**Computational intelligence assignment 2: Object recognition - CIFAR dataset**

**Abstract:**

Deep learning is a subset of machine learning, it is based on learning and improving by itself by analysing computer algorithms. It has brought great impact to artificial intelligence, especially in the fields of image processing, pattern, and object recognition in recent years. Object Recognition has become one of the most famous topics of computer vision. Object Recognition became one of the most famous topics of computer vision. After the success of the technology of convolutional neural network in 2012 of the framework ImageNet, many researchers start to contribute and experiment different type of techniques and technologies combined with the CNN (like Dropout, Data Augmentation, etc.), and so far they’ve provided a lot of datasets that has helped the community build several frameworks, like Inception, Mobilenet, etc.

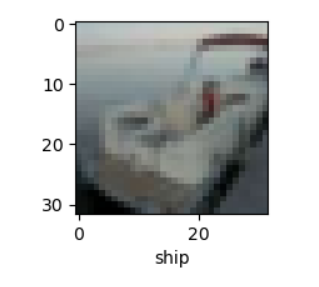
Since deep learning requires a large amount of data in order to return accurate results, data is fed as huge data sets, and when processing it, they are able to classify it with the answers received by a series of binary true or false questions that involve a lot of complex mathematical equations. (Reyes, December 8, 2022)

For this project, I will work with the public dataset CIFAR-10, build 2 neural networks, train them and evaluate them with the testing data and validation data respectively.

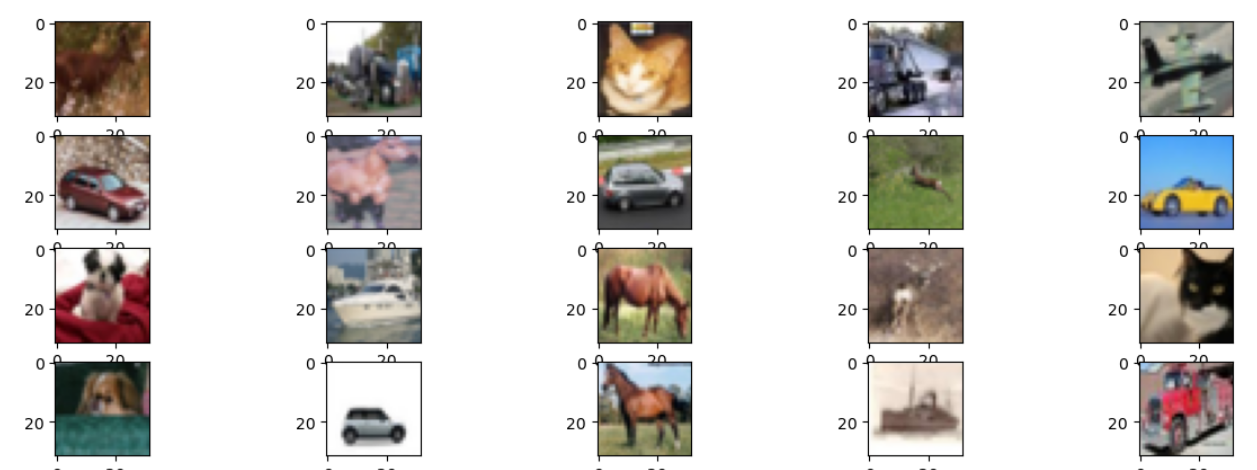
**Data normalization**

“CIFAR-10  is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.”

I created a method to get any image based on its index within the data to confirm it works:



Then, I used the matplotlib library to get 20 random images from the dataset:



In order to normalize the images, I divided all my data by 255 each as that is the biggest amount of pixels that the images went to, in order to have values between 0 and 1 instead of 0 and 255.

**Data preparation and importance of validation data**

To prepare my data for training, I had to split them, usually I have only tried it with training and test data, but now I had to learn about the importance of the validation data as well. I have learnt that it is rather important in the process of model evaluation, as it evaluates it on random invisible data, and it represents the indicator of judging the model if it has sufficient capacity to be more generalized on other invisible data.

So I built the validation data based on the training data, and made it have the same amount of images as the test set. Because the dataset had 60000 images, they have now ended up being split among the training, test and validation, by 40000, 10000, 10000 respectively.

**Model preparation and training**

For my model, I chose the convolutional neural network with 32 filters, and a kernel size of 3 by 3, with a relative rectilinear unit (relu), and I initialized the weights with he\_uniform, which “Draws samples from a uniform distribution within [-limit, limit], where limit = sqrt(6 / fan\_in) (fan\_in is the number of input units in the weight tensor).”

Then I added the max pooling layer, in order to reduce the number of pixels in the output of the previous layer, which will in turn reduce the dimensionality of the image, leading to faster training.

I had a regularization dropout layer, with 20% activation phase, then 2 more CNN layers with 64 filters instead of 32, with the other parameters remaining the same, and another max pooling layer 2 by 2.

Now, I added another regularization dropout layer with 30% activation phase, 2 CNN layers with 128 filters, another max pooling layer and yet another regularization dropout layer with 40% activation phase.

At this point I had added a flatten layer to flatten the output and pass it to the following dense layer, with 128 units, relu activation, and he\_uniform weights initialization, after which I add the final regularization dropout layer with a 50% activation phase.

Finally, I add the final dense layer with 10 units, corresponding to the 10 classes of the dataset, with the softmax activation function which I used to later predict the model.

Unlike the binary classification, in which there are only 2 possible classes, in the multiclass classification, as the name implies, there are more than 2 possible classes. Since the purpose of this project is to classify several thousands of images, I could not use the sigmoid function, as that would take any number and give everything a number in a range of 0,1. I also could not use the argmax function, as that returns the index of the maximum value in the input array. It makes it simple to infer the predicted class from the argmax output, as it would focus on the index, if we want to know how likely an image is to be that of the object, softmax is a lot more suitable.

The softmax function takes in a vector of raw outputs and returns vectors of probability scores, it makes it much easier to interpret the output of the model.

Regardless of whether the input values are positive negative or zero, it will transform them all into values between 0 and 1, so that they could be interpreted as probabilities.

In this case, in my last dense layer, it will predict a multinomial probability distribution, so the class prediction would be the class that has the higher probability.

My second model had an additional batch normalization after each CNN layer in order to standardize the inputs to a layer for each batch and provide additional regularization with the dropout layer, to reduce the overall error.

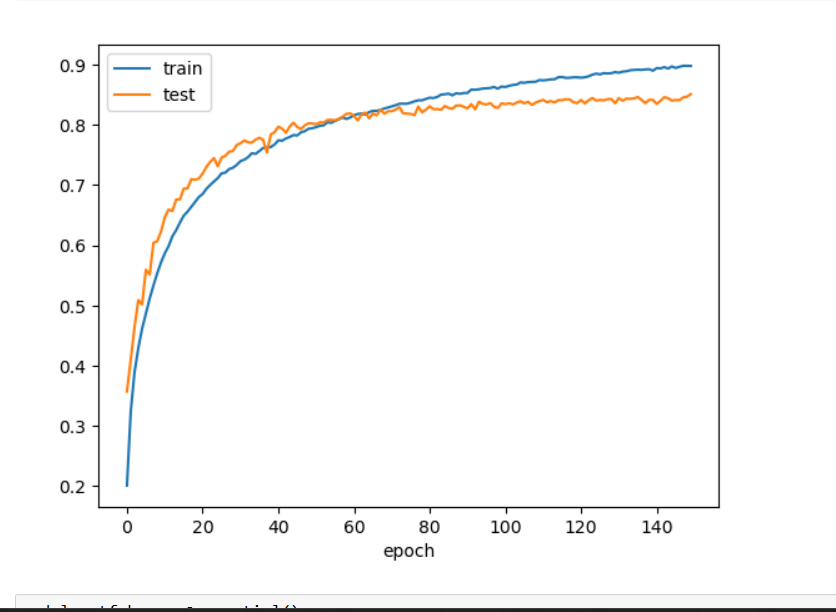
For the training, I had 150 epochs for the first model and 50 for the second, because of the batch normalization added in the second one, despite the having a third of the epochs that the first had, it still ended up having high accuracy, although less than the first.

I had a batch size of 32, and an early stopping callback to stop after 5 epochs if there are no improvements.

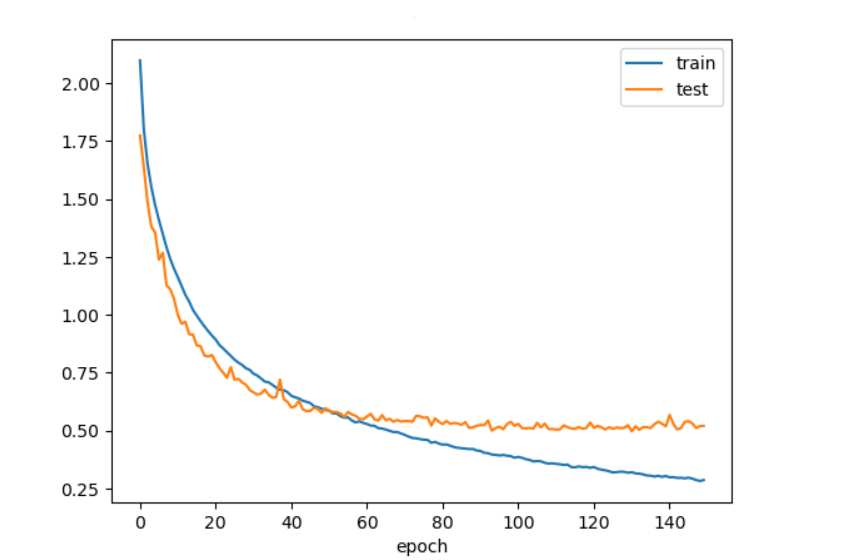
I compiled the model with the sparse categorical crossentropy function, because the labels were integers, and they are mutually exclusive, therefore not in one-hot encoding format, which made this loss function suitable for my models.

**First model:**

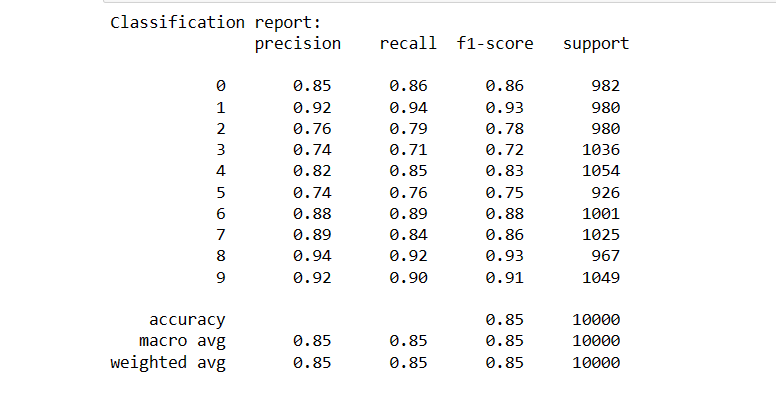
Accuracy:



Loss:

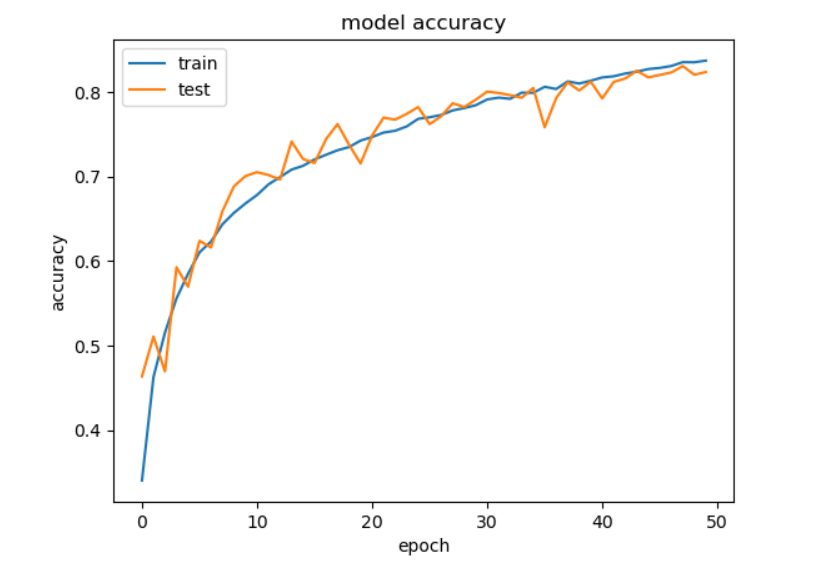


Evaluation of validation data:

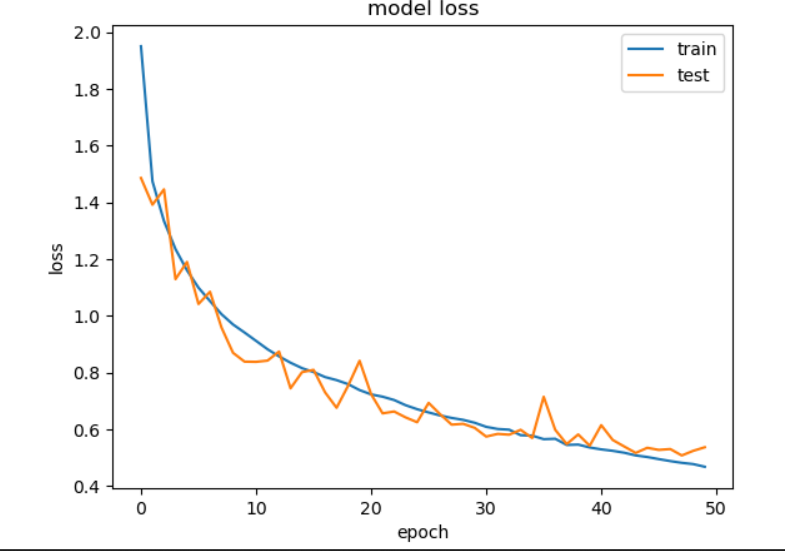


**Second model:**

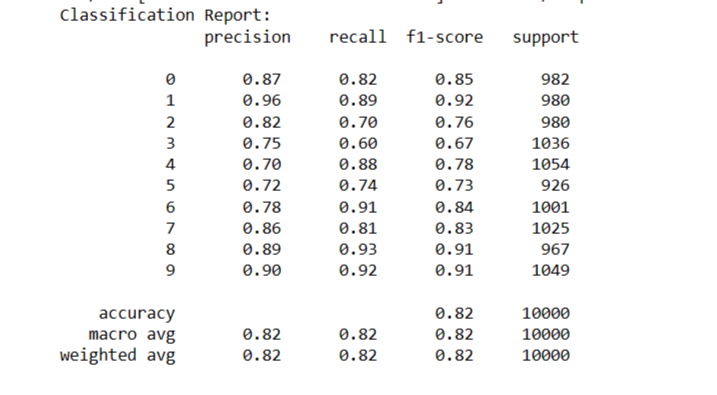
Accuracy:



Loss:



Evaluation of validation data:



**Conclusion**

Based on the results, since I had a different number of epochs. We can see that for the 50th epoch of the first model we had an accuracy of 80%, while the accuracy for the second model is 82%, with this we can conclude that the batch normalization layer improved our accuracy by roughly 2%. The overall accuracy for the first and second model were 85% and 82% respectively.

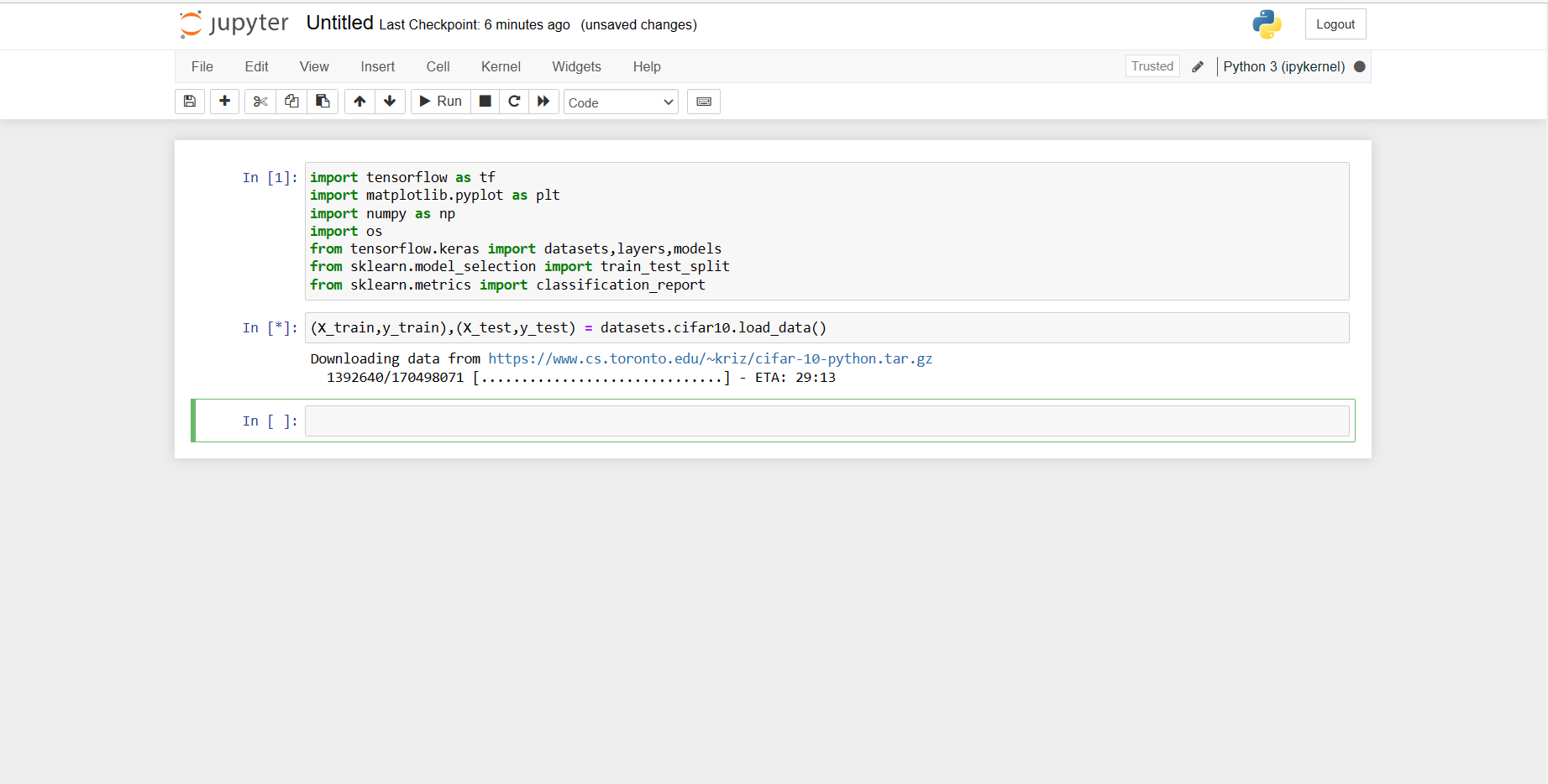
Overall, I learnt a lot about machine and deep learning while making this assignment, I have learnt about image recognition and processing, most of all I have learnt about the keras and tensorflow frameworks, and neural networks in general. I could have probably made the models better by experimenting with different layers and hyper parameters, although the long data training time discouraged me, learning how to reduce the time required at the start would have likely made me more motivated to do so. Fortunately, this made me rather interested in data science, and I would definitely like to learn much more about it, perhaps even pursue a career in it.

**References**

1. **Reyes, December 8, 2022 – What is deep learning and how does it work? -** [**https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-deep-learning**](https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-deep-learning)
2. **CIFAR10 citation -** [**https://www.kaggle.com/c/cifar-10**](https://www.kaggle.com/c/cifar-10)
3. **Weight initializer HeUniform – November 18, 2022 - https://www.tensorflow.org/api\_docs/python/tf/keras/initializers/HeUniform**

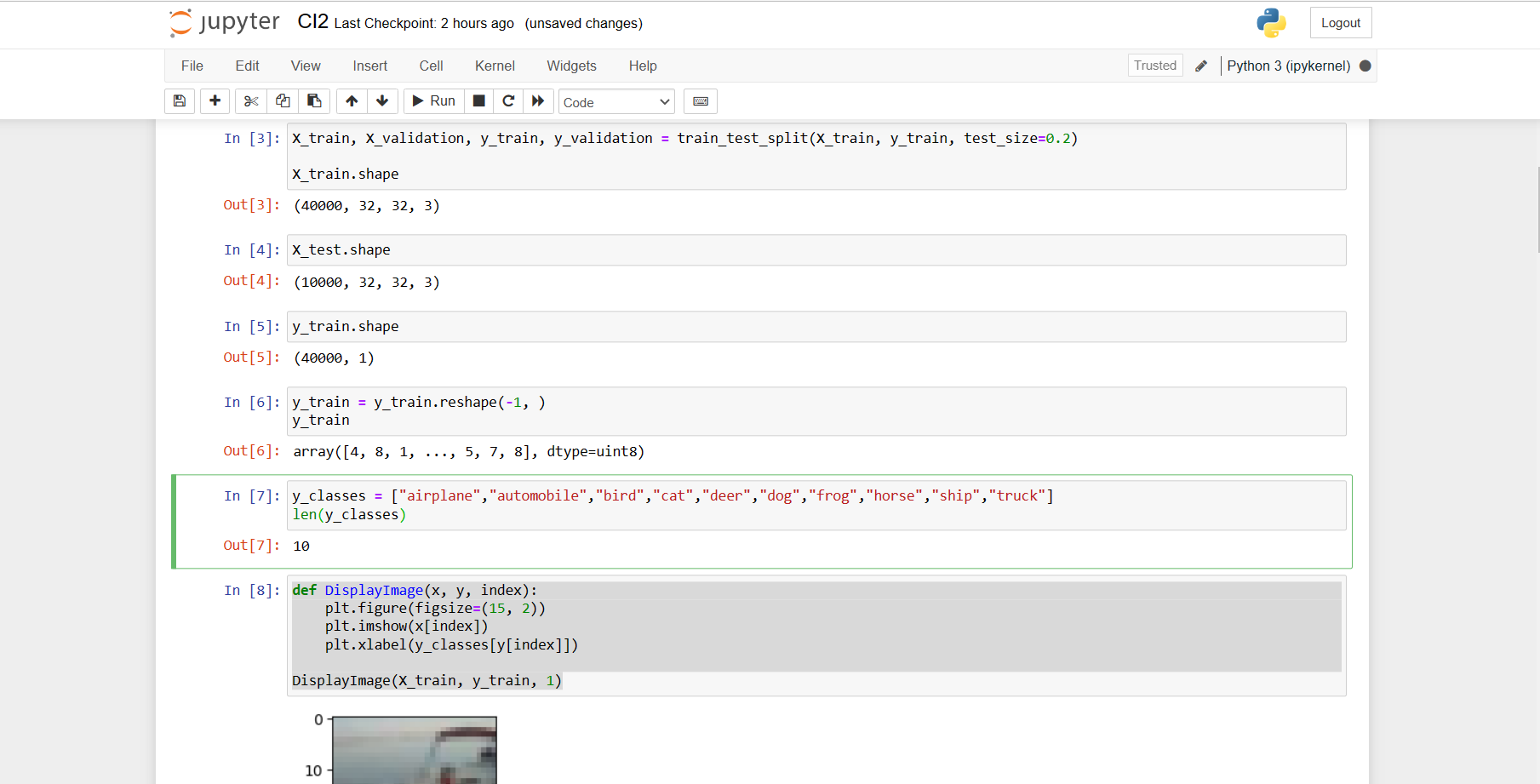
**Step by step process:**

After opening notebook, I began by importing the necessary libraries and loading the CIFAR dataset.

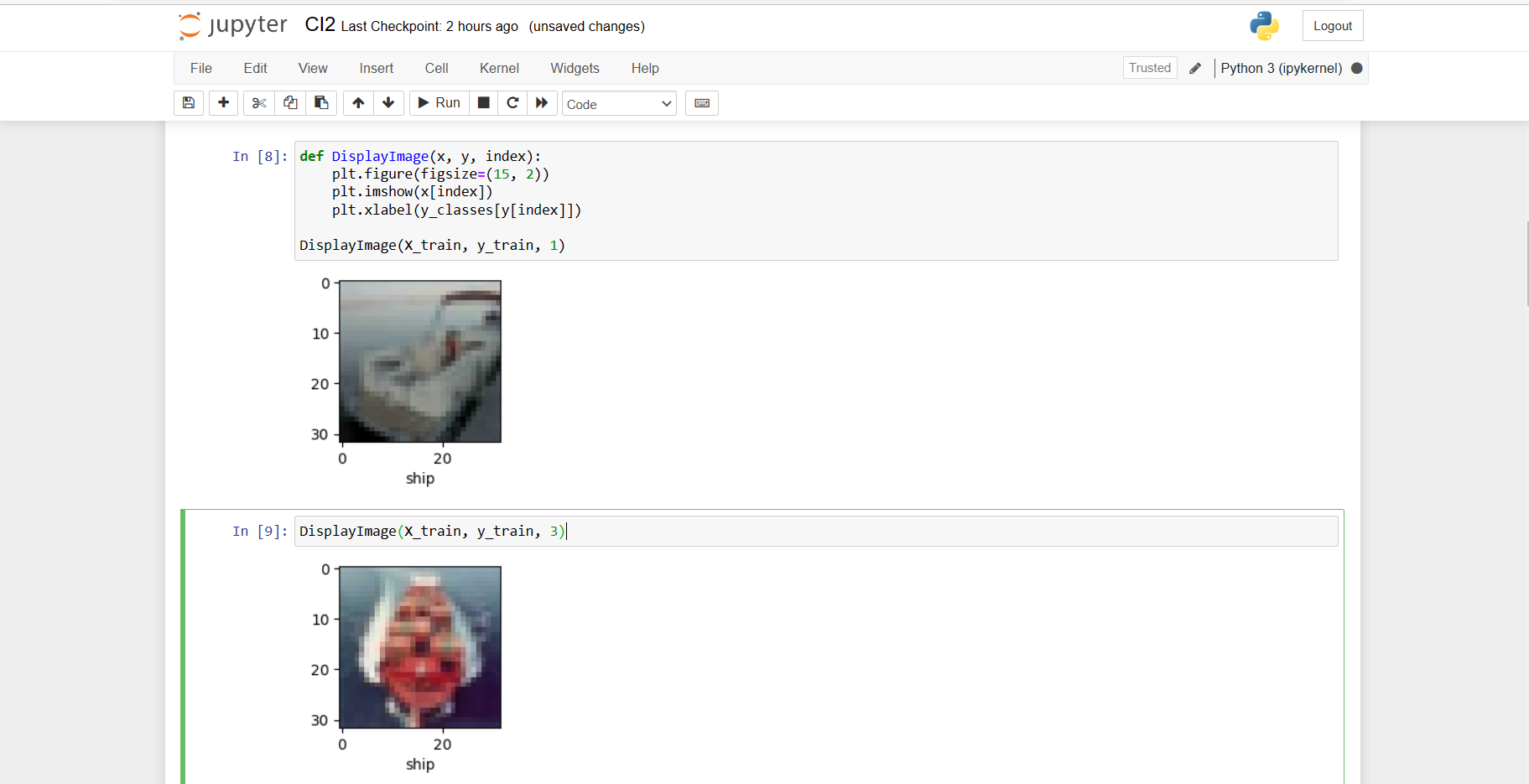


Now, I define the train, test and validation data, and split it by 0.2.

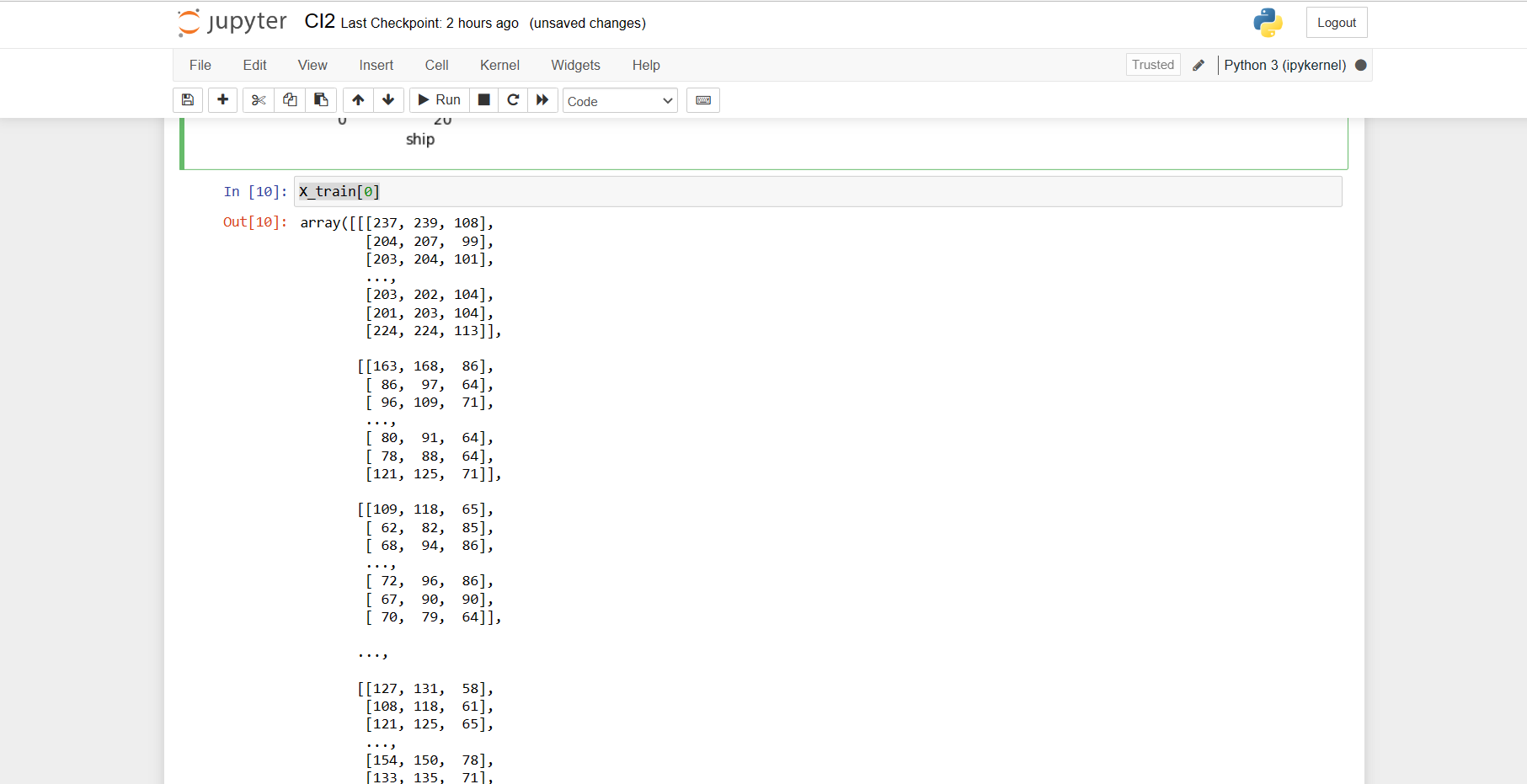
After that, I use X\_train.shape to make sure that the shape matches with the dataset description since there are 60000 training images and they are 32x32 pixels, and 3 color channels. I do the same for the X\_test. Because I had split the data prior to this, I am getting 40000 training images instead, so it is good so far.



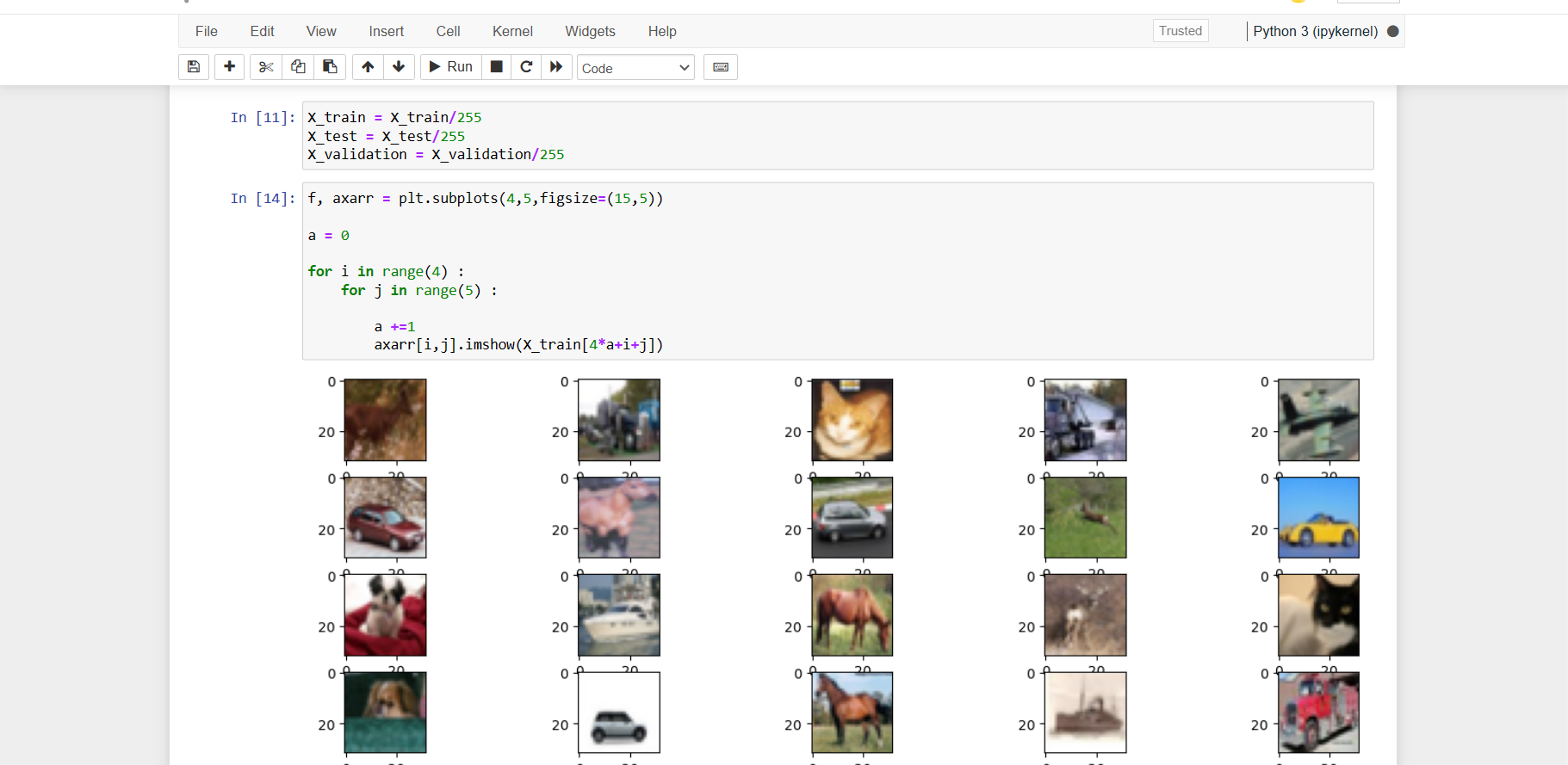
After this, I define the classes listed in the dataset in their order, I reshaped the y\_train to a 1 dimensional array to be able to run my DisplayImage function, as it had caused errors without it, and I run the function twice with different indexes to confirm.



The X\_train[0] is simply to see how the picture looks like, and since it is an image, it works with pixel values, which go from 0 to 255.



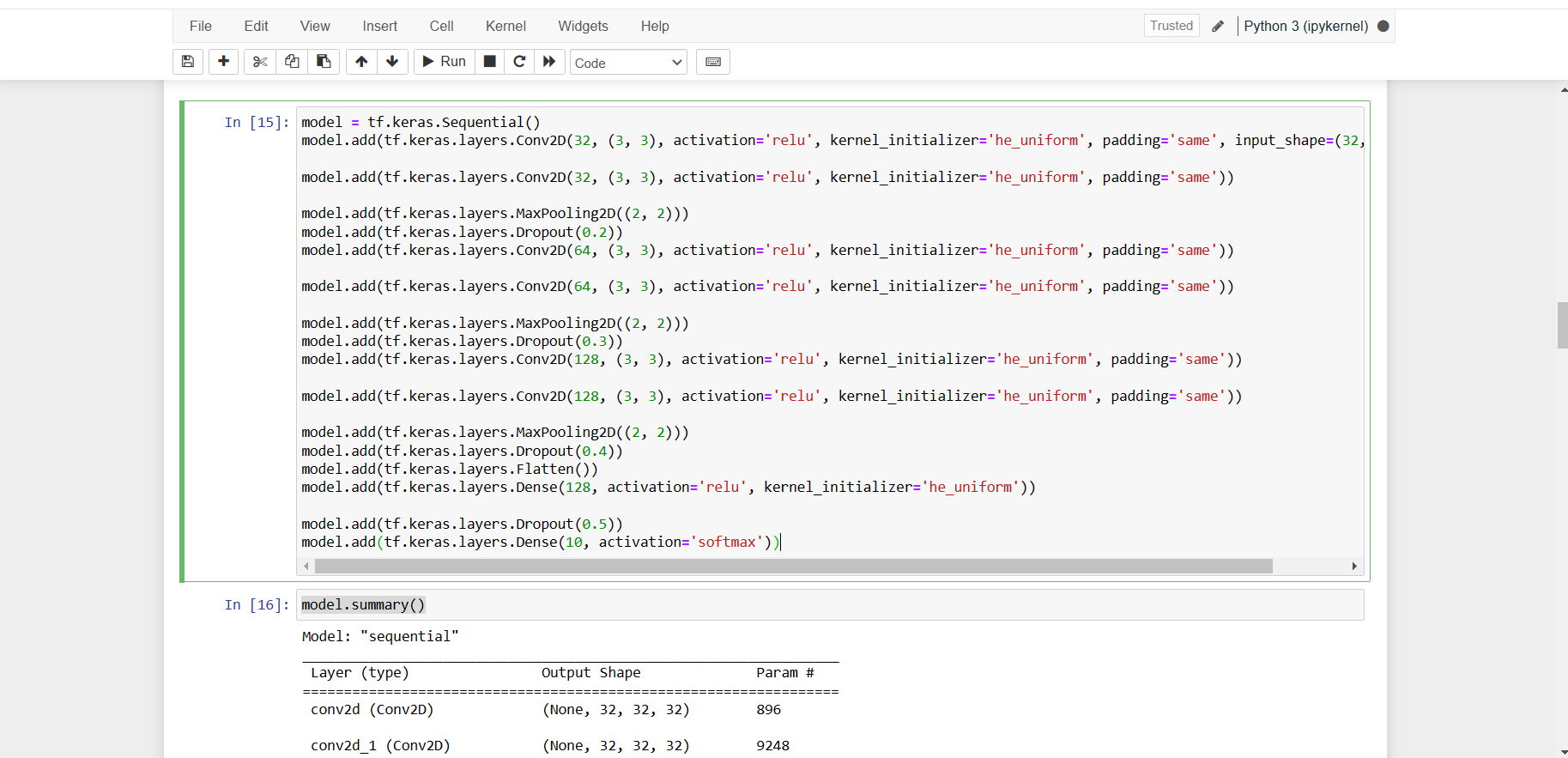
Now, to normalize the data, I divide it by the number of pixels, which is 255 here, to make it easier for the model to learn the data. I am also using plt.subplot 4,5 to show 4 rows and 5 columns of the pictures.



For the model, I made it of sequential type, the convolutional resonance’s filter is 32 and the kernel size 3 by 3. The activation function is a relu, relative rectilinear unit, with the he-uniform initializer, and the input shape is 32x32 like the images.

The max pooling simply goes through each image, and for each 2 by 2 size it will pick the maximum value, it is used to reduce computational load and overfitting.

I use flatten to flatten the output in order to pass it to the dense layer, then I add the dense layer with 128 units, and another with 10 as it is the amount of classes there are in CIFAR, and I give it the softmax activation function.

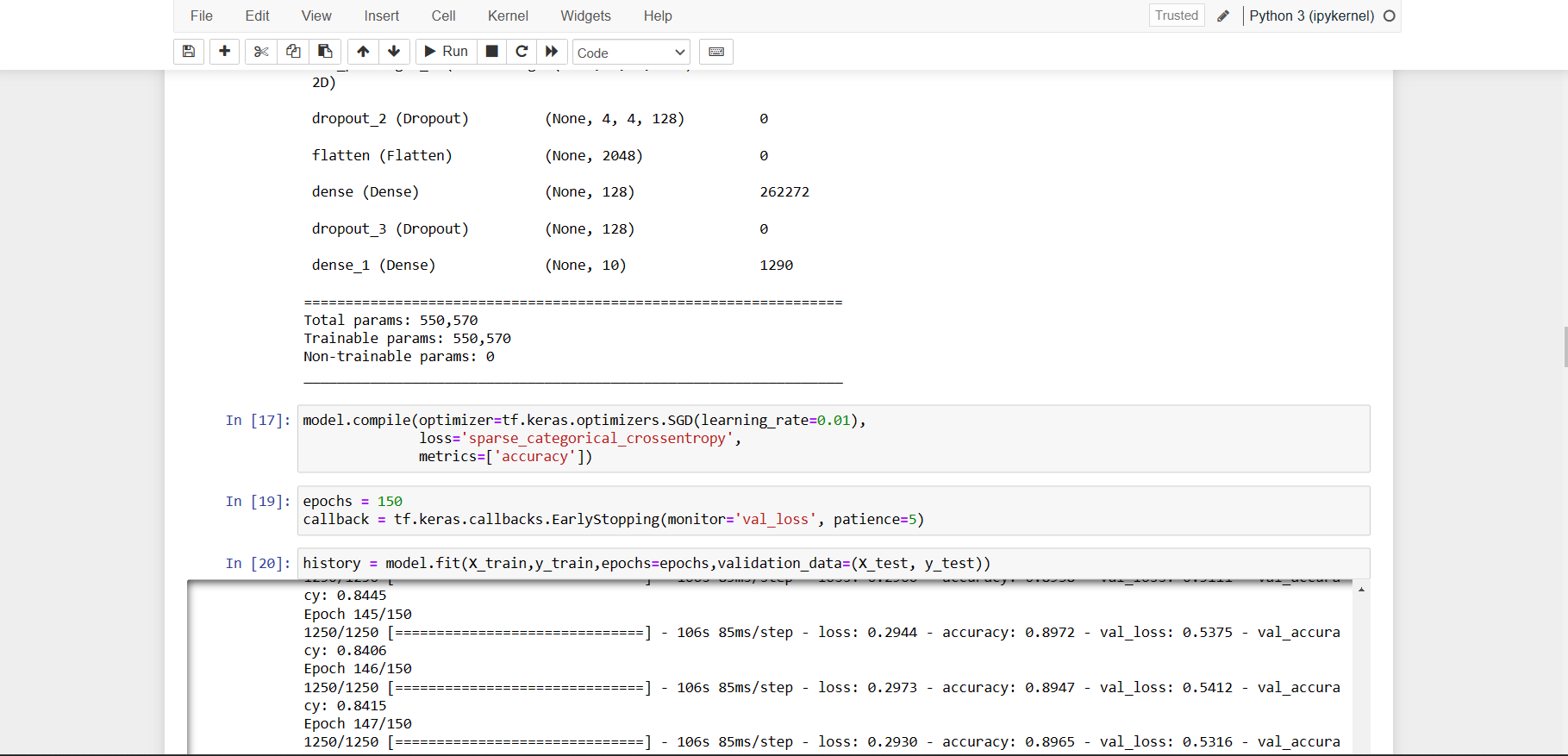


I compile the model and use the stochastic gradient descent as the optimizer, with a learning rate of 0.01, and I use the sparse categorical crossentropy loss function, since I want to measure the accuracy, that is what I pass in the metrics.

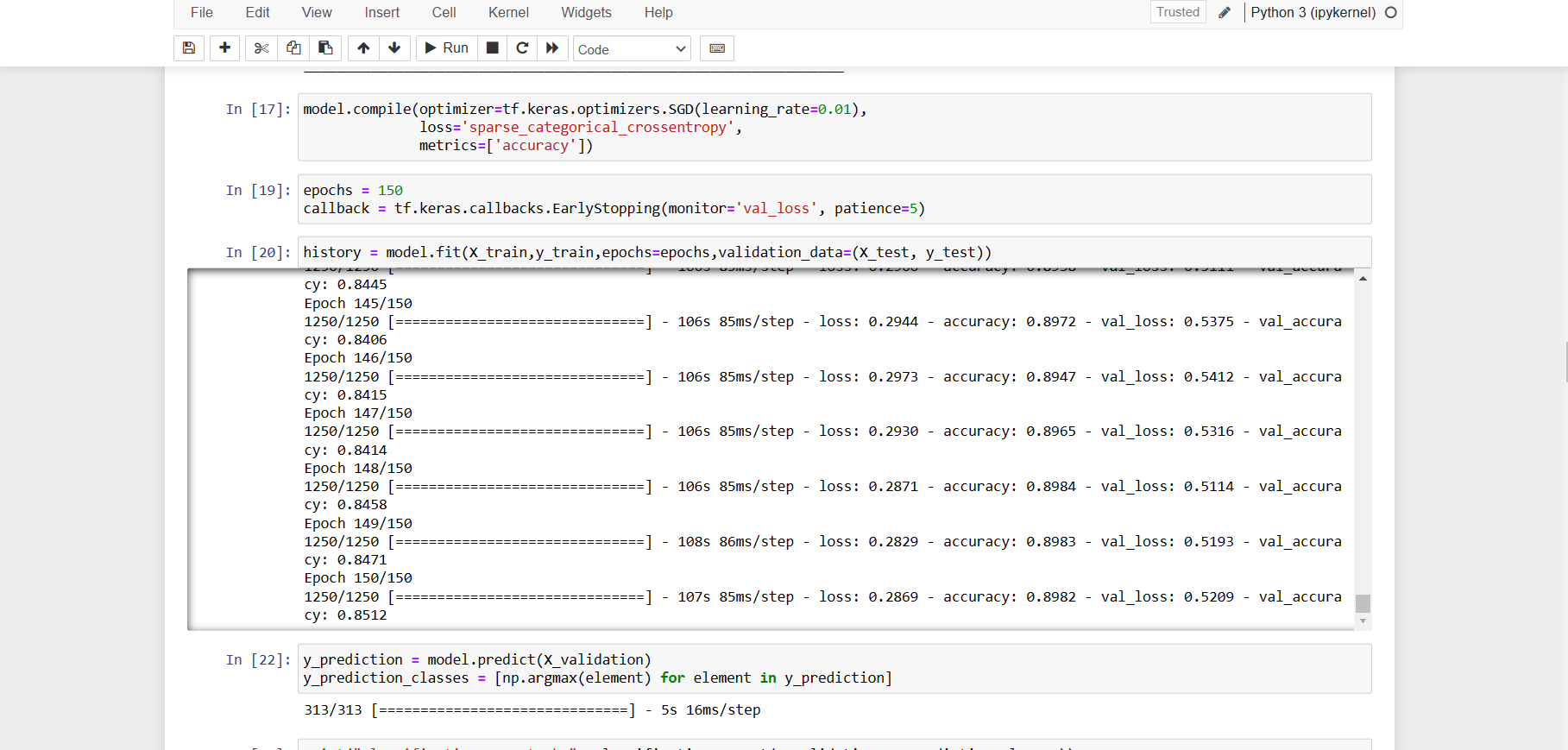
I made the epoch value to 150, which in hindsight was too much, as it took a couple of hours for it to be done processing and it ended up overfitting the model.

I use the early stopping callback in order to have the training stop if the loss has stopped decreasing, with patience=5 meaning if within 5 epochs the loss has not stopped decreasing, the training will stop.

Finally, it was time to train the model, so I let the validation data be a tuple of x\_test and y\_test.

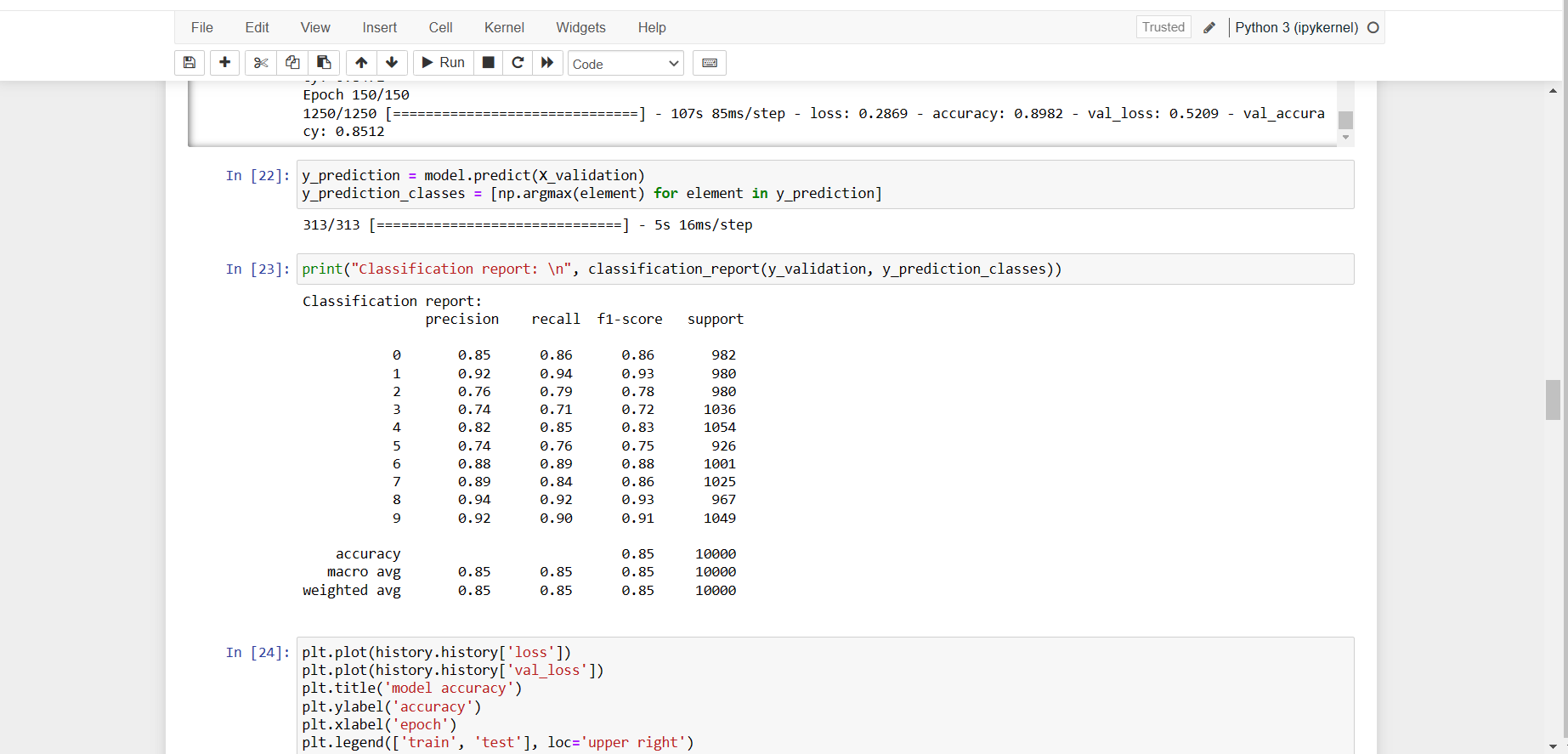


As we can see below, because of the rather excessive amount of epochs, it ended up reaching an accuracy of 85%.

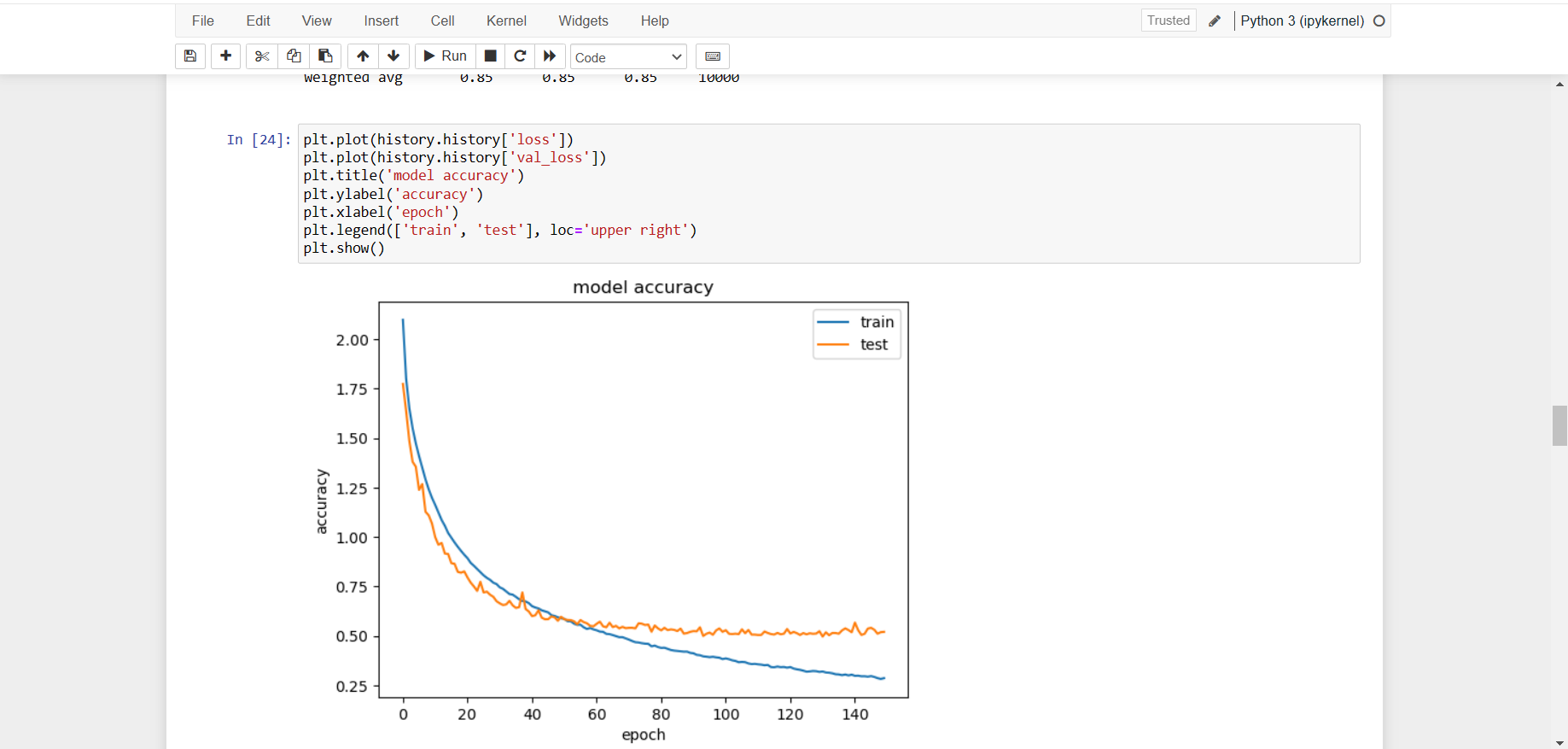


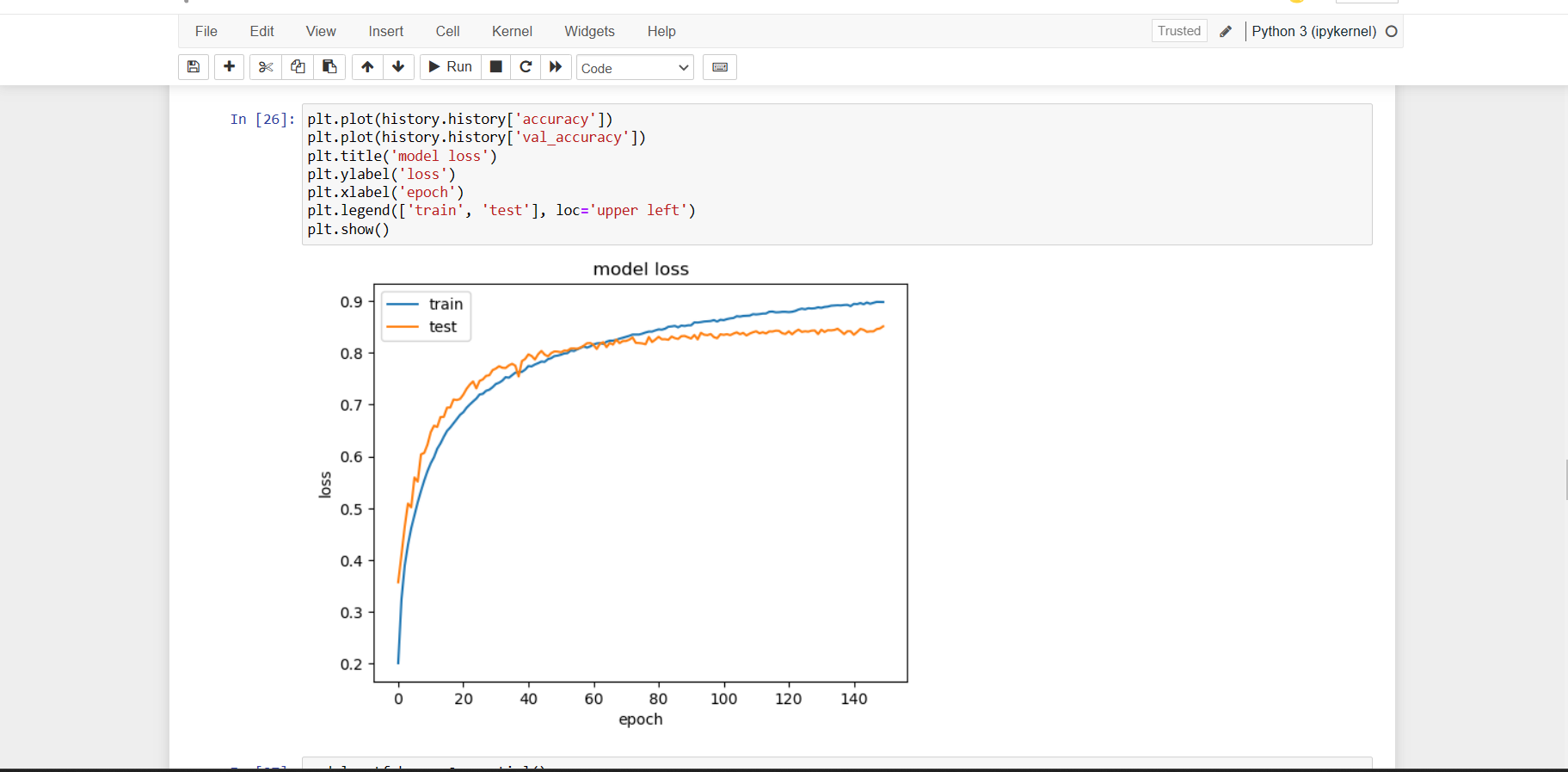
Now, it was time to check with predictions whether the model has learnt anything, so I use numpy, for each of the values that I have to make an array/list and return the maximum value of that specific list.

Then I made the classification report, and based on it we can see of the 10 classes, the precision (prediction) value is lower for the 3rd, 4th and 6th classes which are bird, cat and dog respectively.

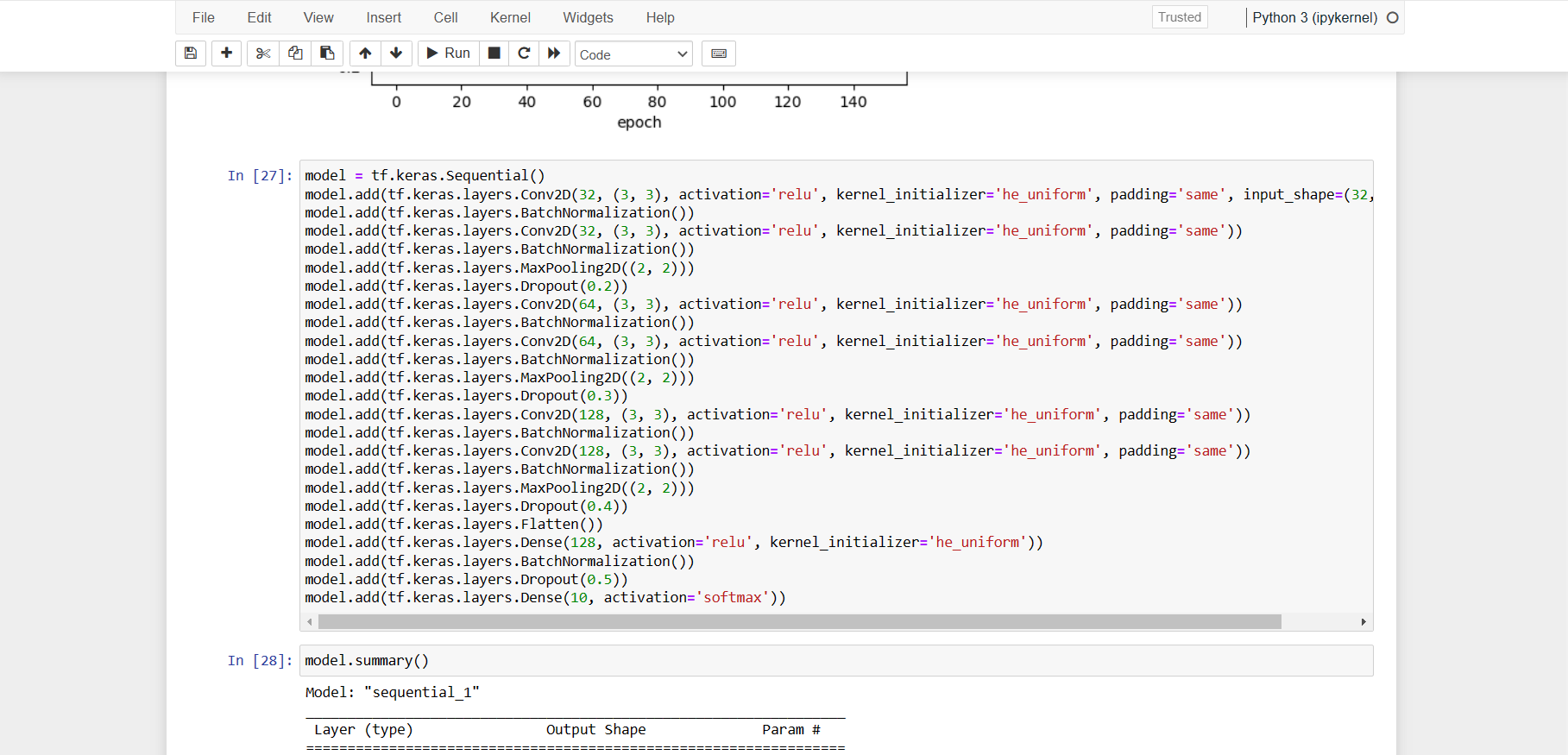


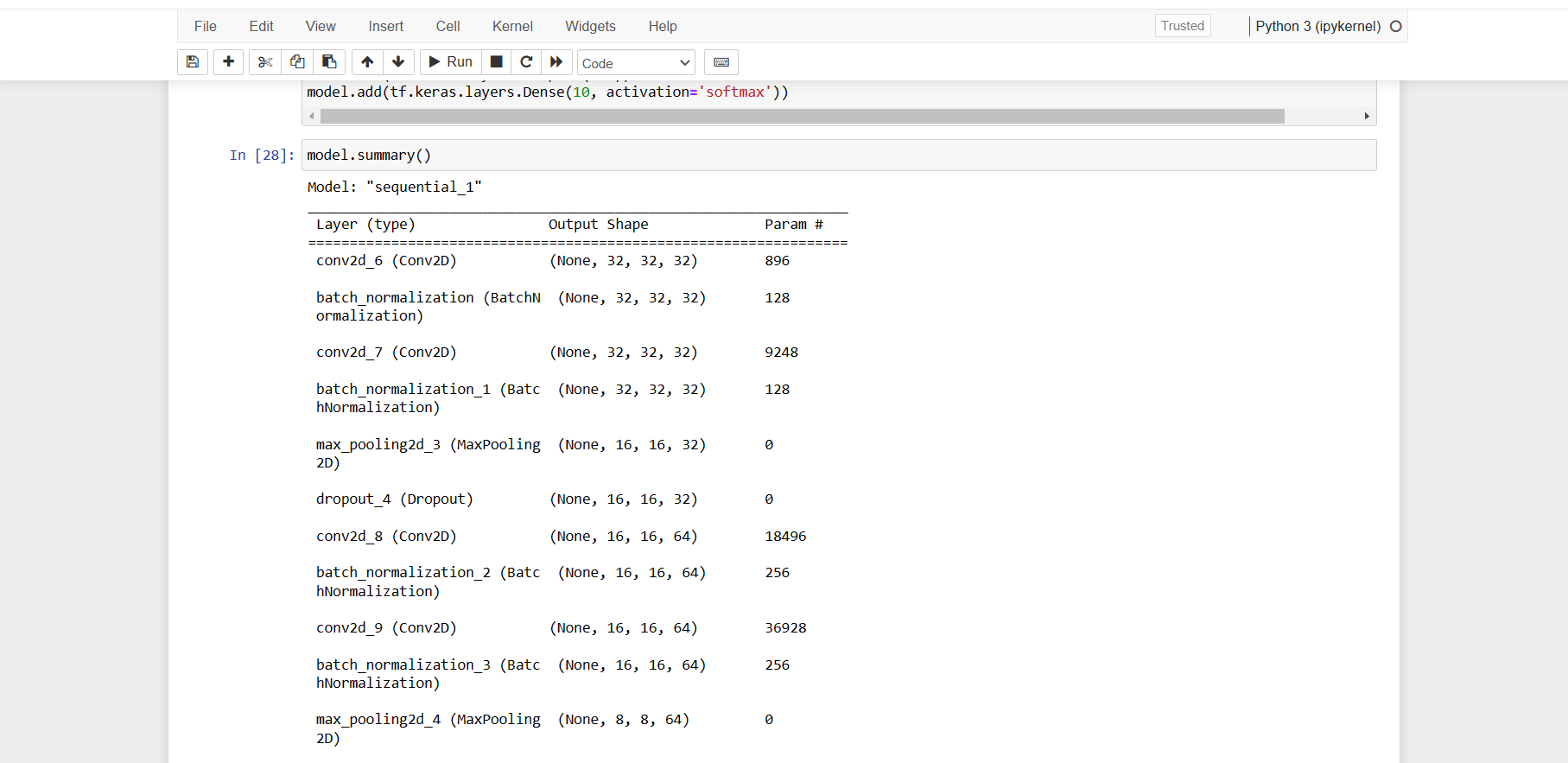
Now, I used matplotlib to display a graph of the accuracy and loss, with the training and test data, and based on it we can see that at approximately the 60th epoch, it had started being stable.

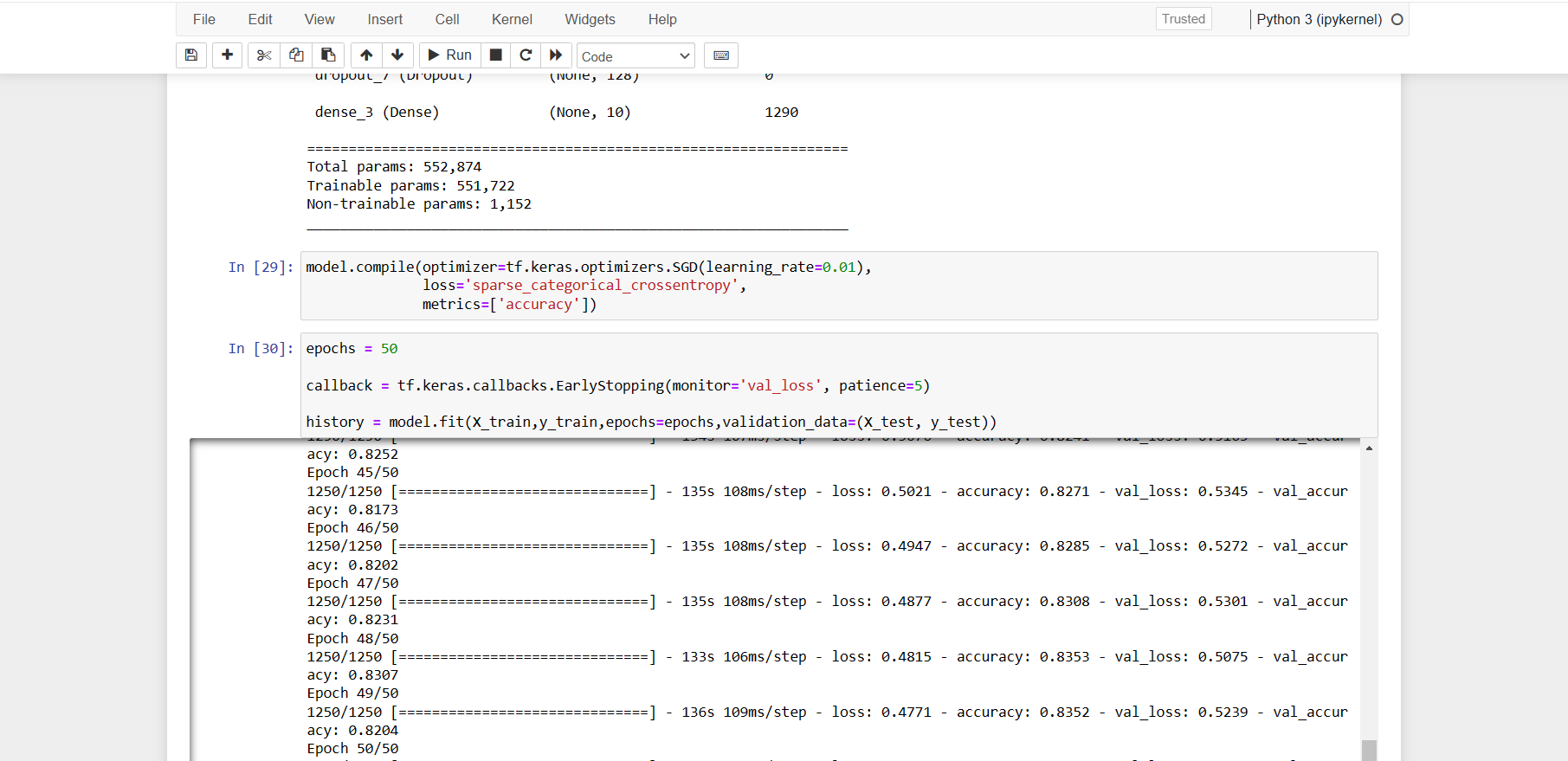




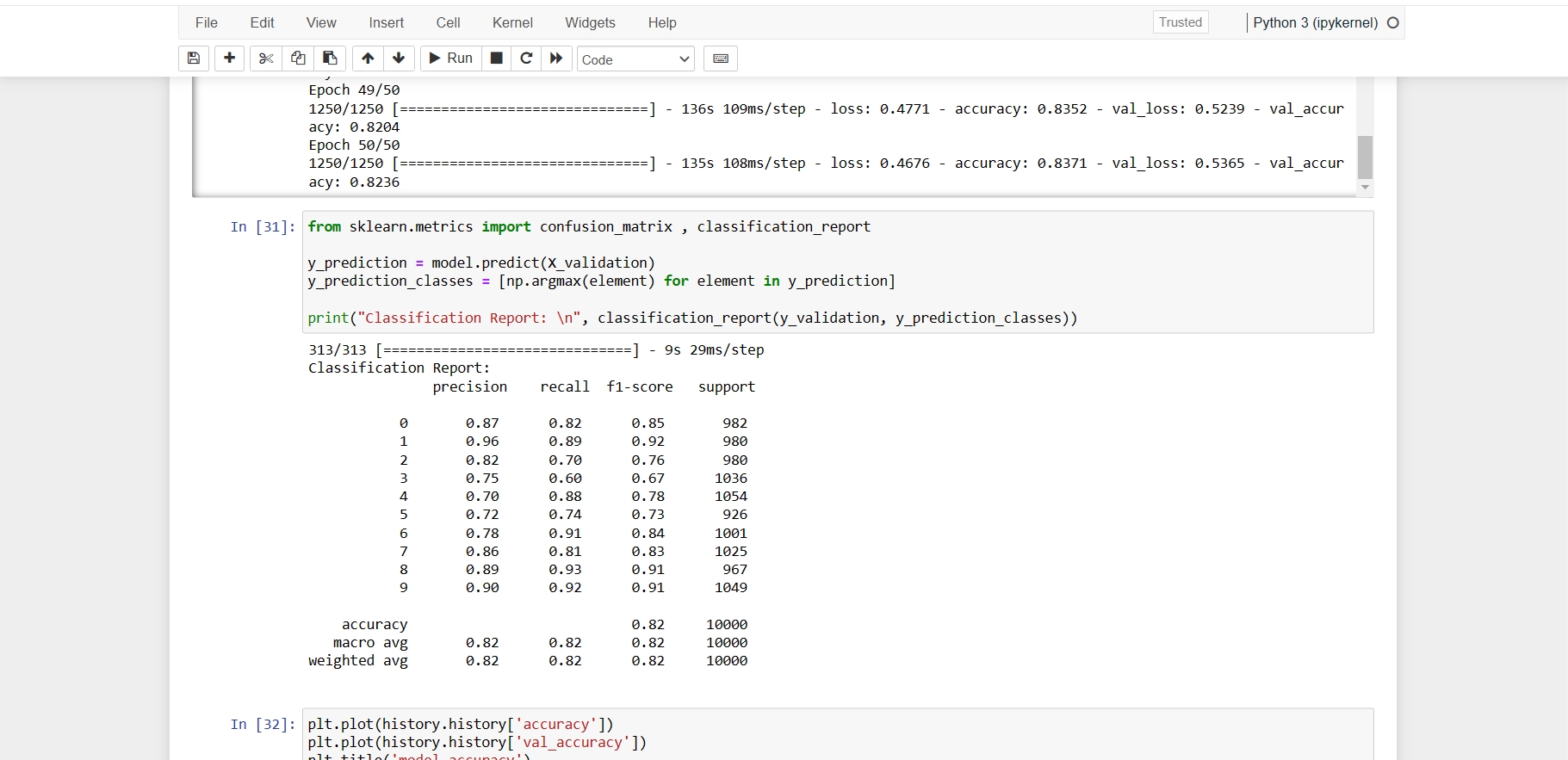
At this point, I create another model with similar values, except this time I gave it 50 epochs instead, and it reached an accuracy of 82% with a loss of 47% (based on last epoch).

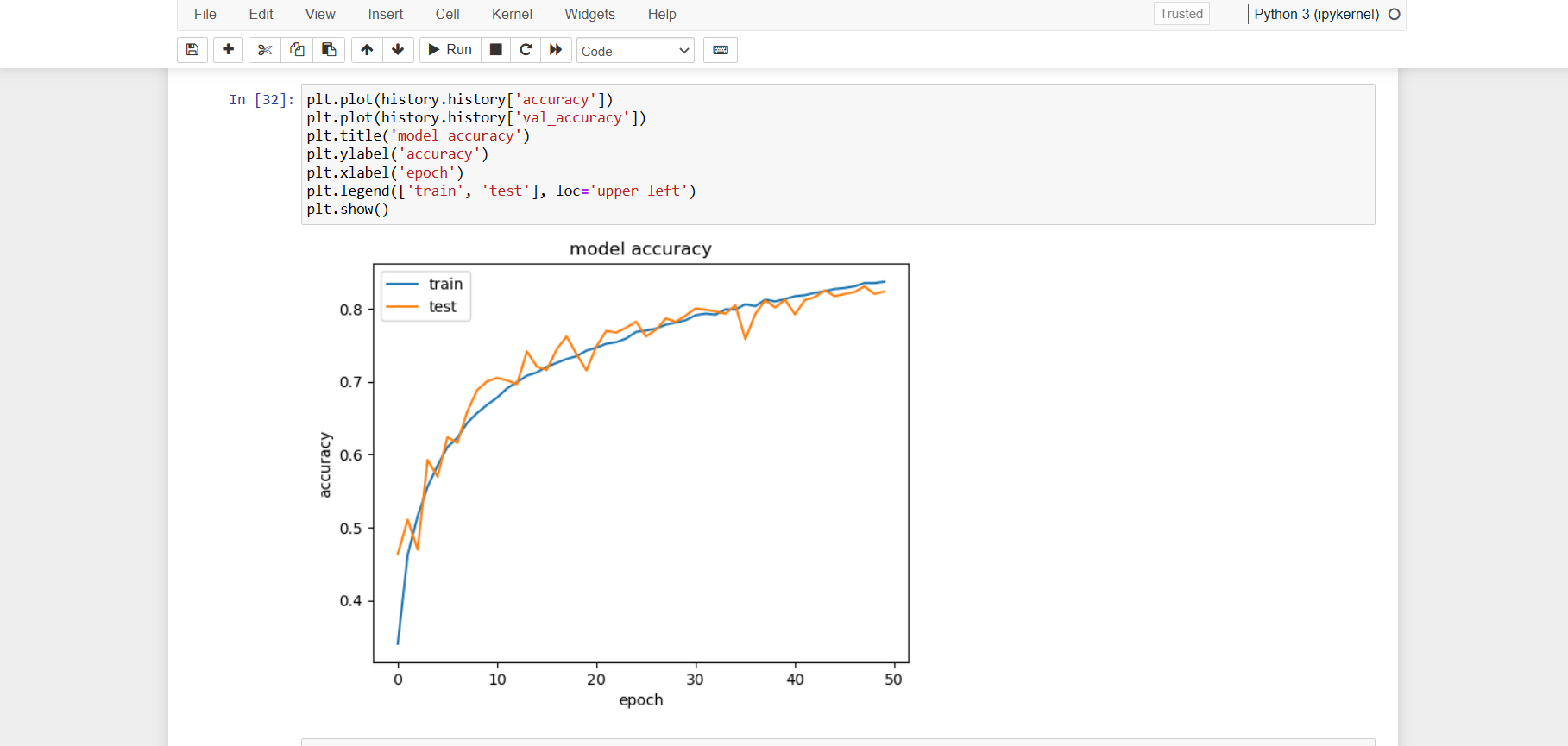


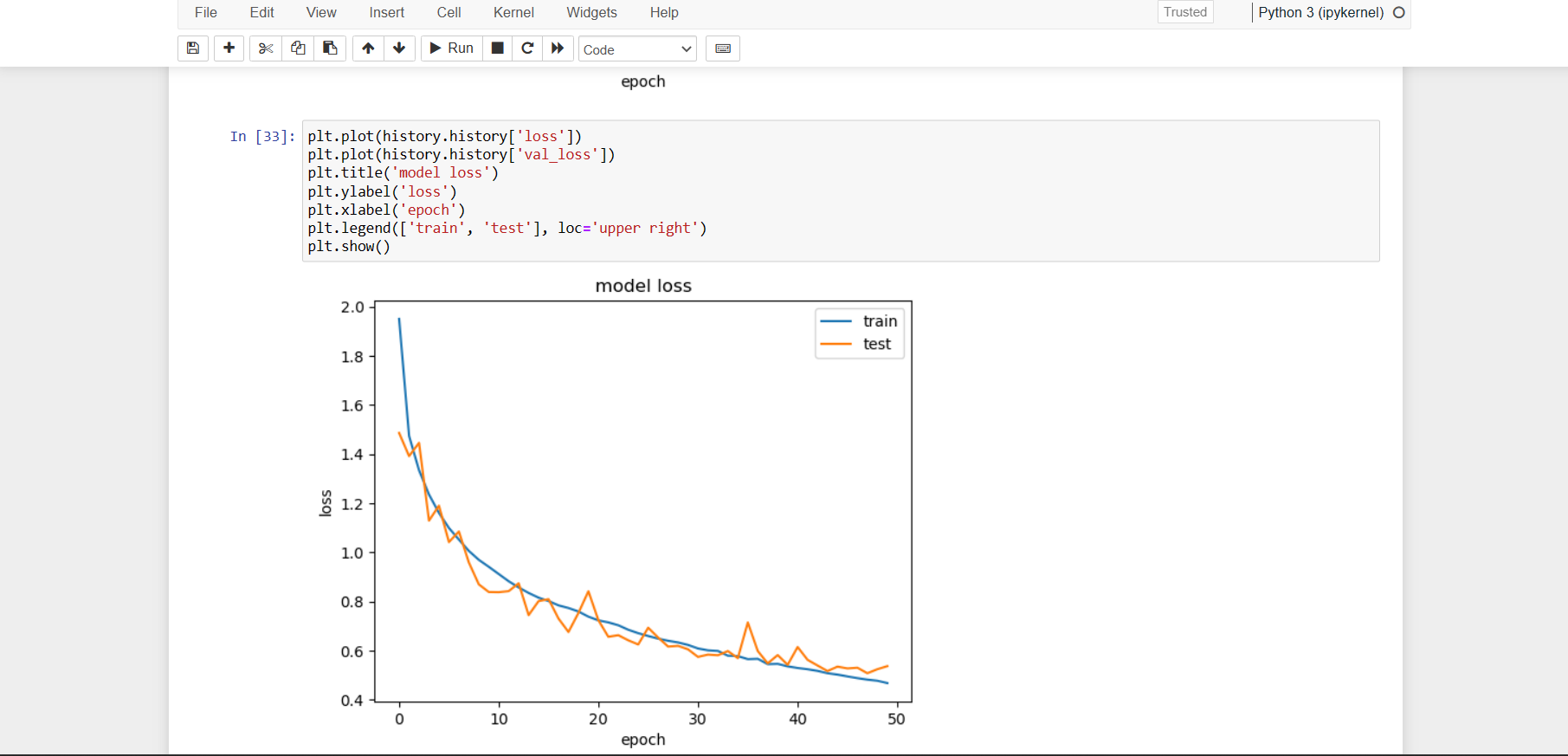




We can see based on the charts that it is fluctuating, as it had 50 epochs, whilst the previous model had started getting stable only at about 60, so based on this I concluded that perhaps 100 epochs would have been best, I had also tried to increase my laptop’s performance to let the data training take less time, though I only figured out later that there are specific ways to reduce the training time, which I did not have time to implement.







Lastly, I use model.evaluate to evaluate the model and confirm what I had concluded with the classification report, and we can see that the accuracy is indeed 82% while the loss is 53% .

