Deep Learning: Image Classification Project

Paulina Kossowska

October 2021

“One of the main objectives of this course is to help you gain hands-on experience in communicating insightful and impactful findings to stakeholders. In this project you will use the tools and techniques you learned throughout this course to use deep learning for a task of your choosing. It can be any Deep Learning application for supervised or unsupervised learning. You choose to work on a classification, image, or text application on a data set that you feel passionate about. Then, you will tweak your deep learning model to best suits your needs, and communicate insights you found from your model development exercise.”

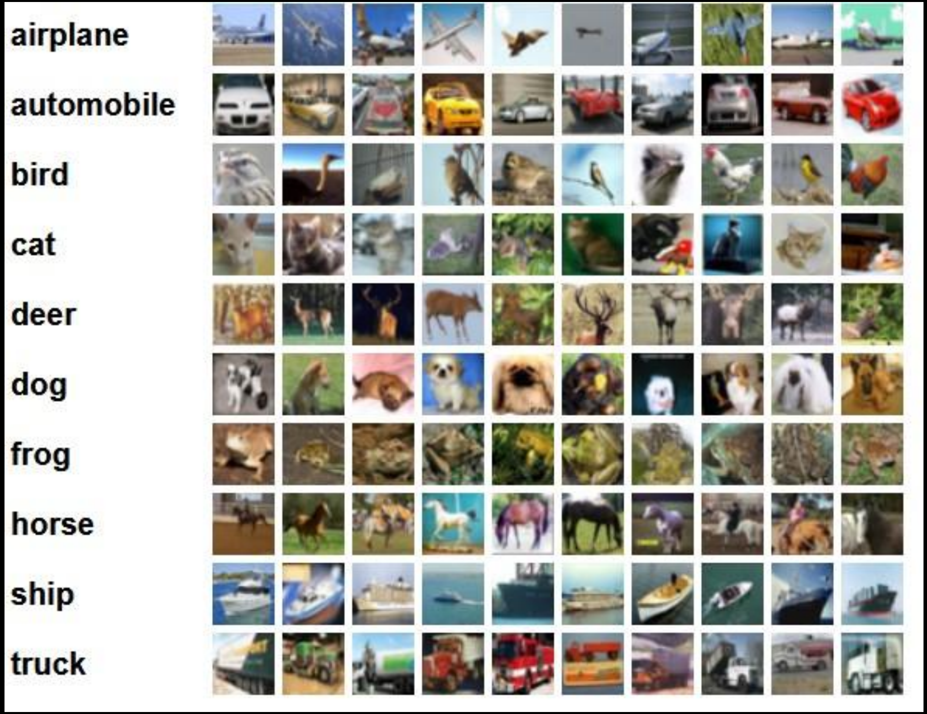
Understanding the Data

CIFAR-10 Dataset

The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.



Goals of this analysis

The main goal for this analysis:

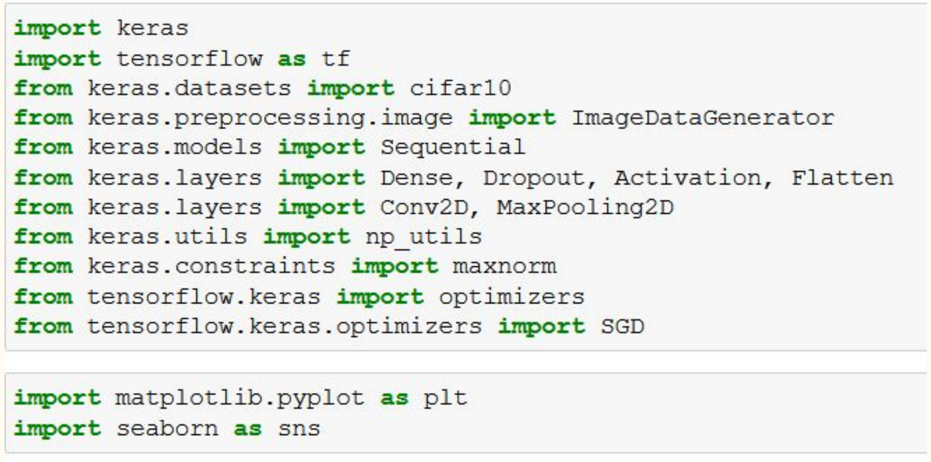
Build and train Convolutional Neural Network

1. Import needed packages
2. Load the data
3. Convert the pixel values of the dataset to float type, normalize the dataset and then perform the one-hot encoding for target classes
4. CNN Model 1
5. CNN Model 2
6. CNN Model 3
7. Summary

CIFAR-10 pre-processing

Import needed packages

The image on the right shows all packages needed to perform deep learning algorithm - Convolutional Neural Network.

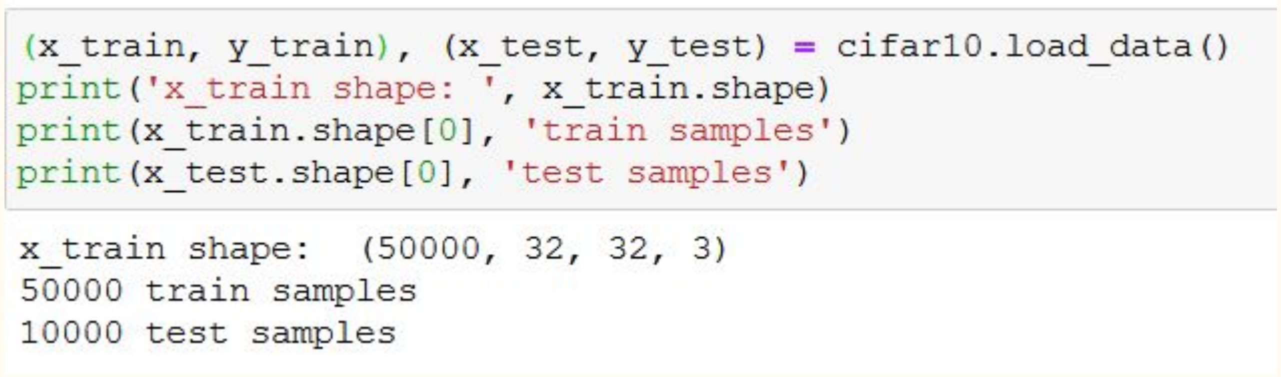


```
import keras
import tensorflow as tf
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import np_utils
from keras.constraints import maxnorm
from tensorflow.keras import optimizers
from tensorflow.keras.optimizers import SGD
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Load the Data

As I struggled with issue when I was trying to load_data() I needed first run some code to successfully load CIFAR-10 dataset. More details about this issue and solving ways could be find at GitHub



```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape: ', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

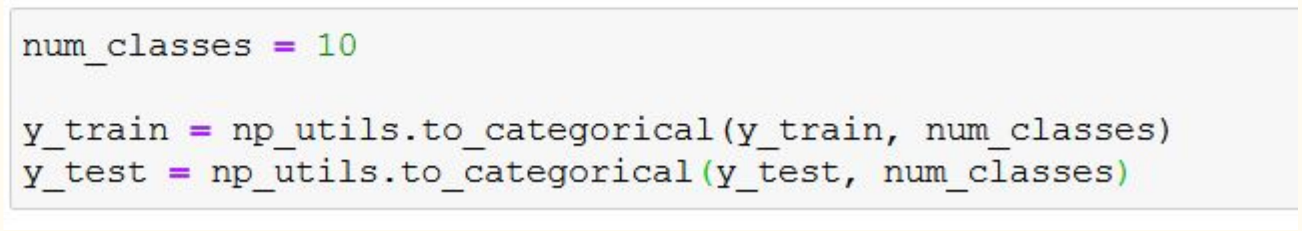
```
x_train shape:  (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

Convert the pixel values of the dataset to float type, normalize the dataset and then perform the one-hot encoding for target classes

In this step of my work I first converted **x_train** and **x_test** variables to float32 using astype() function.

Second step contains normalize dataset. To do it I simply divided my variables by 255.

Finally as image below shows I performed one-hot encoding for target variables.



```
num_classes = 10

y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)
```

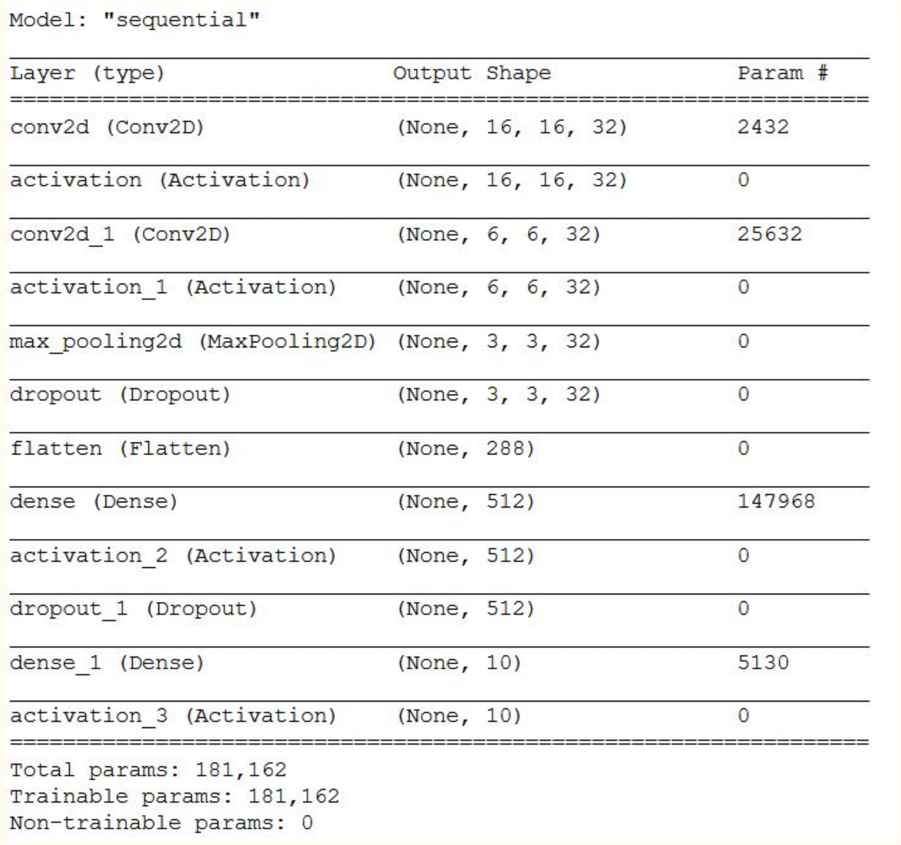
CNN Model 1

CNN Model 1

This model took a structure:

Conv → Conv → MaxPool → (Flatten) → Dense → Final Classification

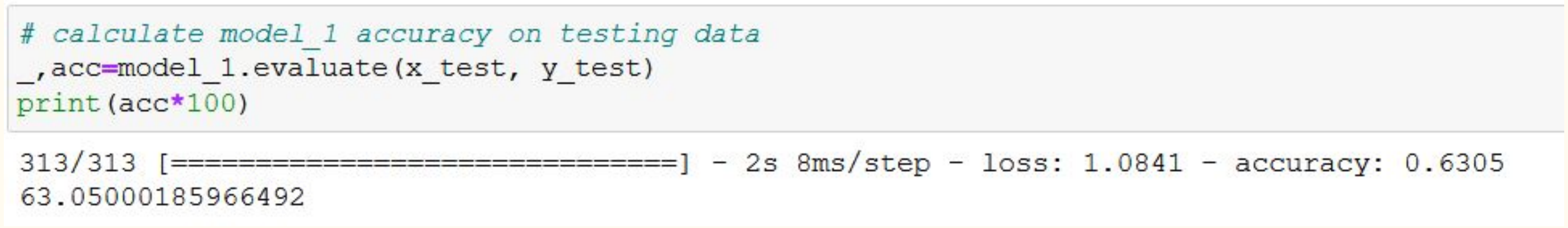
Model summary →



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 16, 32)        2432
_________________________________________________________________
activation (Activation)      (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 6, 6, 32)          25632
_________________________________________________________________
activation_1 (Activation)    (None, 6, 6, 32)          0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 3, 3, 32)          0
_________________________________________________________________
dropout (Dropout)            (None, 3, 3, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 288)               0
_________________________________________________________________
dense (Dense)                (None, 512)               147968
_________________________________________________________________
activation_2 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 181,162
Trainable params: 181,162
Non-trainable params: 0
```

CNN Model 1

I used **compile()** and **fit()** to train my first model. As optimizer I decided to use RMSprop. I used 15 epochs, so each of my models took a little time to run. In this scenario my model 1 achieved accuracy at level **63,05%.**



```
# calculate model_1 accuracy on testing data
_,acc=model_1.evaluate(x_test, y_test)
print(acc*100)
```

```
313/313 [==============================] - 2s 8ms/step - loss: 1.0841 - accuracy: 0.6305
63.05000185966492
```

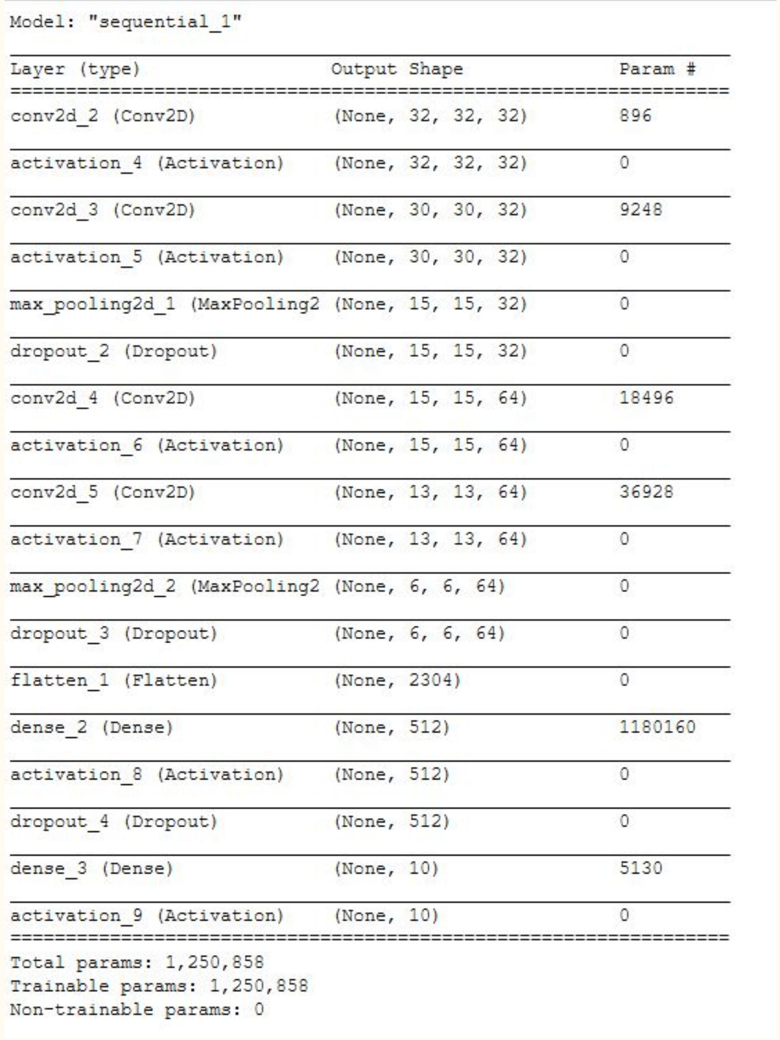
CNN Model 2

CNN Model 2

This model will be a little more complicated and take a structure:

Conv → Conv → MaxPool → Conv → Conv → MaxPool → (Flatten) → Dense → Final Classification

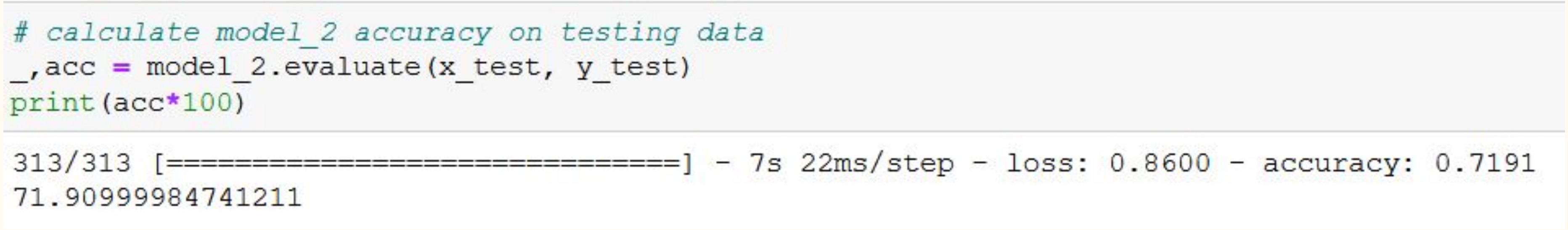
Model summary →



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_4 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
activation_5 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
activation_6 (Activation)    (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
activation_7 (Activation)    (None, 13, 13, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               1180160
_________________________________________________________________
activation_8 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_9 (Activation)    (None, 10)                0
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
```

CNN Model 2

Like in CNN Model 1 this time I also used **compile()** and **fit()** to train my second model. As a structure of it was more complicated using the same parameters like RMSprop as optimizer and 15 epochs allowed me to achieve accuracy at level **71,91%**.



```
# calculate model_2 accuracy on testing data
_,acc = model_2.evaluate(x_test, y_test)
print(acc*100)
```

```
313/313 [==============================] - 7s 22ms/step - loss: 0.8600 - accuracy: 0.7191
71.90999984741211
```

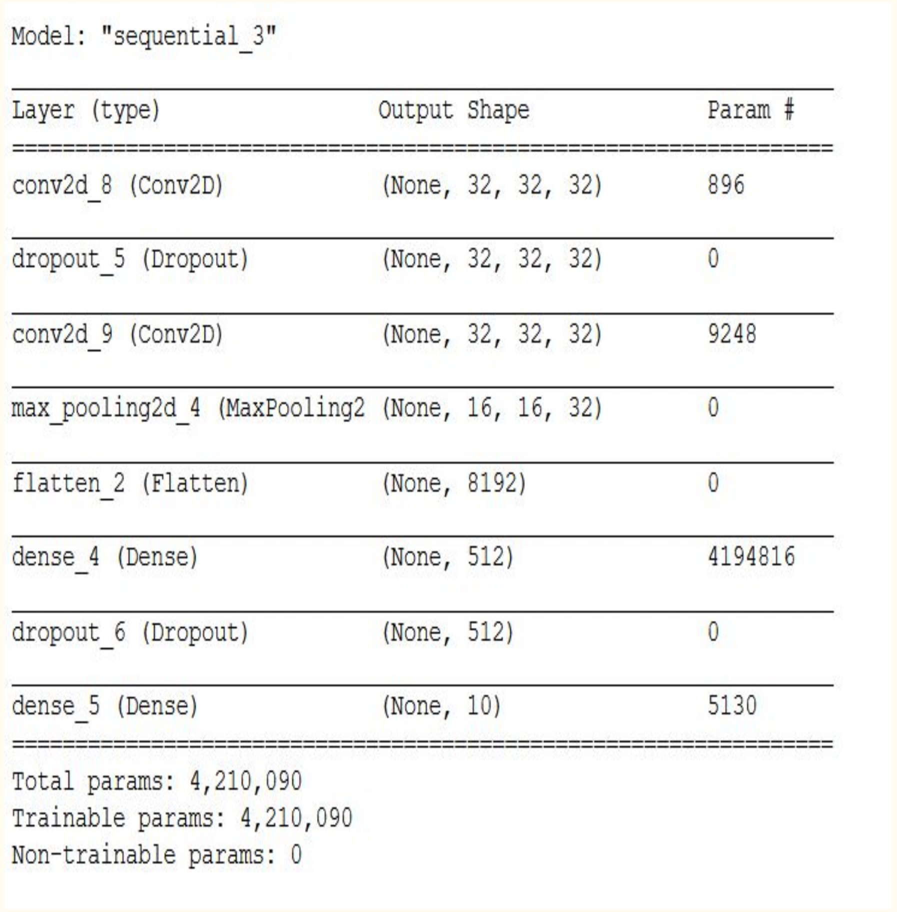
CNN Model 3

CNN Model 3

In this model I decided to use structure similar to first CNN model.

This time I had the highest number of total parameters needed to train.

The image on the right represented the **model_3.summary()**

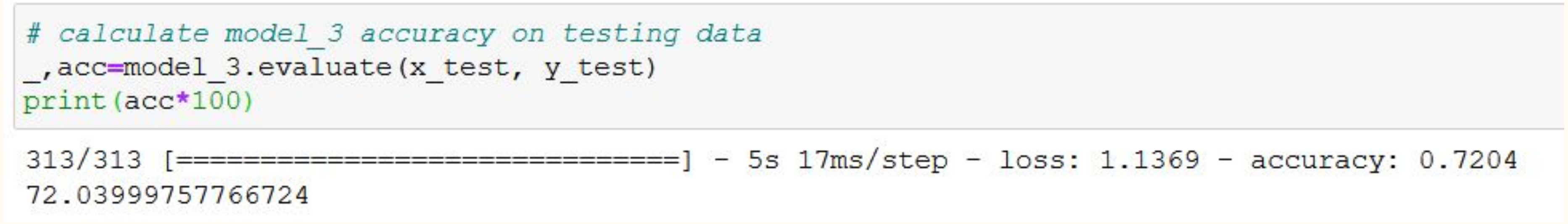


```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
dropout_5 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               4194816
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                5130
=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
```

CNN Model 3

This time like in previous steps I used **compile()** and **fit()** to train the model. But as a optimizer in this model I used **SGD**: stochastic gradient descent with a learning rate equal to 0.01 and momentum equal to 0.9

Then, fitting model on 15 epochs allowed me to achive accuracy at level **72,04%.**



```
# calculate model_3 accuracy on testing data
_,acc=model_3.evaluate(x_test, y_test)
print(acc*100)
```

```
313/313 [==============================] - 5s 17ms/step - loss: 1.1369 - accuracy: 0.7204
72.03999757766724
```

Summary and Model Evaluation

Summary

The main goal for this analysis was to build and train Convolutional Neural Network. This task was completed successfully. I build three different image classification models and each one of them achieved better accuracy. If I need to choose one of them, the model 3 will be my choice.

The models I created achieve high level of accuracy. But there are still some steps I can perform and play around with data I have. For example:

★ play around with different keras optimizers and tune there's parameters
★ build even more complicated structure for model 3 (similar to those from model 2)
★ try to train data for more epochs (but it is highly time consuming)