Lab 2:
Face Detection
using fully connected neural networks

Fergal FENEUIL
Dorian HOUDELETTE
Rémi LEROY

December 2020

1 Introduction

In this lab, we will be using fully connected neural networks to detect faces in images. We will use the neural network as an imagette classifier : given an imagette, the network will predict whether it is a face or not. We will be experimenting with various parameters, such as the number of hidden layers, the number of neurons in each layer, and several activation functions and optimizers.

2 Data set creation

To train our neural networks, we will first need a balanced data set of imagettes. We built our data set using both FDDB and Wider Face. To test the impact of imagette resolution on detection, we made four versions of our data set, with resolution of 8x8, 16x16, 32x32 and 64x64 pixels respectively.

The "positive" imagettes (imagettes that represent a face) are extracted from the images using the original data sets face annotations. We only keep faces that have a sufficient resolution (32x32 pixels). This ensures that networks that will be trained with higher resolution will not be trained using badly resized 8x8 imagettes. The "negative" imagettes are extracted from portions of images which do not contain faces. We made sure that our data set is balanced, with roughly as many negative imagettes as positives.

We split our dataset in a training set, and a validation set. The training set contains 39090 negatives and 39422 positives, while the validation set contains 10061 negatives and 10134 positives. The four version of our data set (8x8, 16x16, 32x32, 64x64) contains exactly the same imagettes, only resized accordingly.

3 Training and experimentation

3.1 First model

Our first neural network is composed of two dense hidden layers of 64 neurons each and a single neuron as an output layer. It uses the ReLU activation function on both hidden layers, and the sigmoid on the output. It was trained on 20000 16x16 imagettes for 31 epochs, after showing no sign of further improvements. It achieved an accuracy of 0.8734 during training, and 0.848 on the test set (showing signs of minor overfitting).

	precision	recall	f1-score
not face	0.757	0.956	0.845
face	0.940	0.693	0.798
accuracy			0.824
macro avg	0.848	0.824	0.821
weighted avg	0.848	0.824	0.821

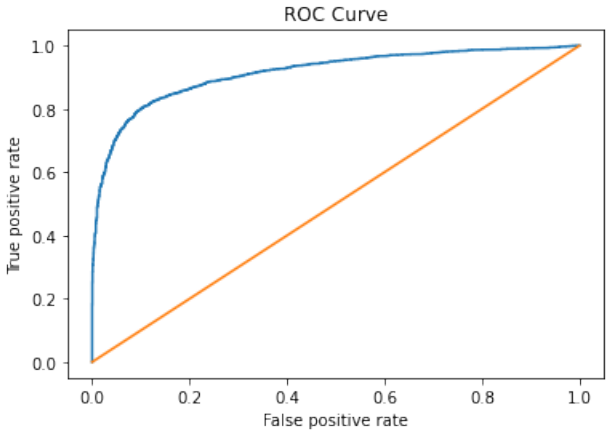


Figure 1: Results of our first model

This model is quite simple and its performances are correct, yet not excellent. From now we will use it as a basis to test the impact of various parameters in the hope of achieving better performances.

3.2 Impact of the number of layers

To have an idea of the impact of the number of layers, we built eight models, each one containing a different number of layers (from one to eight). All these layers are dense and contains 32 neurons. We trained all these models in the same way we trained our first model.

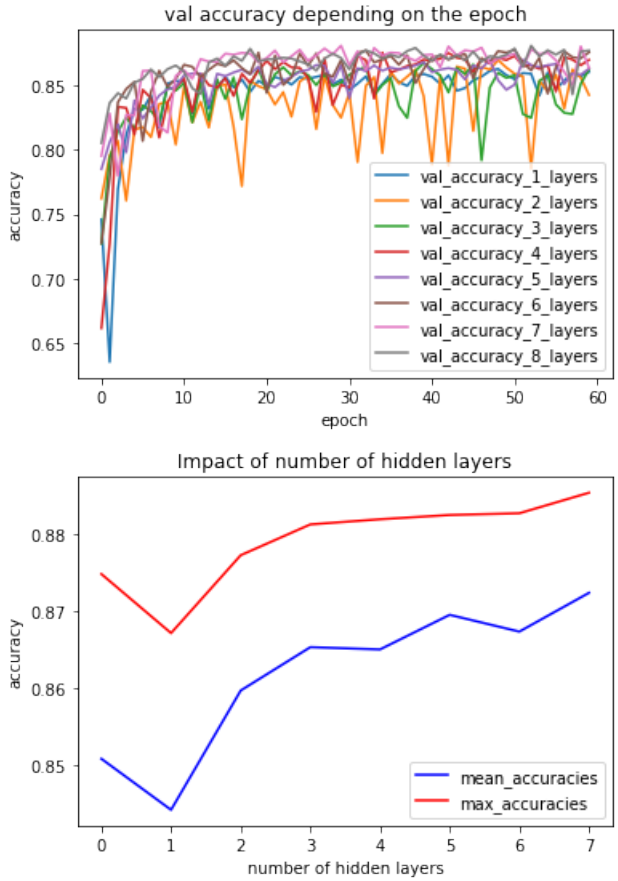


Figure 2: Results with respect of the number of layers

We observe that the accuracy of the models seems to grow slightly with the number of layers, though it does not have much impact. Once the number is sufficient (around 3-4 layers), the model performances seems to be plateauing. 5 layers seems to be the optimal parameter.

3.3 Impact of the number of neurons per layer

We want to know how the number of neurons per layer affect the results. To do so, we tested five different models : each one has two dense layers with varying number of neurons. The models have 8, 16, 32, 64, 128, 256 and 512 neurons per layer respectively.

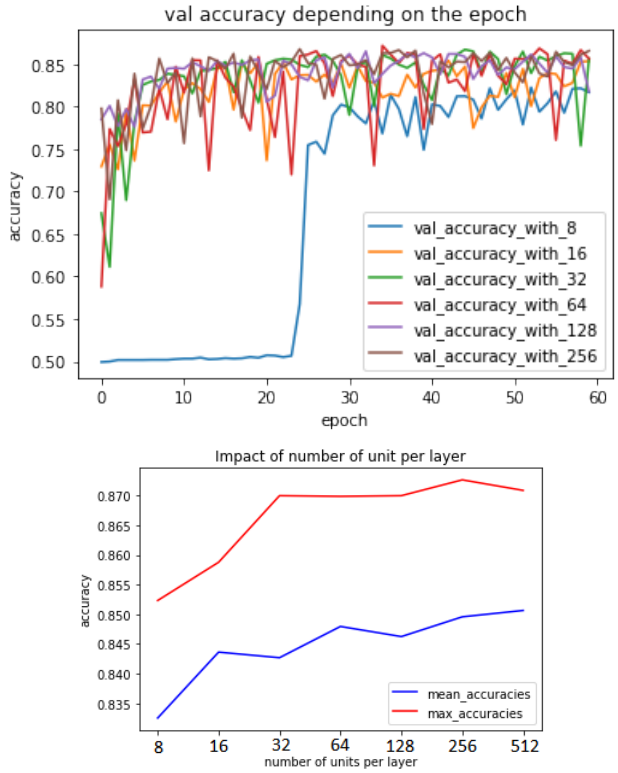


Figure 3: Results with respect of the number of neurons

There is a difference in results mainly between the model with 16 neurons per layer and the model with 32. However, adding more neurons does not affect the results much. We choose 128 neurons for the following part for efficiency purpose.

3.4 Impact of input resolution

To test the impact of input resolution on our networks results, we used the different versions of our data set, with resolution of 8x8, 16x16, 32x32 and 64x64 pixels respectively. The models we used have three dense hidden layers of decreasing size : 64 neurons, then 32, then 16. For hidden layers, we use the ReLU activation function.

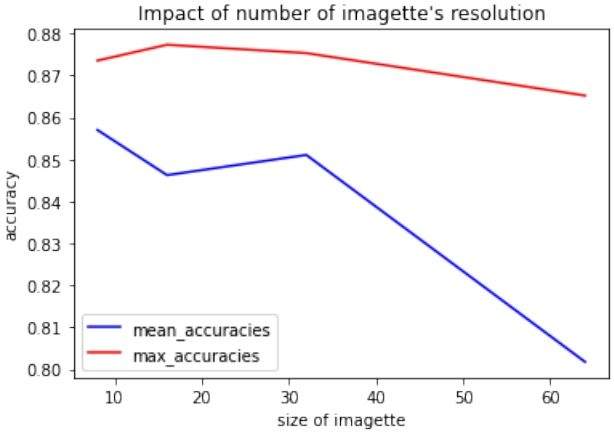
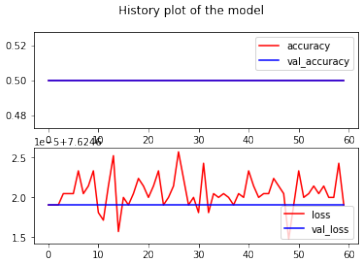


Figure 4: Results with respect of the input resolution

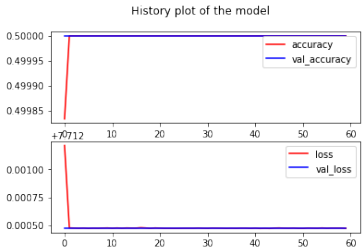
These results are surprising : we could expect that using a higher resolution would mean getting better performances. Instead, the performances of the model trained on 64x64 imagettes are significantly lower than others. It may be linked to the way we constructed our data set : because we kept imagettes with an original resolution of 32x32 at least, the 64x64 data set is the only one that contains resized imagettes. As a consequence several neighbour pixels will repeat and that might constitute noise for our network. The model with the best max accuracy seems to be the one that use 16x16 imagettes. We may also consider that more complex images could require more training.

3.5 Impact of activation functions

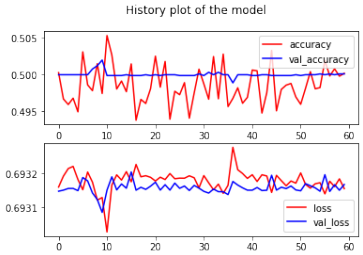
Using a model with two hidden layers, we tried several combinations of activation functions for both hidden layers and the output layer: ReLU, ELU, SeLU, sigmoid, tanh and softmax.



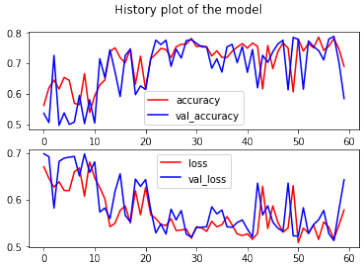
(a) relu on hidden, softmax on output



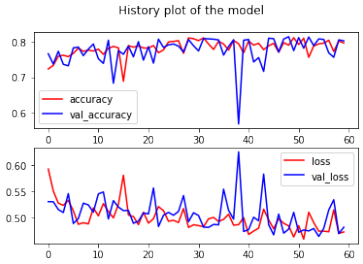
(b) relu on hidden, tanh on output



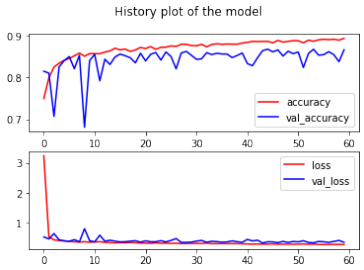
(c) softmax on hidden, sigmoid on output



(d) tanh on hidden, sigmoid on output



(e) sigmoid on hidden, sigmoid on output



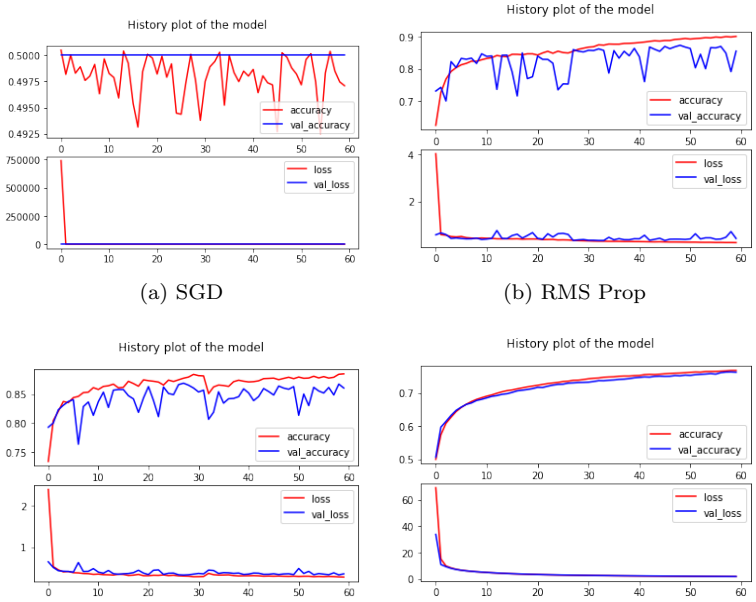
(f) relu on hidden, sigmoid on output

Figure 5: Training history depending on the activation functions

All the models we tried that do not use sigmoid on the output layer are not able to train at all and have a final accuracy of 0.5. The model that use softmax on hidden layers is also not able to train at all. The models that use sigmoid on the output, and either tanh or sigmoid on the hidden layers have sufficient performances but their training is quite unstable : their accuracy fluctuate a lot across epochs. The best combination seems to be relu on hidden layers, and sigmoid on the output : this model shows good results after a few epochs, and keeps improving at a steady rate without oscillations.

3.6 Impact of the optimizer

We tried various optimizers to train a model with two hidden layers : SGD, RMS Prop, Adam, Adadelta, Adagrad, Adamax and Nadam.



(a) SGD

(b) RMS Prop

(c) Adam

(d) Adadelta

Figure 6: Training history using different optimizers

It seems like all optimizers give similar results, with the exceptions of SGD. Using the Adam optimizer, we also tried different values for the learning rate.

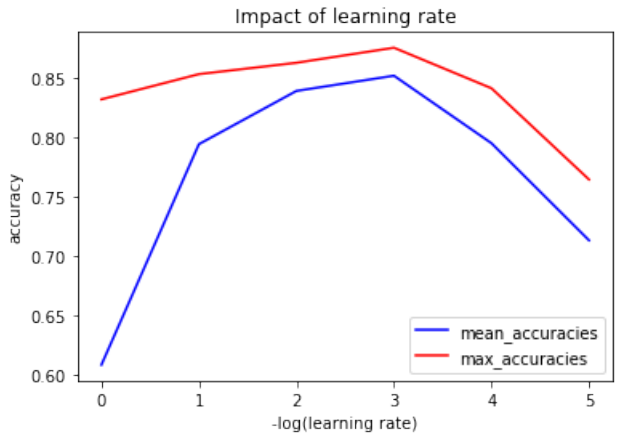


Figure 7: Impact of the learning rate

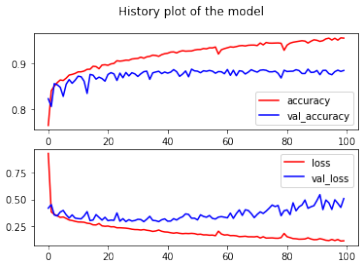
The performances of the model are lower when the learning rate is either too low or too high. A learning rate of 0.001 seems to be optimal.

4 Conclusion

Our previous experiments highlighted values for each of the different hyper-parameters that seem optimal. By combining all these "optimal" parameters into one model, we hope to achieve better results.

We built a model with these parameters :

- 5 hidden layers composed of 128 neurons each
- 16x16 resolution for input imagettes
- ReLU activation function for hidden layers
- Sigmoid activation function for the output layer
- Adam optimizer with a learning rate of 0.001



(a) Training history

	precision	recall	f1-score
not face	0.898	0.867	0.882
face	0.871	0.902	0.886
accuracy			0.884
macro avg	0.885	0.884	0.884
weighted avg	0.885	0.884	0.884

(b) Final performance evaluation

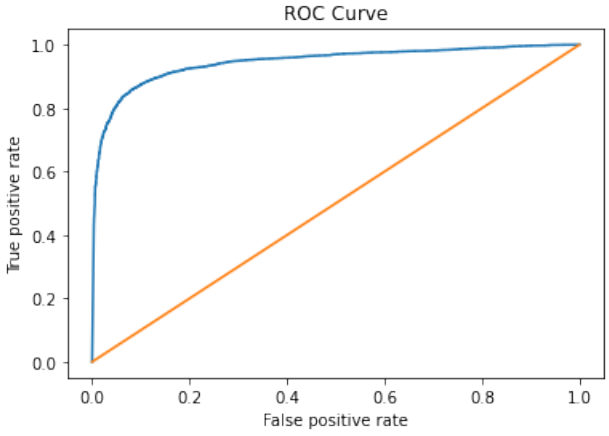


Figure 8: Performances of our best model

As we expected, combining all the optimal parameters resulted in a better model. This model has a final accuracy of 0.884 on the test set, which a bit is better than our first model (0.848).

We used this model on the Head Pose data set to test its sensibility to face orientation.

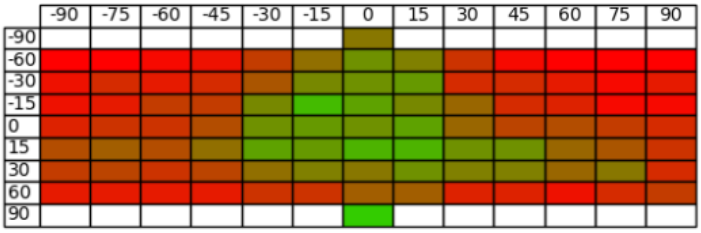


Figure 9: Proportion of detected faces depending on the face orientation (Light Green = 100%, Red = 0%)

As we can see, this model is very sensible to face orientation that are over 30 degrees. These results are similar to the ones we had with the Viola Jones detector. They are yet more "random" with some unexpected variations.

5 To go further

In all our experiments, we used neural networks as imagette classifiers. To actually detect faces in a picture, we use a sliding window approach to extract imagettes at multiple scales in the picture, so that we can classify them using our network.

We only considered square windows since it would be too to iterate on each possible width and height. However, this restriction may reduce the performances of our classifier since it has been trained on rectangular faces that were resized to squares. Then our model will classify better inputs that fit exactly the shape of the face, whereas we give squares that are unlikely to frame perfectly faces. One solution to this issue would be to train our model on faces that has not been deformed: instead of considering the deformed rectangular face, we consider a square contained in the rectangle. In this way the model is more likely to recognize a face in a square window. However, the sliding window is still not really efficient. Firstly, we cannot implement a real time detection using Python since iterating through each window is really slow. Using another language may fix the problem. In addition, there are a lot of false positive, since even with 88% of accuracy, it means many windows which would be considered as valid while not.

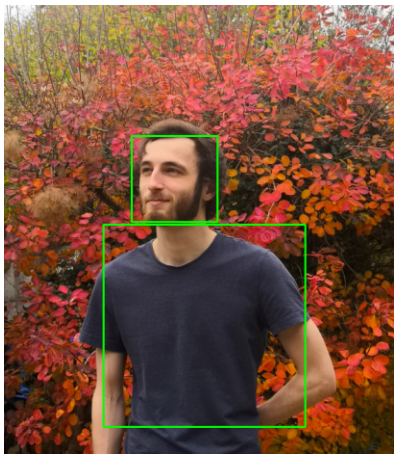


Figure 10: Face detection using the sliding window

6 Project structure

You can find in the "lab2" folder of the project the following code:

- The "dataGeneration" jupyter notebook for the code generating the imagettes from the raw databases
- The "training" jupyter notebook for creating the model and evaluating its performances
- The "faceOrientation" python script to evaluate efficiency in respect with face orientation
- The "imageAnalysis" python script to execute a sliding window algorithm on a given image using a model previously used.

The project also contains some scripts for general purpose, mainly in the "common" folder.