The BeeImage Dataset

Semester project for NIE-MVI course

Davide Ariotto

17/12/22

Introduction

Convolutional Neural Networks (CNN) have been successfully used to classify images. I will focus on these networks and try to explore their capabilities.

The task is to explore this Kaggle dataset: https://www.kaggle.com/datasets/jenny18/honey-bee-annotated-images and implement a CNN to predict some of the attributes images of bees. The dataset contains more than 5'000 images annotated with location, date, time, subspecies, health condition, caste, and pollen.

The original batch of images was extracted from time-lapse videos. The single bees were then cropped out of some frames so that each image has only one bee. Because each video is accompanied by a form with information about the bees and hive, the labeling process is semi-automated.

Other than the images a it is available a csv file containing the values of each attribute for every image.

Description of my approach

First of all I imported the data using pandas, then I did some quick checks on them: I first looked for NULL values but there weren't any so I didn't have to find the best way to manage them; then I checked if every instance in the data had a corresponding image and vice versa and also here there were no problems. To do it, I intersected the data table with the images files names and then I checked the length of the newly created set. Finally I deleted all instances with 'subspecies' equal to -1 because those are incomplete data.

After this I plotted the distributions of the features to take a look at them. Firstly I noticed that there was an error in the 'location' attribute: some instances had a value equal to 'Athens, Georgia' and others had 'Athens, GA' that are obviously the same thing. I ended up not using it but I corrected it anyway. The most promising attributes for a classification task were: subspecies, health condition, pollen carrying and caste. I decided to discard the last two: over 95% of the images refer to bees not carrying pollen, so the data are too much unbalanced; for the caste it's even worse because there are no images of queen bees (all of the images are about worker bees). The data are quiet unbalanced also for the first two features but initially I didn't balance them.

Then I prepared the train, validation and test data: the format is a numpy array of images for the Xs and a one-hot encoded pandas data frame for Ys.

Lastly, I defined the model, trained and tested it. The description of the configurations and the experimental results are in the next part.

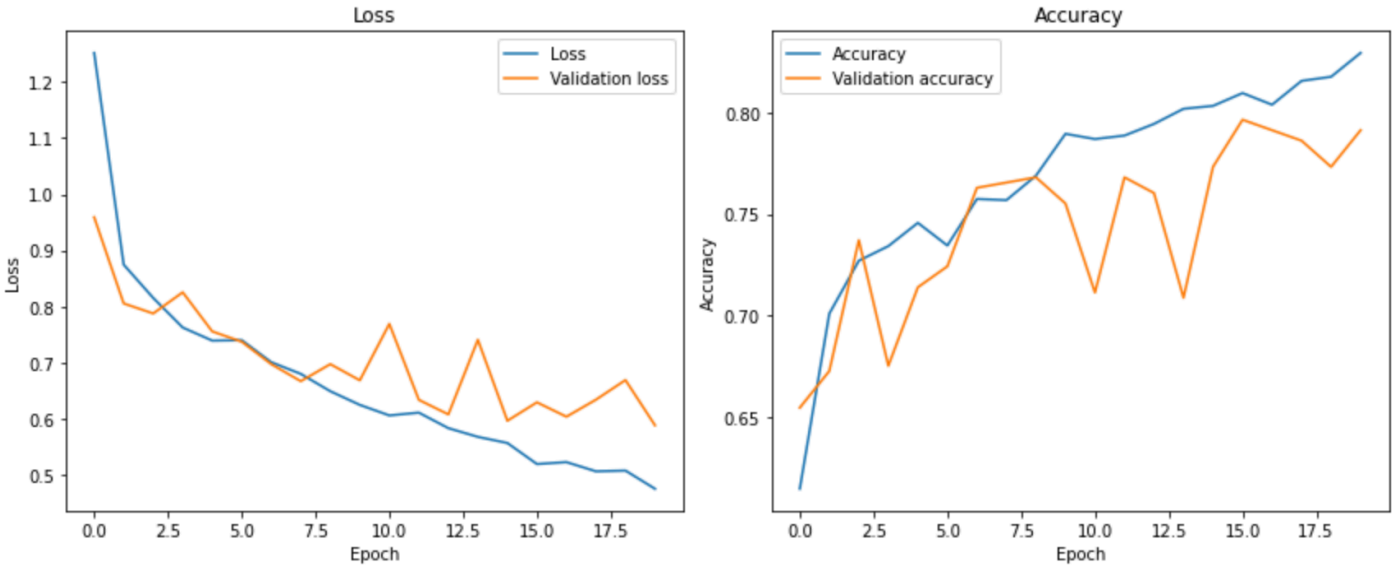
Experimental results (subspecies)

Model 1

The first model is the simplest:

1. Convolutional layer with two filters of size 3x3 (padding = 'same' means that 0s are added next to the borders of the image)
2. Max pooling layer of size 2x2
3. Flatten layer
4. Dense layer with softmax that outputs the predicted layer

Training:



Testing:

Loss = 0.58

Accuracy = 80.43%

Comment:

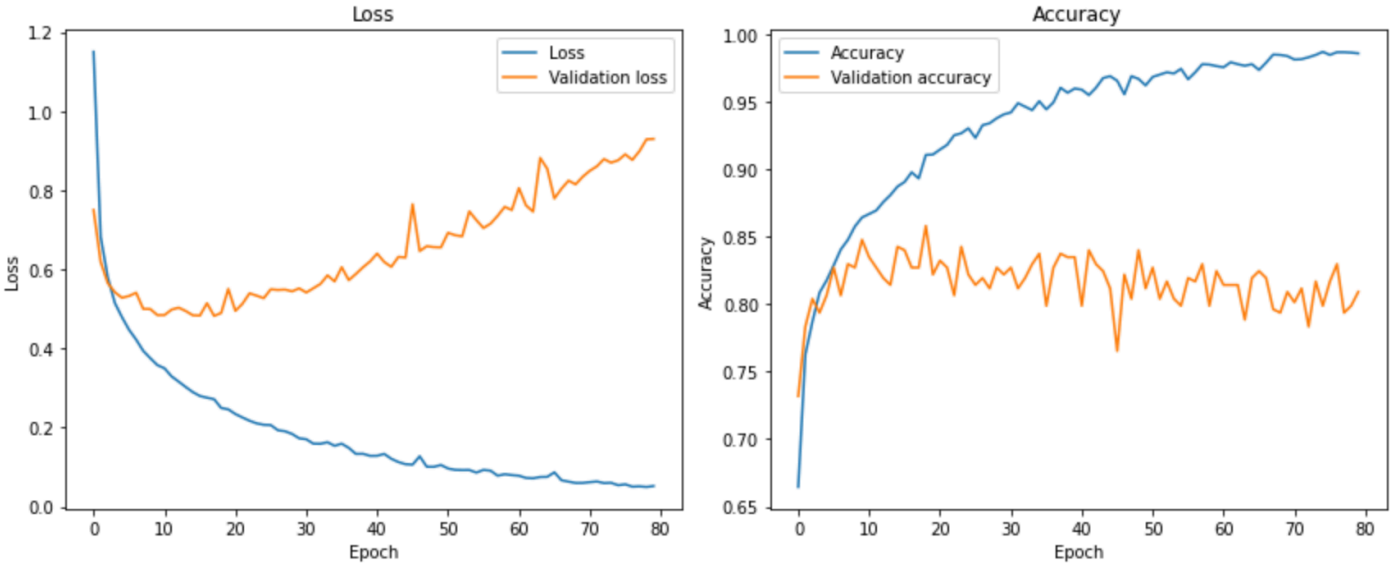
The results are quiet good give that the architecture is really simple but we can notice two things: both curves aren't flat at the end meaning that there is still space for improvement and they have several spikes so again the model is not learning enough (underfitting) leading to poor results on the validation sets.

Firstly I'll try to improve it by just increasing the number of epochs and then I'll try with a more complicated architecture.

Model 2

Before the number of epochs was 20, now I set it to 80.

Training:



Testing:

Loss = 0.89

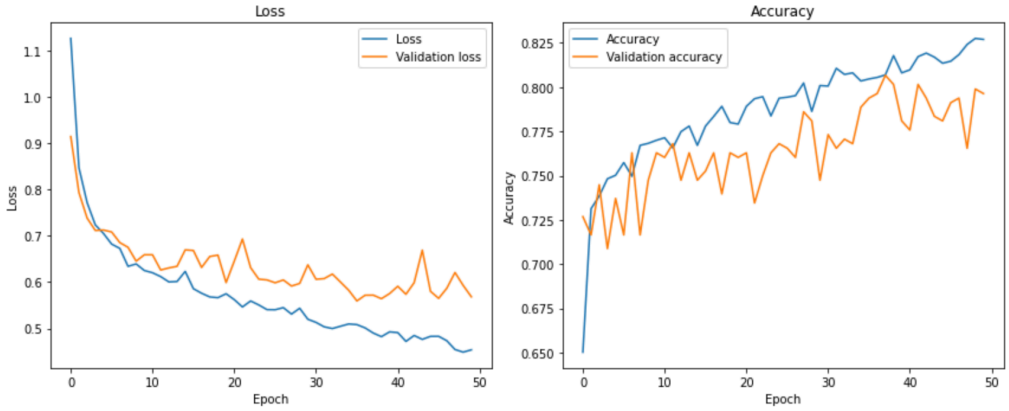
Accuracy = 81.13 %

Comment:

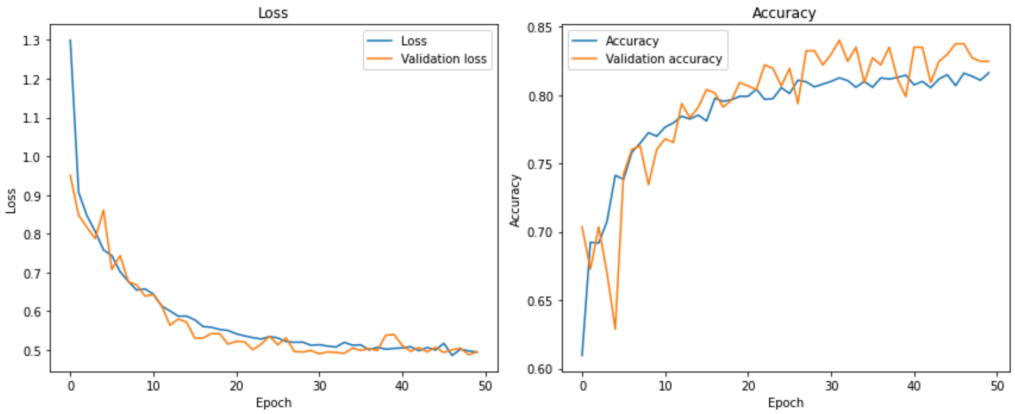
Here we have the opposite problem: the curves start to diverge almost immediately and this means that the model is overfitting. In addition to this, 80 epochs are a lot, so I decided to lower this value to 50 and build a more complicated architecture.

Model 3

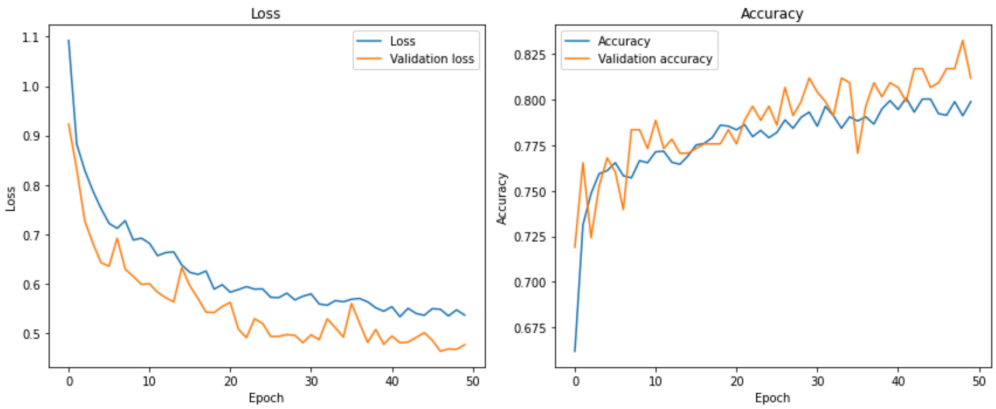
In order to tackle the overfitting I applied some data augmentation. First I added a layer that perform a random flip of the image before the convolutional layer:



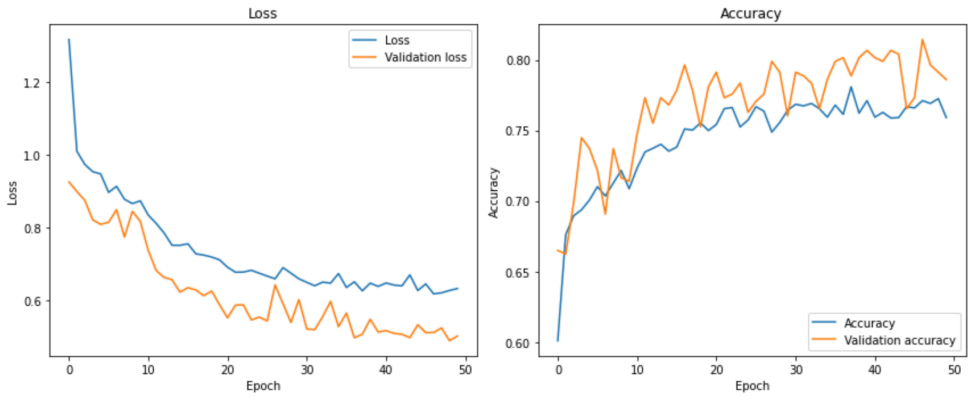
Then I also added a layer that perform a random rotation of the image:



Then a layer that perform a random change in the image contrast:



And finally a layer that performs a random transaction of the image:



These are the results of testing them:

	Loss	Accuracy (%)
1	0.52	79.42
2	0.59	78.58
3	0.56	77.80
4	0.54	79.35

Comment:

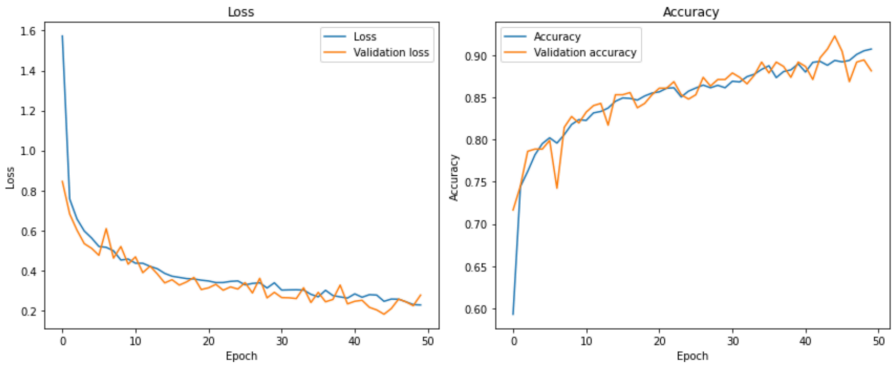
Focusing more on the graphs than on the testing results I decided to choose the second configuration, the one with a 'RandomFlip' and a 'RandomRotation' layer, where we can see that there's almost no divergence in the loss curves.

Then I tried to increase the number of filters in convolutional layer.

Model 4

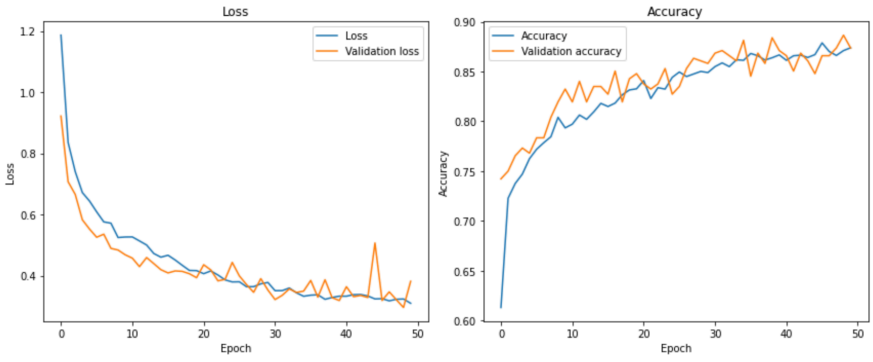
I increased the number kernels and the number of convolutional layers in order to try to extract more features.

First I set just one convolutional layer with 8 filters and these are the training results:



Test loss = 0.37 and accuracy = 86.00%

Then I increased the number of layers: one convolutional layer with 3 filters, one max-pool layer, one convolutional layer with 9 filters and another max-pool. These are the training results:



Test loss = 0.44 and accuracy = 83.99%

Comment:

We can see that the model can learn more because the curves aren't flattening, so the next step will be increasing even more the complexity and the number of epochs but adding an 'EarlyStopping' to speed up the computations.

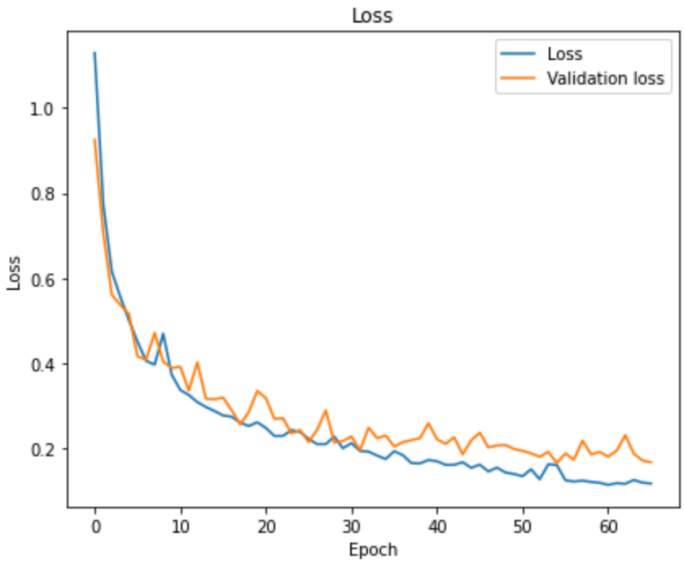
Model 5

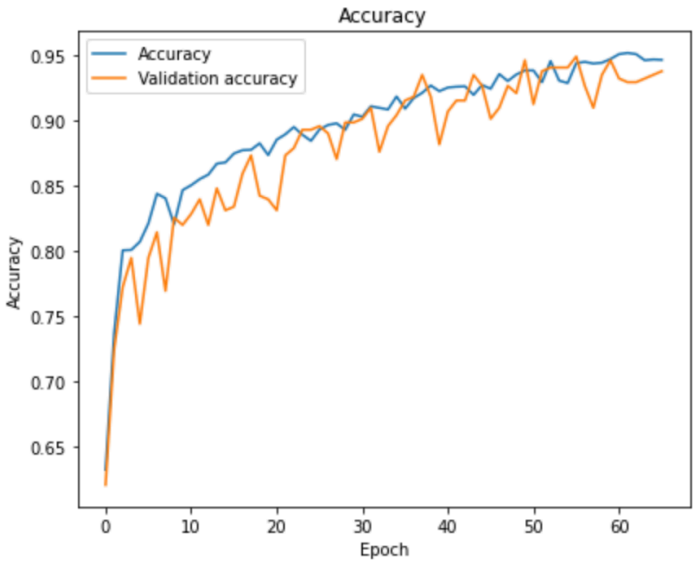
At this point I tried adding more convolutional and pooling layers first and then more dense layers, I increased the number of epochs to 80 but I also added an EarlyStopping callback that interrupts the training if there isn't an improvement of 0,001 in the loss for 5 epochs. I also tried adding a Dropout layer that at each epochs randomly 'disable' some neurons in order to promote strong and general patterns instead of weak and very specific ones, but results got worse. I also tried to change the max-pooling layers with average-pooling ones but the accuracy was comparable.

The chosen and final architecture is:

- Convolutional layer (3 kernels)
- Max pooling
- Convolutional layer (9 kernels)
- Max pooling
- Convolutional layer (27 kernels)
- Max pooling
- Flatten layer
- Dense layer (56 outputs and 'relu' activation function)
- Dense layer (14 outputs and 'relu')
- Dense layer (7 outputs, equal to the number of labels, softmax activation function)

Training:





Testing:

Loss = 0.13

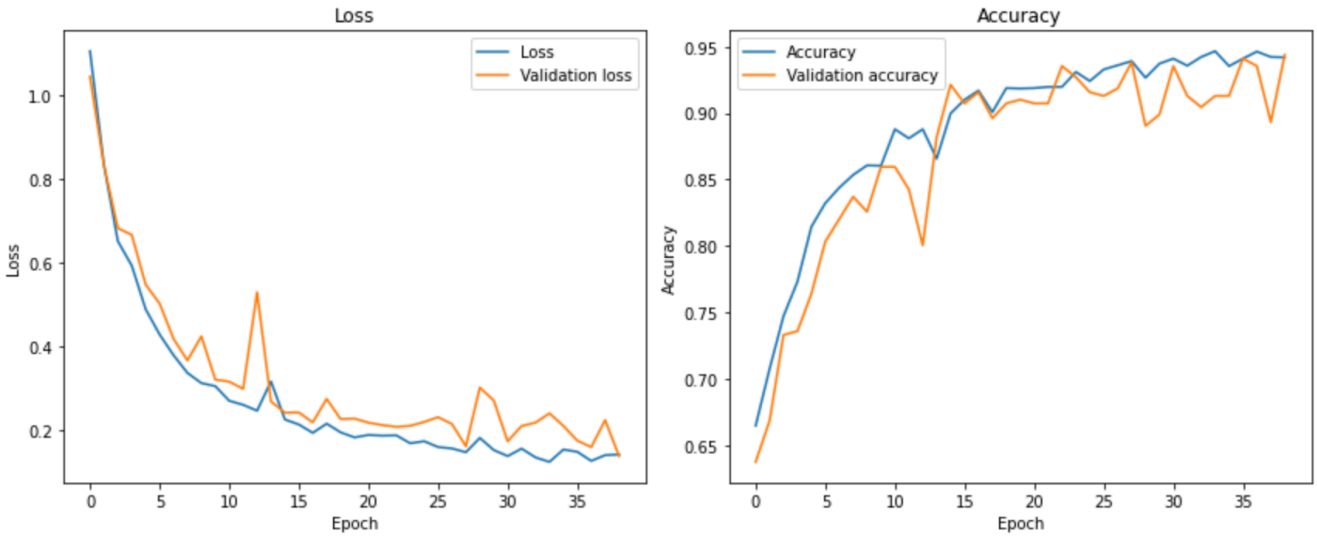
Accuracy = 94.86%

Experimental results (health condition)

Model 1

I decided to start using the final model of the subspecies classification and see if it performs well and if there's space for improvements.

Training:



Testing:

Loss = 0.19

Accuracy = 91.32%

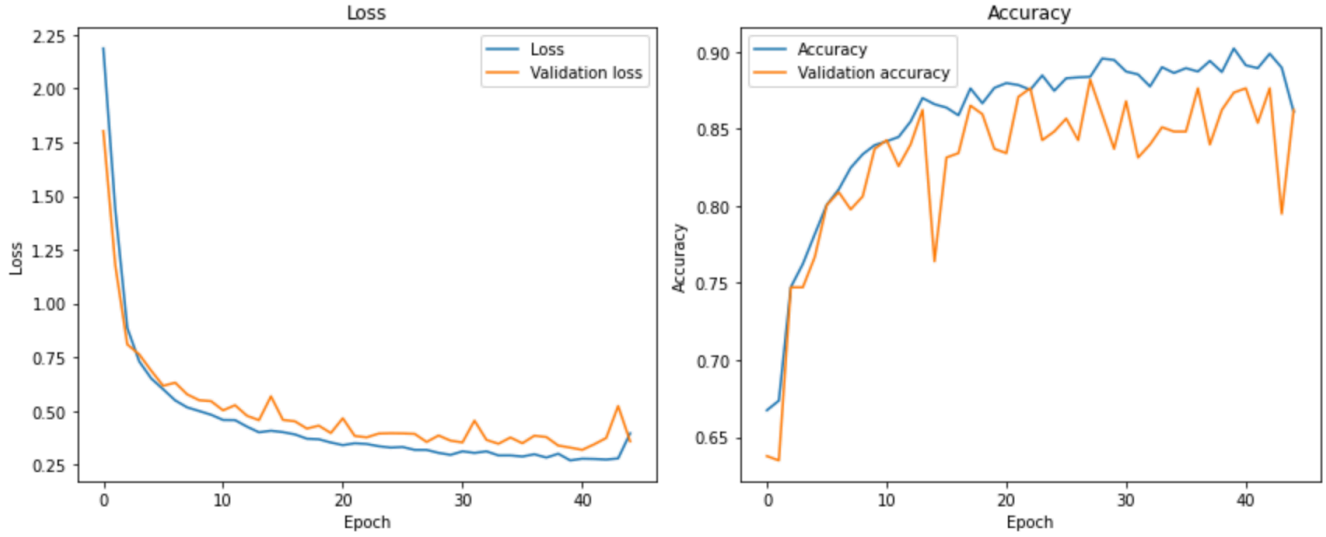
Comments:

Results are quiet good but we can see that there's still a bit of oscillations in the graphs, especially in the validation accuracy.

Model 2

My goal now is to tackle this oscillations, signs that my model is a bit overfitting. For this purpose I added some regularization to the dense layers. After trying various settings, this is the one that gave me the best results. I added an l2 regularizer to the first dense layer with a regularization factor of 0.001 and another to the second dense layer (this time a combined l1 and l2 regularizer) with factor equal to 0.01. Here are the results.

Training:



Testing:

Loss = 0.28

Accuracy = 89.88%

Comments:

The oscillations seem to have decreased a bit but also the results quality had so I will keep as my final model the previous one.