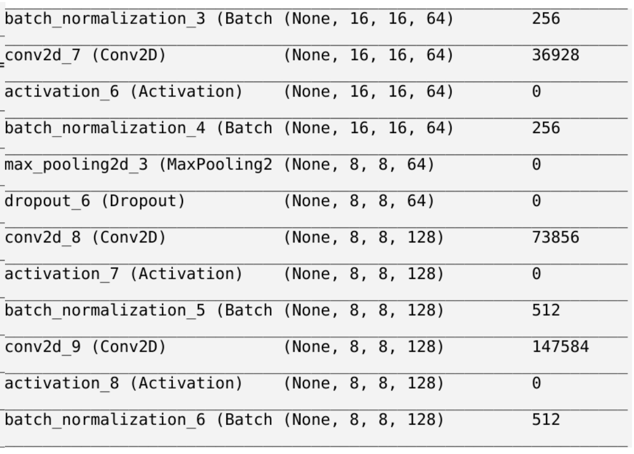
I could improve my CNN by adding more layers, but it would largely increase the time to compute eveything. The results are already really fine so I kept my architecture as it is.

Part 2 – CIFAR-10

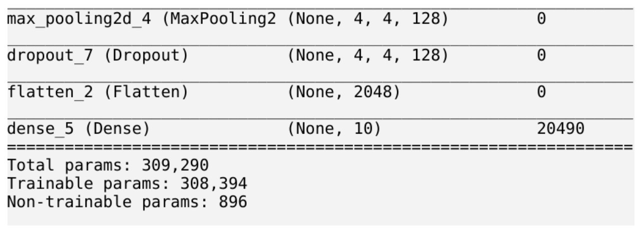
I first tried to use a similar architecture as for the MNIST dataset, but I hadn’t any really good result. I managed to have around 0.5 accuracy but I felt like I could do better with another architecture. Based on research on internet and reading of several forums, I tried the following architecture :



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_3 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
activation_4 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_5 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
```



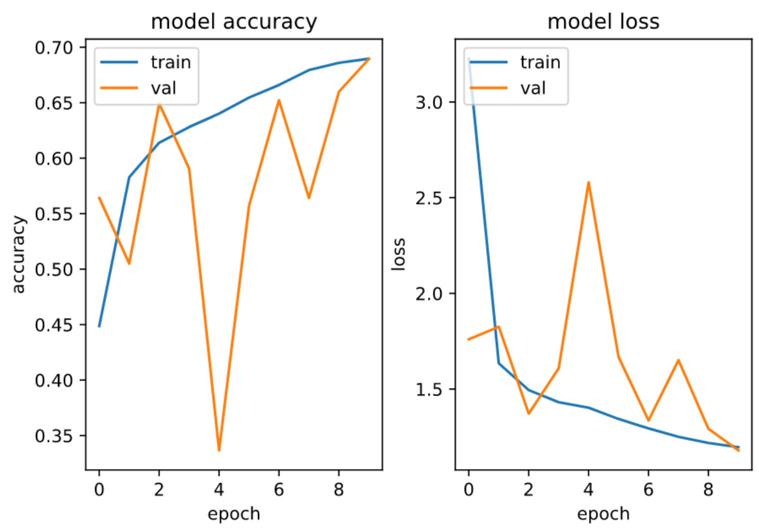
```
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
activation_6 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_6 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
activation_7 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
activation_8 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_6 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
```



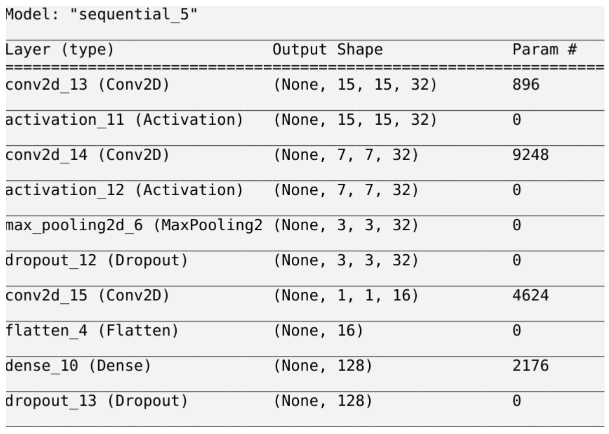
```
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_7 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                20490
=================================================================
Total params: 309,290
Trainable params: 308,394
Non-trainable params: 896
```

The computation was really long, around 30 minutes for ten epoch. So I stopped at ten epoch.

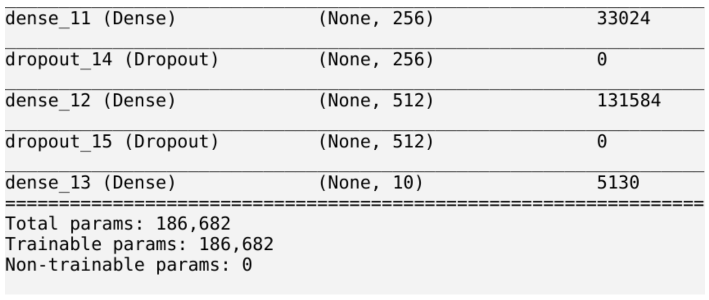
I had the following results :



I was really not satisfied with these, because the validation set was really not stable, probably because the model is overfitting the training set. So I went back to my first architecture and tried to triple the number of epoch. Here is my architecture, much simpler than the previous one :

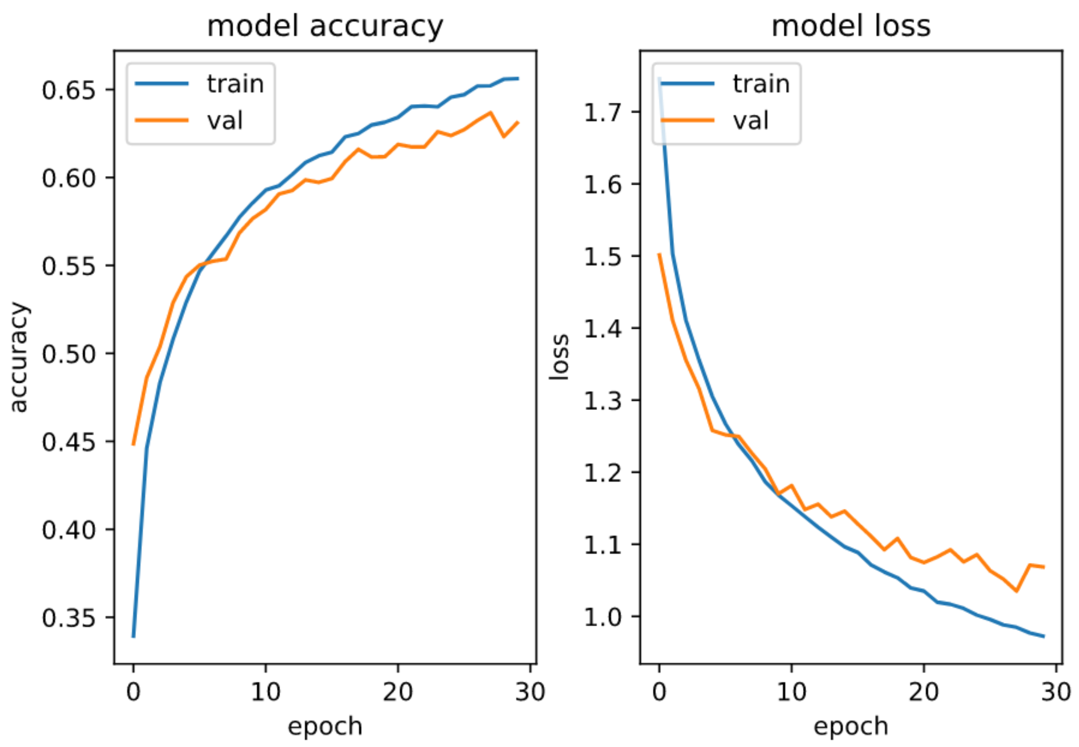


```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 15, 15, 32)        896
_________________________________________________________________
activation_11 (Activation)   (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 7, 7, 32)          9248
_________________________________________________________________
activation_12 (Activation)   (None, 7, 7, 32)          0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
dropout_12 (Dropout)         (None, 3, 3, 32)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 1, 1, 16)          4624
_________________________________________________________________
flatten_4 (Flatten)          (None, 16)                0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               2176
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0
_________________________________________________________________
```

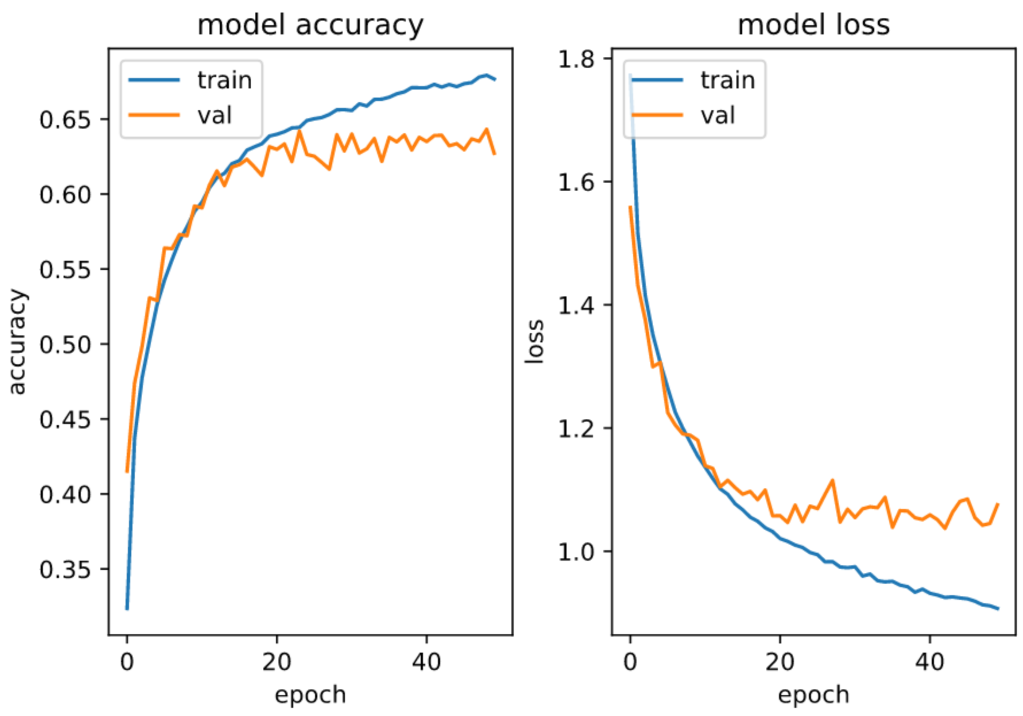


```
_________________________________________________________________
dense_11 (Dense)             (None, 256)               33024
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_12 (Dense)             (None, 512)               131584
_________________________________________________________________
dropout_15 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_13 (Dense)             (None, 10)                5130
=================================================================
Total params: 186,682
Trainable params: 186,682
Non-trainable params: 0
```

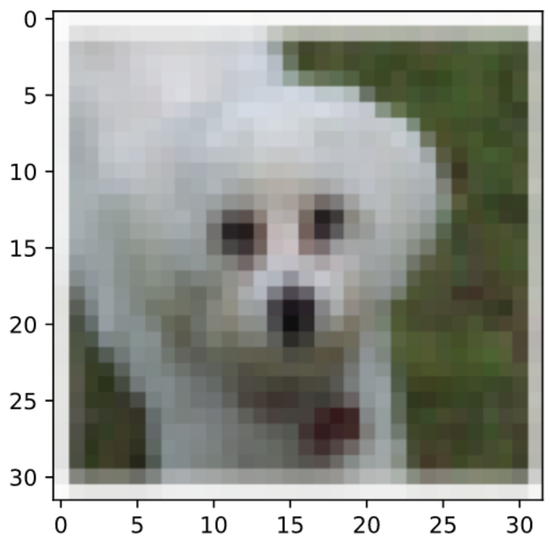
I had the following accuracy and loss with 30 epoch, for 10 minutes of computation



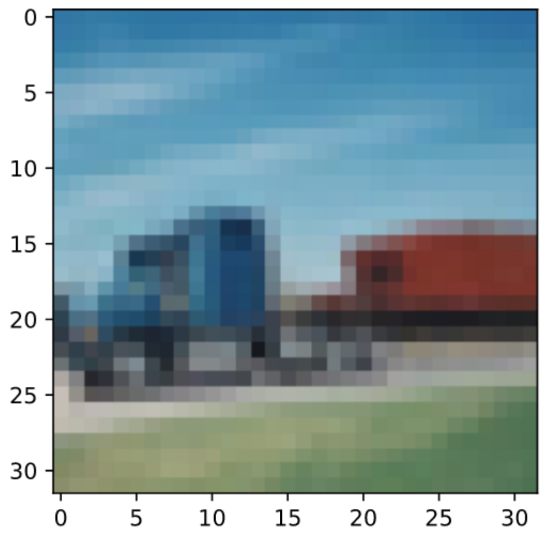
Since my results were still increasing decently, I tried to increase another time the number of epoch to 50. Following figure shows my results :



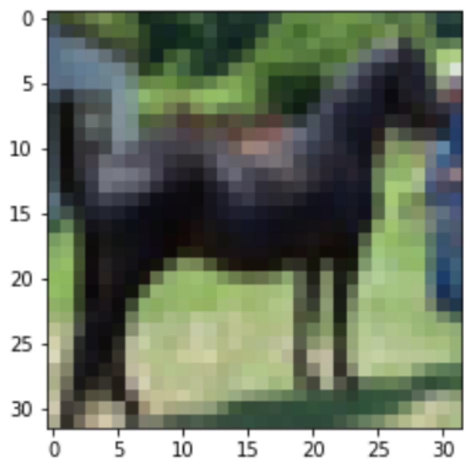
Missclassified image



This image was classified as being a cat, whereas it is a dog.

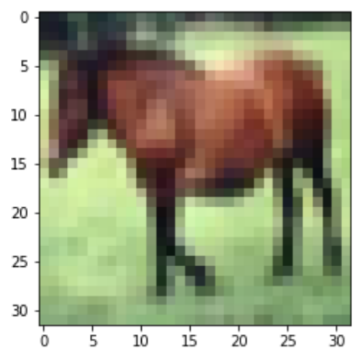


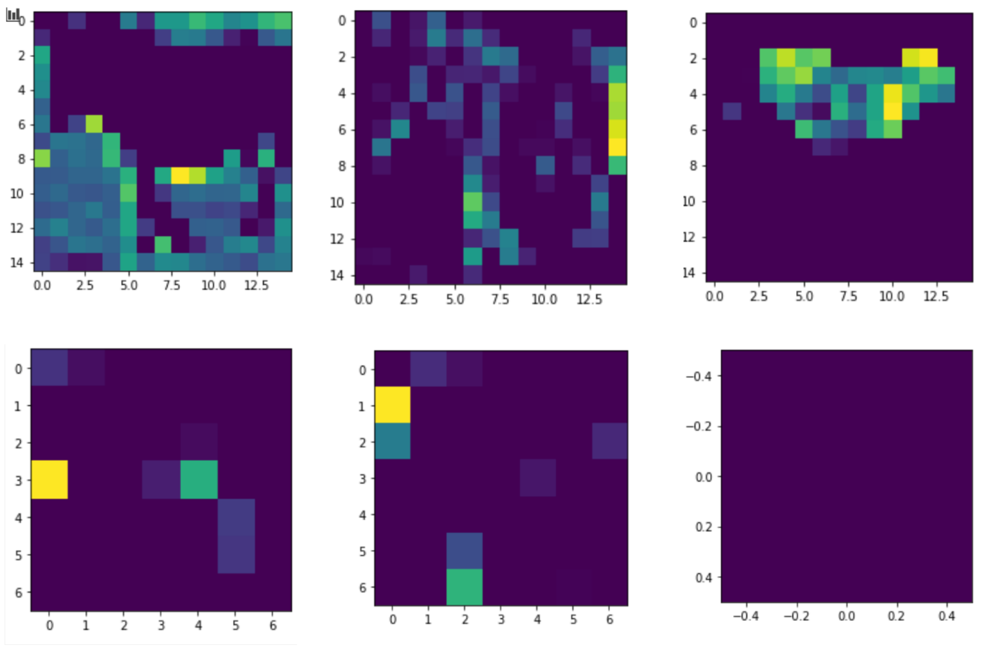
This image was classified as being an airplane, whereas it is a truck.



And this image was correctly guessed by our CNN as being a horse.

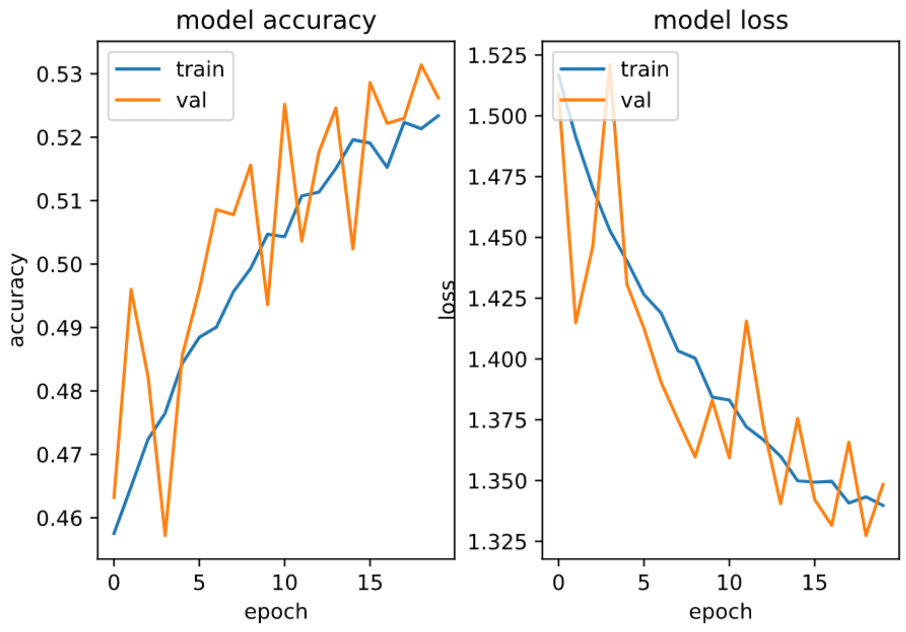
Feature maps





Adding regularization

I tried to add a regularization term on every convolutionnal layer and fully connected layer but I had a bad result. So I think it was underfitting the data way too much, so I just added a regularization term on a few layer and had the following results :



Bias is getting lower but the variance is increasing, which is a known phenomen.

Preprocessing

The data were not really ready to use directly from the downloaded file. I first had to use pickle to load the data. There were 5 separated files to use, so I decided to concatenate their content into two large numpy array for the images (one training set and one validation set) and another one for the labels (as for the images, one training set and validation set). I used keras to one-hot vectorize the labels, and I normalized the array containing the pictures data. Once everything was done I was ready to use my CNN to fit my classifier using keras again.
I wrote my preprocessing in a separate file, which needs the initial files in the same directory to output new files corresponding to my processed data. I wrote my new file using pickle as well.