Dorian PILLARD

Technical exercise

Part 1 : Roads and fields classifier

For this exercise, I developed a classifier using Keras and Tensorflow and wrote my code in a Jupyter Notebook, so I could follow the training using a Tensorboard.

I started with a simple CNN with the following default architecture I begin with when solving light problems like this one:

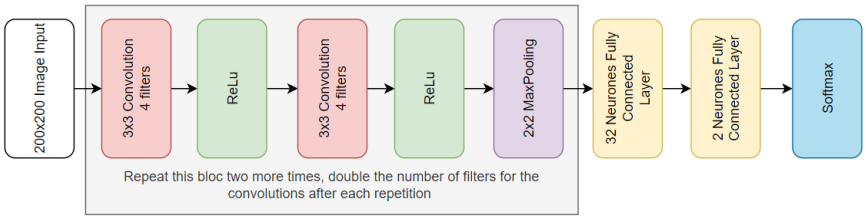

Figure 1. Network architecture

In the case of a binary classification problem a sigmoid activation function for the last FC layer could also have worked, but I encountered issues making it work with the confusion matrix function when evaluating model performance.

I started training over 40 epochs with a constant learning rate of 0.01 and did a 4-1 split of the training images into 'train' and 'validation' sets. I trained with batches of 8 images.

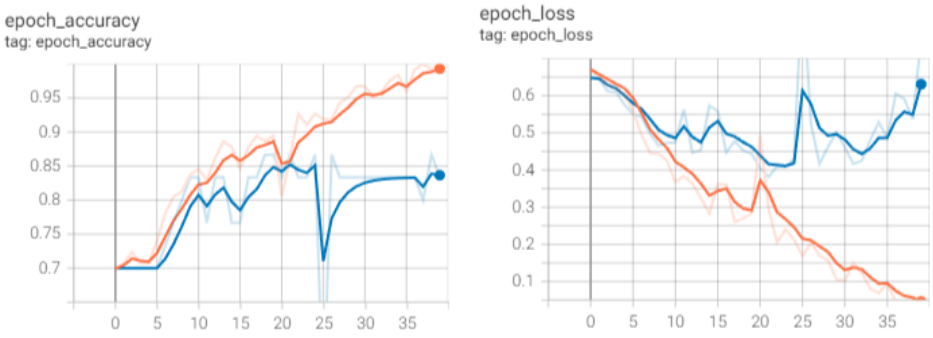

Figure 2. Accuracy and loss of the model after each epoch

We can immediately see what the problem is: the model is overfitting, because the training loss (in orange) is still going down at the end of the model fitting, but the validation loss (in blue) stays constant for a while before going back up. It is a good starting point, because it shows the network can learn features, but it should be the very next thing to correct.

The model could be overfitting for a couple of reasons:

- Few labeled data
- Model learning a certain path when trying to classify an image

To solve this, I implemented data augmentation on the training set and a dropout layer after the hidden fully condensed layer. I also added batch normalization to each convolution to accelerate the training, and increased the number of epochs to 100, seeing as the training loss was still going down after the first training:

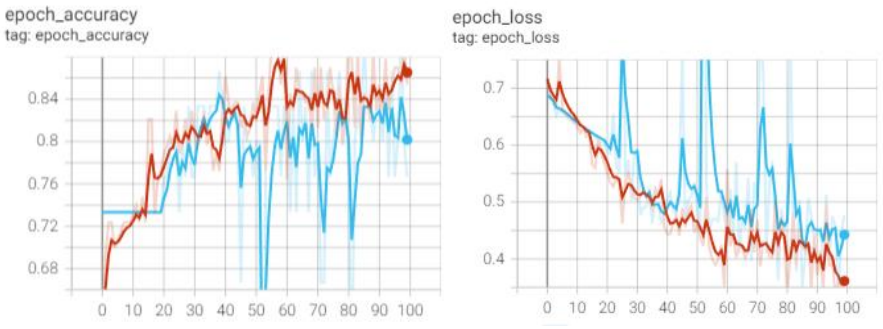

Figure 3. Accuracy and loss of the model after each epoch, with improved training options and model

The overfitting issue seems resolved, since now the validation loss follows the training loss, except for some occasional spikes. However, the loss is still going down, so I did one last fitting over 300 epochs to make sure the loss converges when the fitting is over:

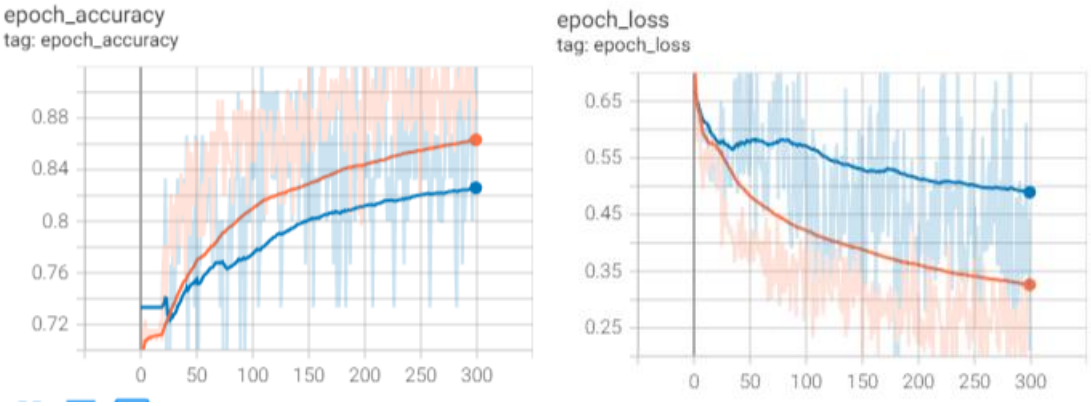

Figure 4. Final model accuracy and loss after each epoch, smoothed

The loss is still going down for the training (orange) set, but it reached a plateau for the validation (blue) set so there is no need to prolong the fitting; otherwise, there might be some overfitting again.

The last step is to test the classifier on the test set, to have an unbiased evaluation of the model. I used the confusion matrix function from the sklearn library. I also split the test data into two separate repositories ('fields' and 'roads') according to the class of the image so it would work with the function:

```
Confusion Matrix
[[3 1]
 [0 6]]
```

Figure 5. Confusion matrix

The model reaches 90% accuracy on the test set, with one field being classified as a road. This probably comes from the fact that there are more roads than fields in the training dataset, so there is a little bit of bias when the model is uncertain of how to classify the image.

I expect that the model will do similarly well on a different test set, considering I used data augmentation and applied a range of transformations to the training set. This should have allowed the model to learn more difficult cases than with the base set, which should have expanded its range of effectiveness.

Using transfer learning could have yielded better results, but to show that I understand how CNNs function and how to trouble shoot issues when training such models, I chose to train a model from scratch.