YILDIZ TECHNICAL UNIVERSITY
ELECTRICAL- ELECTRONICS
FACULTY
COMPUTER ENGINEERING
DEPARTMENT

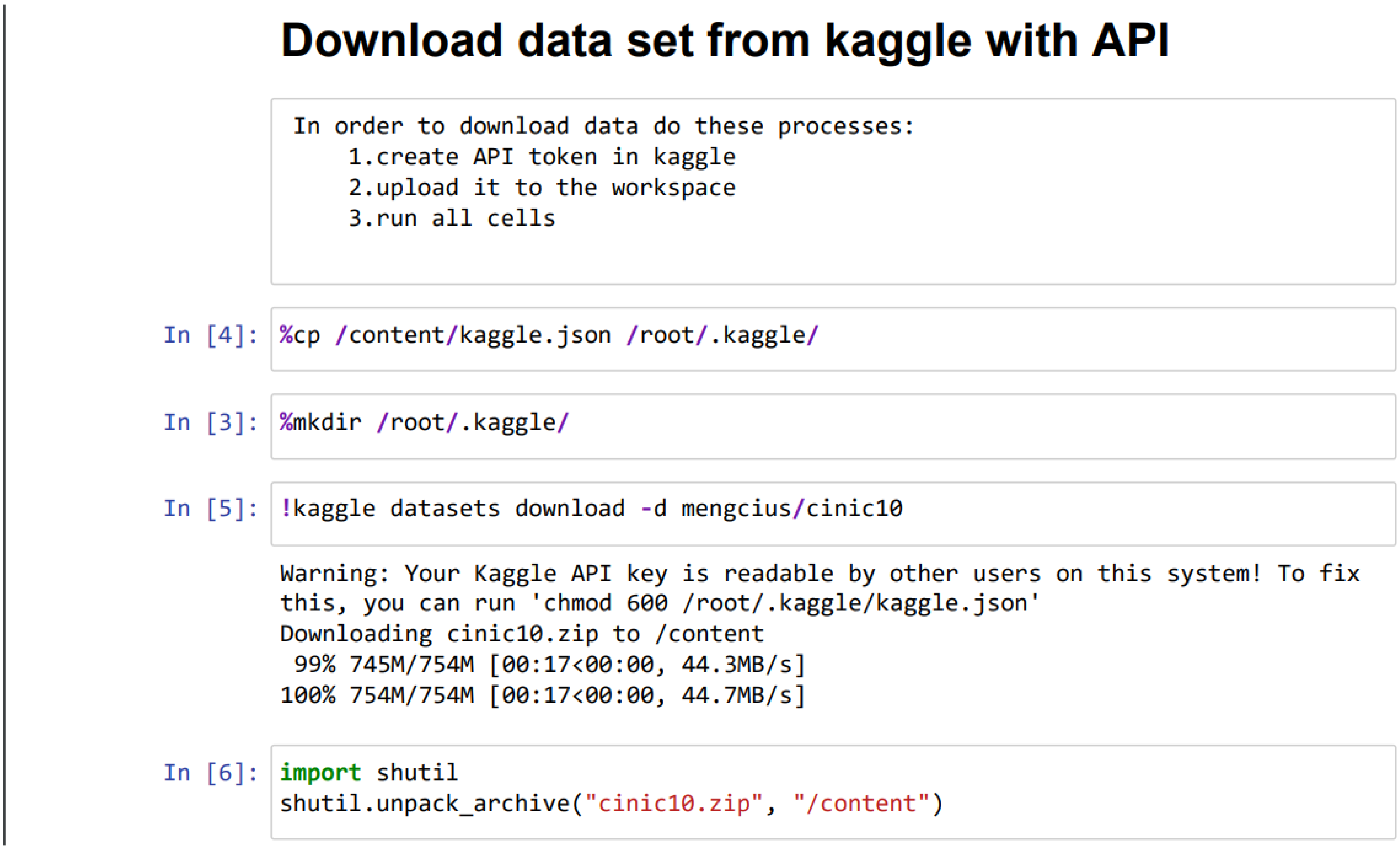
Image Processing Lecture
Semester Project

KAMRAN BALAYEV 17011904

The content of this report is creating CNN arhitecture and Realization of Transfer Learning.

CNN MODELS

In this project, i have used colab for model training. In order to get data from kaggle i need to get kaggle API and run these cells.



Download data set from kaggle with API

```
In order to download data do these processes:
    1.create API token in kaggle
    2.upload it to the workspace
    3.run all cells
```

```
In [4]: %cp /content/kaggle.json /root/.kaggle/
```

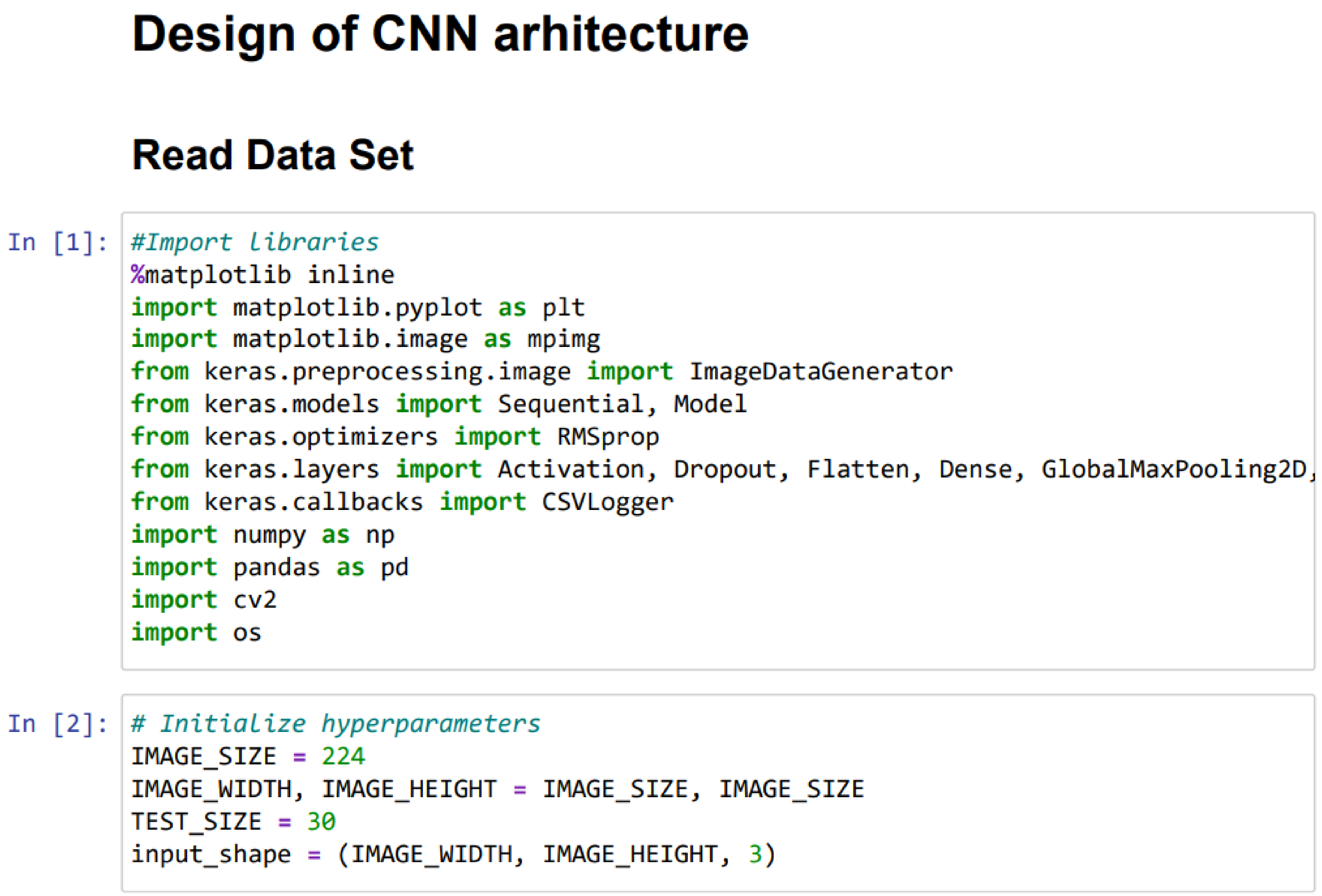
```
In [3]: %mkdir /root/.kaggle/
```

```
In [5]: !kaggle datasets download -d mengcius/cinic10
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix
this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading cinic10.zip to /content
 99% 745M/754M [00:17<00:00, 44.3MB/s]
100% 754M/754M [00:17<00:00, 44.7MB/s]
```

```
In [6]: import shutil
        shutil.unpack_archive("cinic10.zip", "/content")
```

In the design of arhitecture of CNN i have used keras library. After importing libraries , parameter initialization starts



Design of CNN arhitecture

Read Data Set

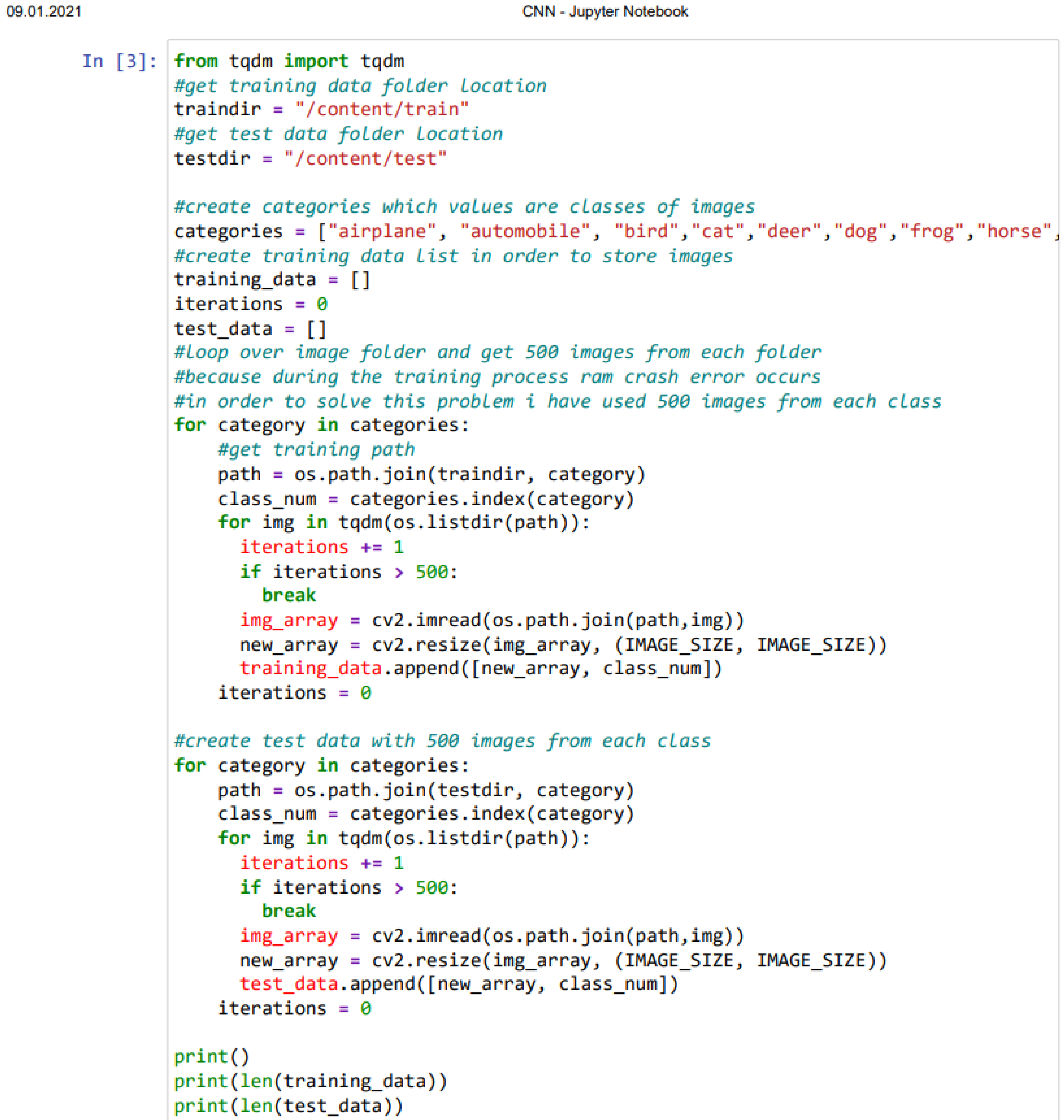
In [1]:

```
#Import libraries
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential, Model
from keras.optimizers import RMSprop
from keras.layers import Activation, Dropout, Flatten, Dense, GlobalMaxPooling2D,
from keras.callbacks import CSVLogger
import numpy as np
import pandas as pd
import cv2
import os
```

In [2]:

```
# Initialize hyperparameters
IMAGE_SIZE = 224
IMAGE_WIDTH, IMAGE_HEIGHT = IMAGE_SIZE, IMAGE_SIZE
TEST_SIZE = 30
input_shape = (IMAGE_WIDTH, IMAGE_HEIGHT, 3)
```

Create training and test data with getting them from folder

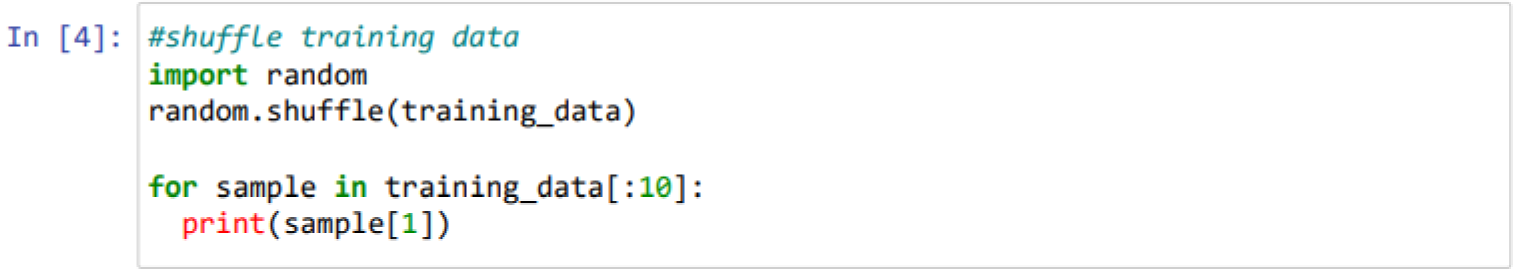


09.01.2021 CNN - Jupyter Notebook

In [3]:

```
from tqdm import tqdm
#get training data folder location
traindir = "/content/train"
#get test data folder location
testdir = "/content/test"

#create categories which values are classes of images
categories = ["airplane", "automobile", "bird","cat","deer","dog","frog","horse",
#create training data list in order to store images
training_data = []
iterations = 0
test_data = []
#loop over image folder and get 500 images from each folder
#because during the training process ram crash error occurs
#in order to solve this problem i have used 500 images from each class
for category in categories:
    #get training path
    path = os.path.join(traindir, category)
    class_num = categories.index(category)
    for img in tqdm(os.listdir(path)):
      iterations += 1
      if iterations > 500:
        break
      img_array = cv2.imread(os.path.join(path,img))
      new_array = cv2.resize(img_array, (IMAGE_SIZE, IMAGE_SIZE))
      training_data.append([new_array, class_num])
    iterations = 0

#create test data with 500 images from each class
for category in categories:
    path = os.path.join(testdir, category)
    class_num = categories.index(category)
    for img in tqdm(os.listdir(path)):
      iterations += 1
      if iterations > 500:
        break
      img_array = cv2.imread(os.path.join(path,img))
      new_array = cv2.resize(img_array, (IMAGE_SIZE, IMAGE_SIZE))
      test_data.append([new_array, class_num])
    iterations = 0

print()
print(len(training_data))
print(len(test_data))
```

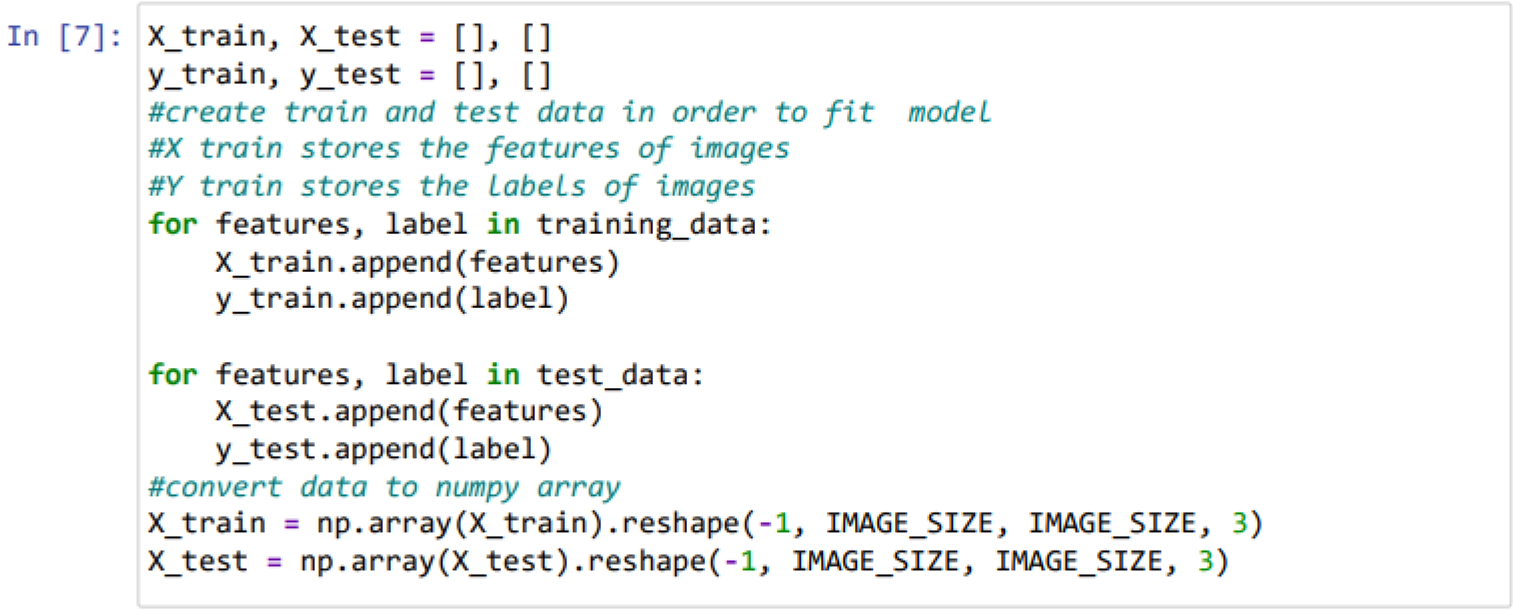
Shuffle training data in order to get proper results in model



In [4]:

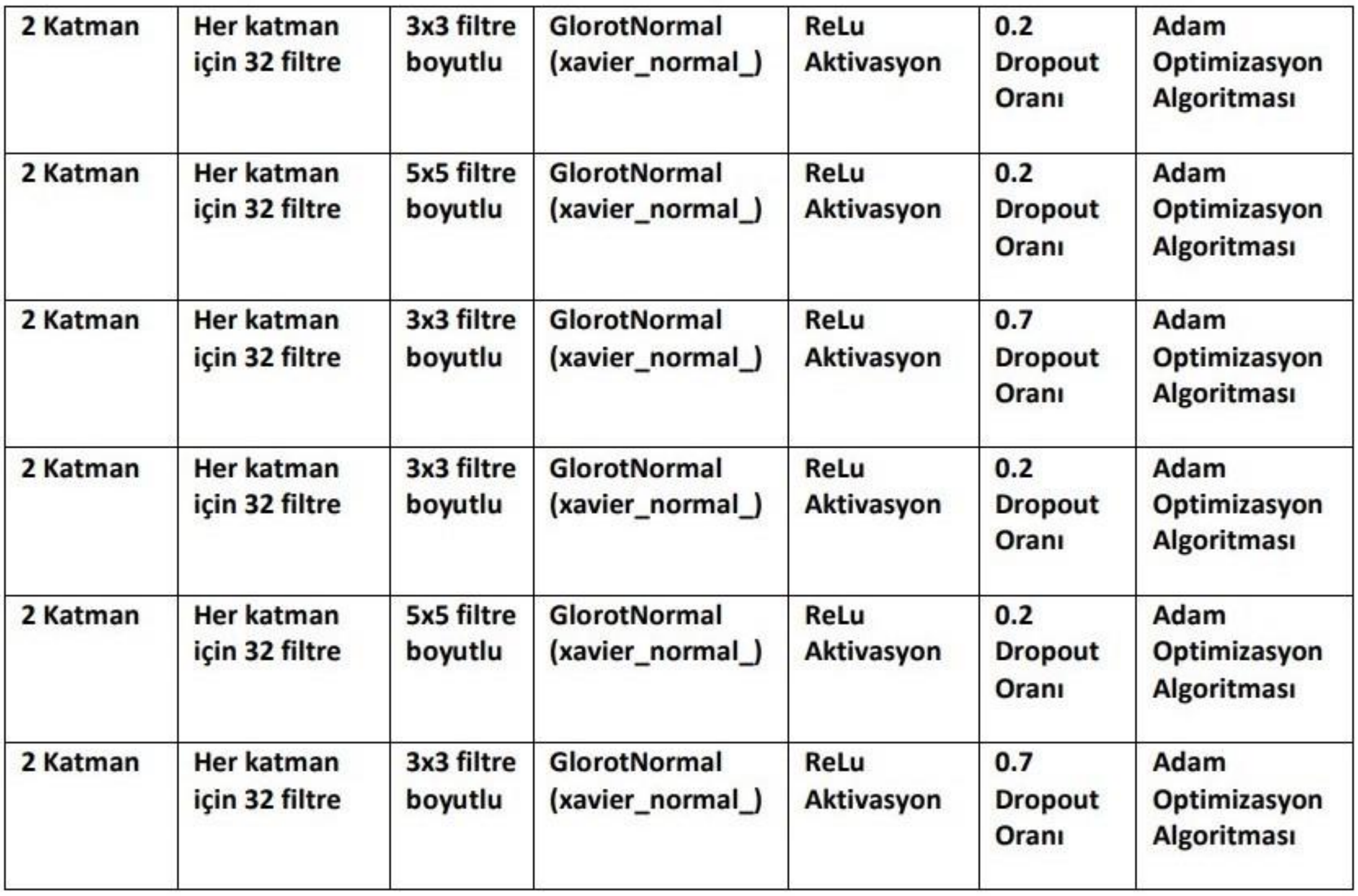
```
#shuffle training data
import random
random.shuffle(training_data)

for sample in training_data[:10]:
  print(sample[1])
```

Creaate training and test variables in order to fit model



```
In [7]: X_train, X_test = [], []
        y_train, y_test = [], []
        #create train and test data in order to fit  model
        #X train stores the features of images
        #Y train stores the labels of images
        for features, label in training_data:
            X_train.append(features)
            y_train.append(label)

        for features, label in test_data:
            X_test.append(features)
            y_test.append(label)
        #convert data to numpy array
        X_train = np.array(X_train).reshape(-1, IMAGE_SIZE, IMAGE_SIZE, 3)
        X_test = np.array(X_test).reshape(-1, IMAGE_SIZE, IMAGE_SIZE, 3)
```

Arhitecture of models in the table below:



1	2 Katman	Her katman için 32 filtre	3x3 filtre boyutlu	GlorotNormal (xavier_normal_)	ReLu Aktivasyon	0.2 Dropout Oranı	Adam Optimizasyon Algoritması
2	2 Katman	Her katman için 32 filtre	5x5 filtre boyutlu	GlorotNormal (xavier_normal_)	ReLu Aktivasyon	0.2 Dropout Oranı	Adam Optimizasyon Algoritması
3	2 Katman	Her katman için 32 filtre	3x3 filtre boyutlu	GlorotNormal (xavier_normal_)	ReLu Aktivasyon	0.7 Dropout Oranı	Adam Optimizasyon Algoritması
4	2 Katman	Her katman için 32 filtre	3x3 filtre boyutlu	GlorotNormal (xavier_normal_)	ReLu Aktivasyon	0.2 Dropout Oranı	Adam Optimizasyon Algoritması
5	2 Katman	Her katman için 32 filtre	5x5 filtre boyutlu	GlorotNormal (xavier_normal_)	ReLu Aktivasyon	0.2 Dropout Oranı	Adam Optimizasyon Algoritması
6	2 Katman	Her katman için 32 filtre	3x3 filtre boyutlu	GlorotNormal (xavier_normal_)	ReLu Aktivasyon	0.7 Dropout Oranı	Adam Optimizasyon Algoritması

In [11]:

```
#Create CNN model with Keras
'''
MODEL 1 includes :
    # 2 layers
    # 32 filter for each layer
    # 3x3 filter size
    # 0.2 Dropout rate
'''


model = Sequential()

model.add(Conv2D(32, (3,3), kernel_initializer='GlorotNormal', input_shape=input_shape, activation='relu'))
model.add(Dropout(0.2))

model.add(Conv2D(32, (3,3), kernel_initializer='GlorotNormal', activation='relu'))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(len(categories), activation="softmax"))

model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])

model.summary()
model.fit(X_train, y_train, batch_size=32,  epochs=10)
```

In [12]:

```
#Create CNN model with Keras
'''
MODEL 2 includes :
    # 2 layers
    # 32 filter for each layer
    # 5x5 filter size
    # 0.2 Dropout rate
'''
model = Sequential()

model.add(Conv2D(32, (5,5), kernel_initializer='GlorotNormal', input_shape=input_shape, activation='relu'))
model.add(Dropout(0.2))

model.add(Conv2D(32, (5,5), kernel_initializer='GlorotNormal', activation='relu'))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(len(categories), activation="softmax"))
```

In [14]:

```
#Create CNN model with Keras
'''
MODEL 3 includes :
    # 2 layers
    # 32 filter for each layer
    # 3x3 filter size
    # 0.7 Dropout rate
'''
model = Sequential()

model.add(Conv2D(32, (3,3), kernel_initializer='GlorotNormal', input_shape=input_shape, activation='relu'))
model.add(Dropout(0.7))

model.add(Conv2D(32, (3,3), kernel_initializer='GlorotNormal', activation='relu'))
model.add(Dropout(0.7))

model.add(Flatten())
model.add(Dense(len(categories), activation="softmax"))
```

In [11]:

```
#Create CNN model with Keras
'''
MODEL 4 includes :
    # 2 layers
    # 32 filter for each layer
    # 3x3 filter size
    # 0.2 Dropout rate
'''
model = Sequential()

model.add(Conv2D(32, (3,3), kernel_initializer='GlorotNormal', input_shape=input_shape, activation='relu'))
model.add(Dropout(0.2))

model.add(Conv2D(32, (3,3), kernel_initializer='GlorotNormal', activation='relu'))
model.add(Dropout(0.2))

model.add(Conv2D(32, (3,3), kernel_initializer='GlorotNormal', activation='relu'))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(len(categories), activation="softmax"))

model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])

model.summary()
model.fit(X_train, y_train, batch_size=32,  epochs=10)
```

In [12]:

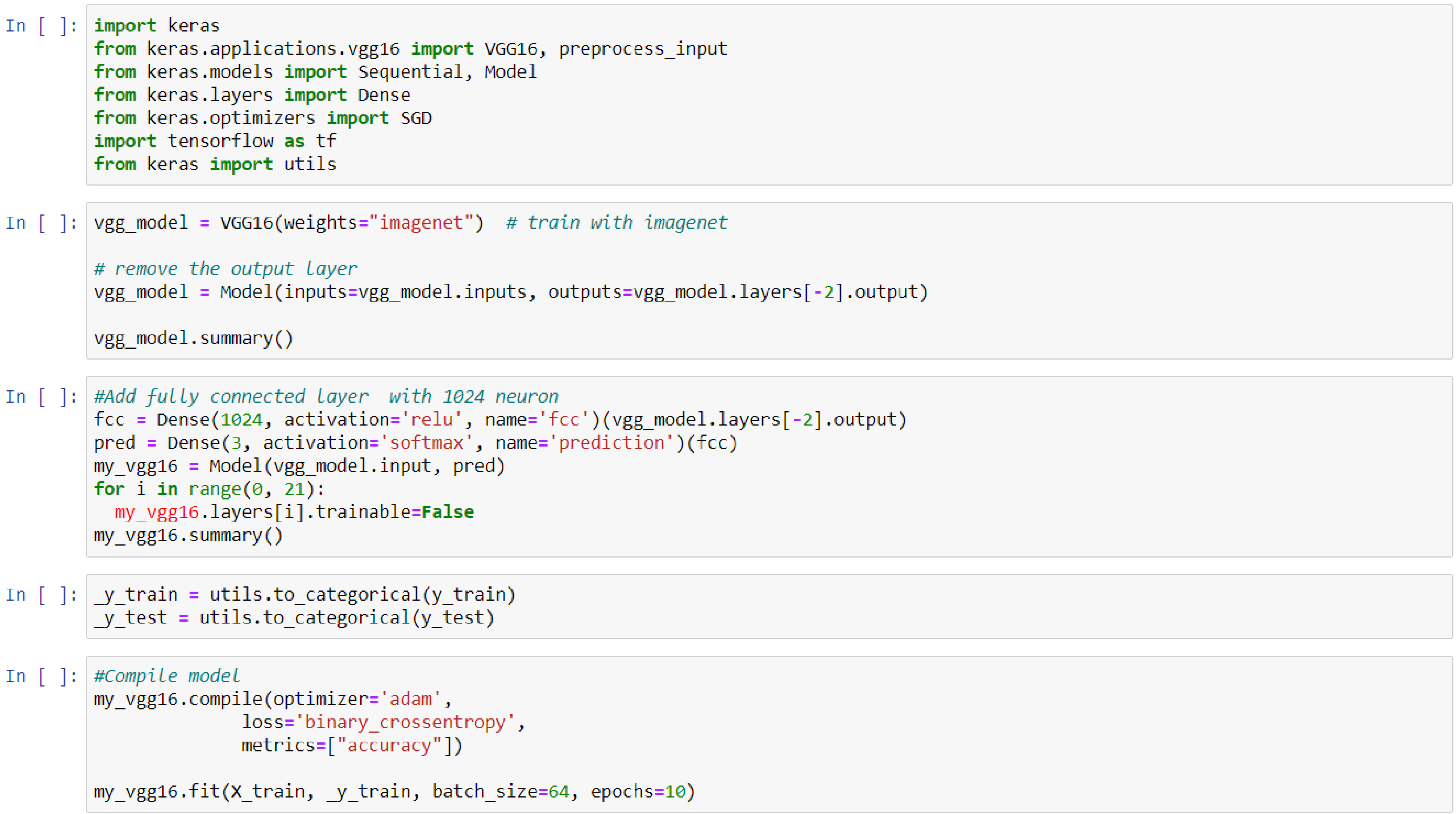
```
#Create CNN model with Keras
'''
MODEL 5 includes :
    # 2 layers
    # 32 filter for each layer
    # 5x5 filter size
    # 0.2 Dropout rate
'''
model = Sequential()

model.add(Conv2D(32, (5,5), kernel_initializer='GlorotNormal', input_shape=input_shape, activation='relu'))
model.add(Dropout(0.2))

model.add(Conv2D(32, (5,5), kernel_initializer='GlorotNormal', activation='relu'))
model.add(Dropout(0.2))

model.add(Conv2D(32, (5,5), kernel_initializer='GlorotNormal', activation='relu'))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(len(categories), activation="softmax"))

model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])

model.summary()
model.fit(X_train, y_train, batch_size=32,  epochs=10)
```

In [13]:

```
#Create CNN model with Keras
'''
MODEL 6 includes :
    # 2 layers
    # 32 filter for each layer
    # 3x3 filter size
    # 0.7 Dropout rate
'''
model = Sequential()

model.add(Conv2D(32, (3,3), kernel_initializer='GlorotNormal', input_shape=input_shape, activation='relu'))
model.add(Dropout(0.7))

model.add(Conv2D(32, (3,3), kernel_initializer='GlorotNormal', activation='relu'))
model.add(Dropout(0.7))

model.add(Conv2D(32, (3,3), kernel_initializer='GlorotNormal', activation='relu'))
model.add(Dropout(0.7))

model.add(Flatten())
model.add(Dense(len(categories), activation="softmax"))

model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])

model.summary()
model.fit(X_train, y_train, batch_size=32,  epochs=10)
```

VGG MODEL

Remove last layer, add new layer with 1024 neuron and then compile and fit model.



In [ ]:

```
import keras
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.models import Sequential, Model
from keras.layers import Dense
from keras.optimizers import SGD
import tensorflow as tf
from keras import utils
```

In [ ]:

```
vgg_model = VGG16(weights="imagenet")  # train with imagenet

# remove the output layer
vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)

vgg_model.summary()
```

In [ ]:

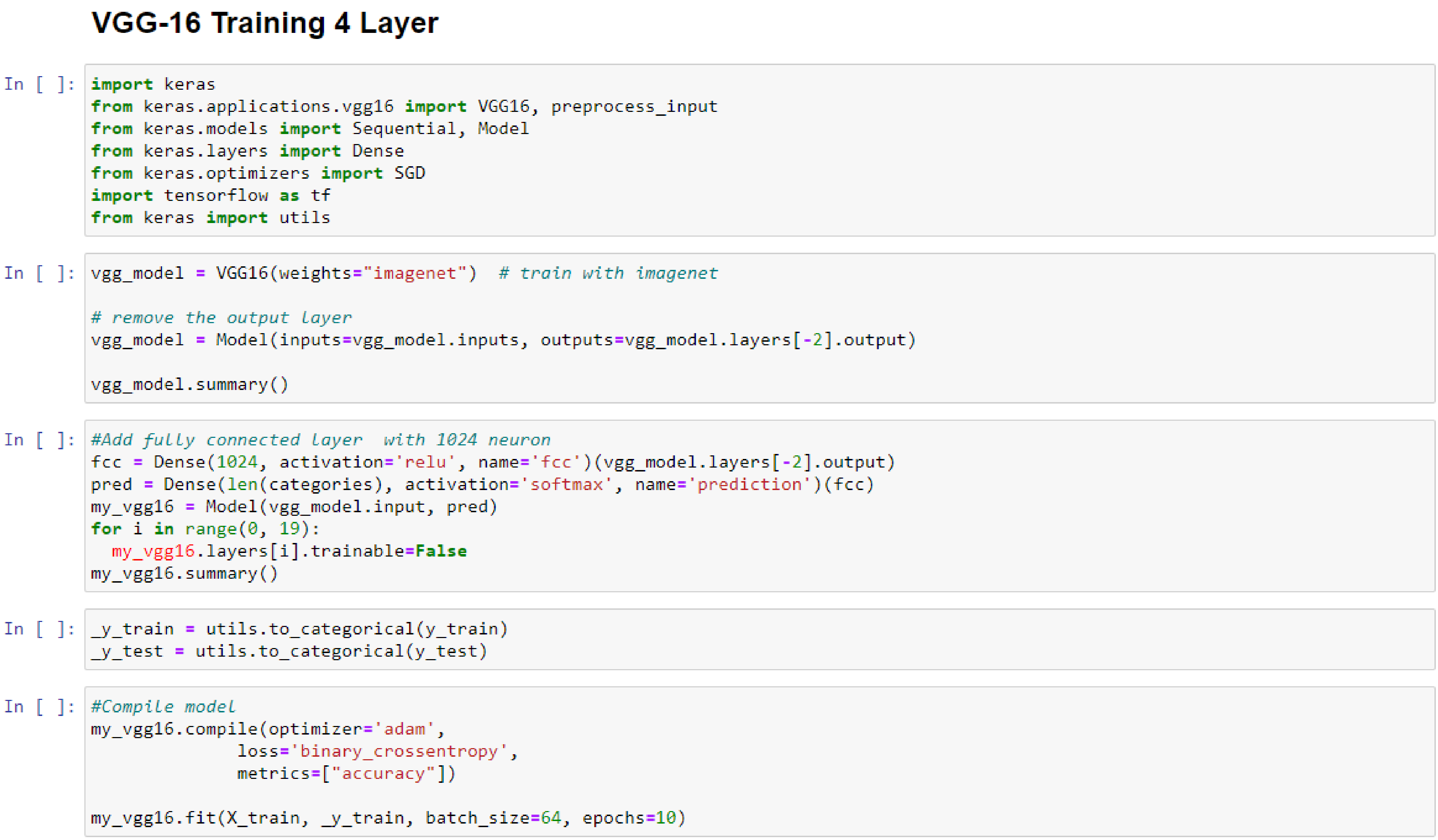
```
#Add fully connected layer  with 1024 neuron
fcc = Dense(1024, activation='relu', name='fcc')(vgg_model.layers[-2].output)
pred = Dense(3, activation='softmax', name='prediction')(fcc)
my_vgg16 = Model(vgg_model.input, pred)
for i in range(0, 21):
  my_vgg16.layers[i].trainable=False
my_vgg16.summary()
```

In [ ]:

```
_y_train = utils.to_categorical(y_train)
_y_test = utils.to_categorical(y_test)
```

In [ ]:

```
#Compile model
my_vgg16.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=["accuracy"])

my_vgg16.fit(X_train, _y_train, batch_size=64, epochs=10)
```



VGG-16 Training 4 Layer

In [ ]:

```
import keras
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.models import Sequential, Model
from keras.layers import Dense
from keras.optimizers import SGD
import tensorflow as tf
from keras import utils
```

In [ ]:

```
vgg_model = VGG16(weights="imagenet")  # train with imagenet

# remove the output layer
vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)

vgg_model.summary()
```

In [ ]:

```
#Add fully connected layer  with 1024 neuron
fcc = Dense(1024, activation='relu', name='fcc')(vgg_model.layers[-2].output)
pred = Dense(len(categories), activation='softmax', name='prediction')(fcc)
my_vgg16 = Model(vgg_model.input, pred)
for i in range(0, 19):
  my_vgg16.layers[i].trainable=False
my_vgg16.summary()
```

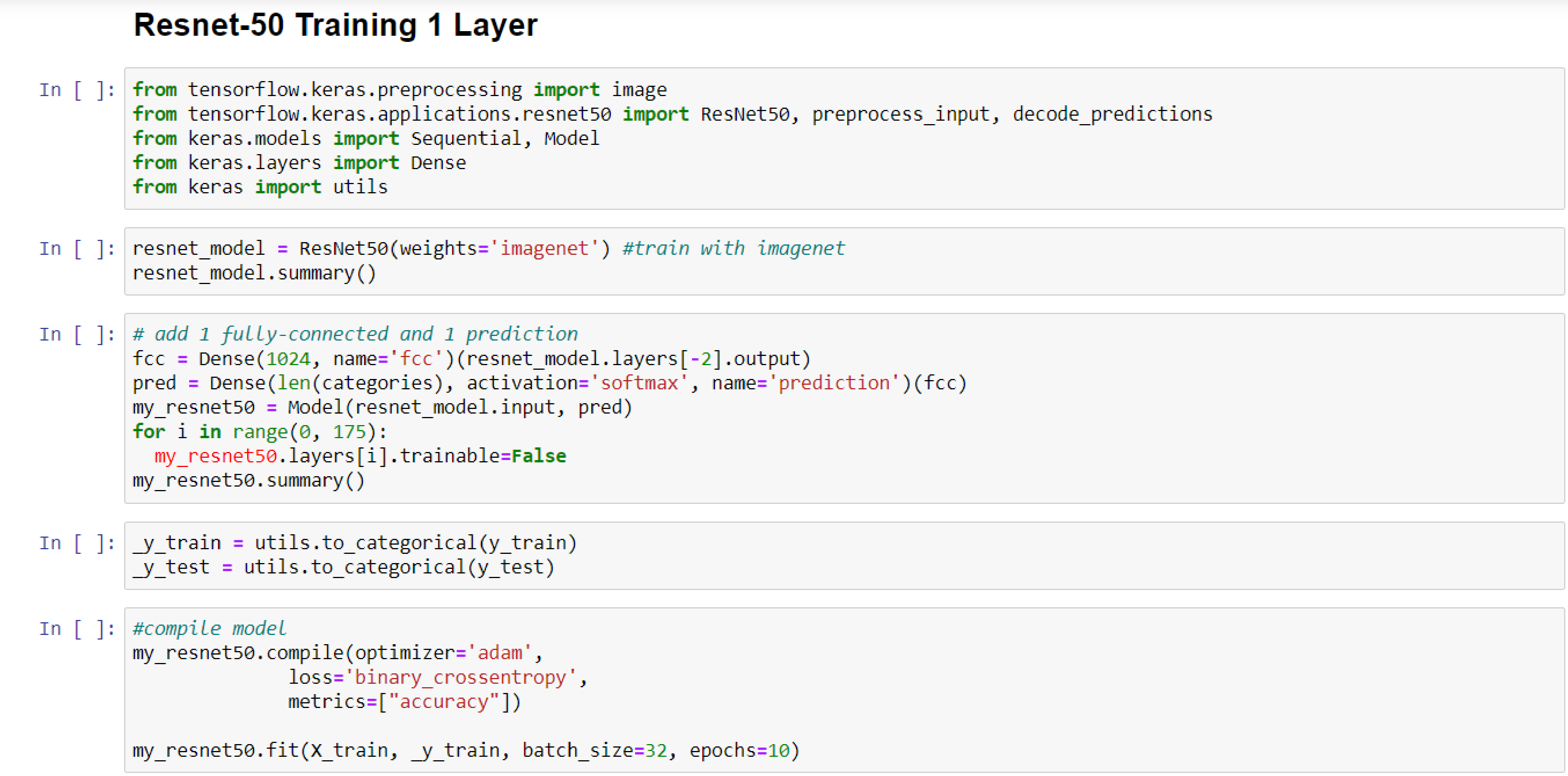
In [ ]:

```
_y_train = utils.to_categorical(y_train)
_y_test = utils.to_categorical(y_test)
```

In [ ]:

```
#Compile model
my_vgg16.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=["accuracy"])

my_vgg16.fit(X_train, _y_train, batch_size=64, epochs=10)
```

RESNET MODEL



Resnet-50 Training 1 Layer

```
In [ ]: from tensorflow.keras.preprocessing import image
        from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input, decode_predictions
        from keras.models import Sequential, Model
        from keras.layers import Dense
        from keras import utils
```

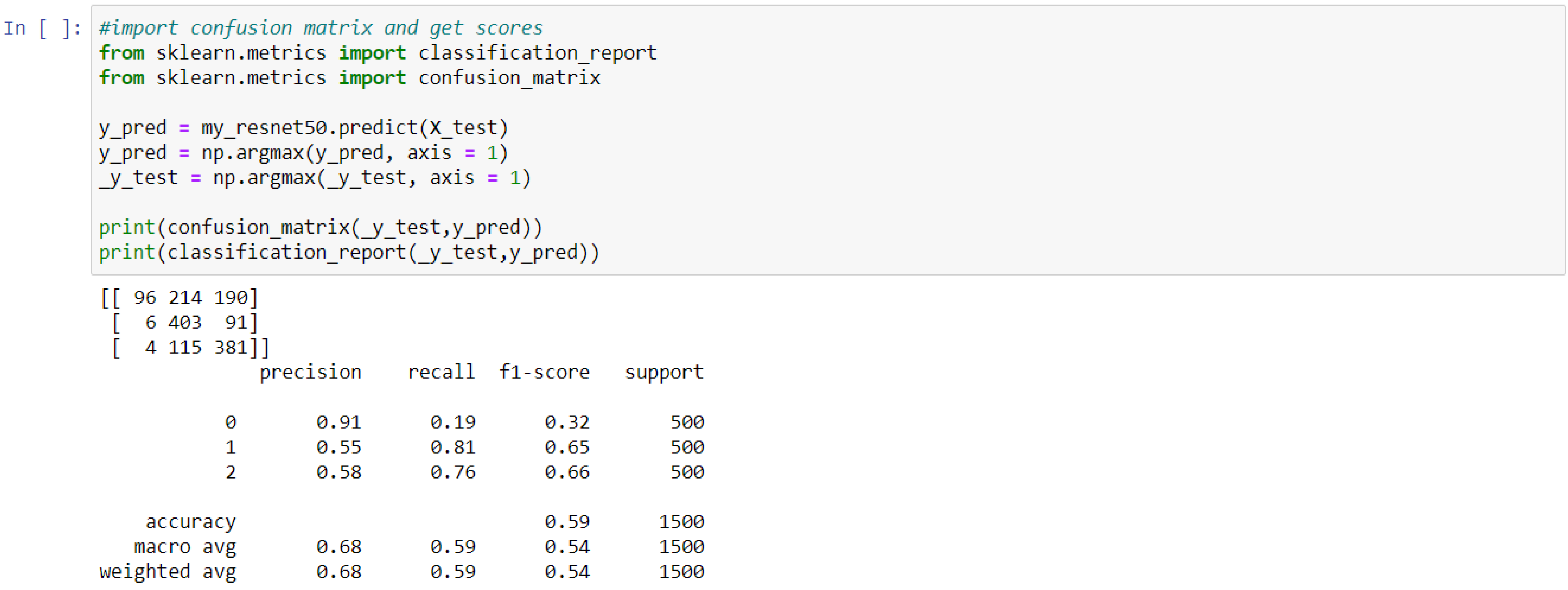
```
In [ ]: resnet_model = ResNet50(weights='imagenet') #train with imagenet
        resnet_model.summary()
```

```
In [ ]: # add 1 fully-connected and 1 prediction
        fcc = Dense(1024, name='fcc')(resnet_model.layers[-2].output)
        pred = Dense(len(categories), activation='softmax', name='prediction')(fcc)
        my_resnet50 = Model(resnet_model.input, pred)
        for i in range(0, 175):
          my_resnet50.layers[i].trainable=False
        my_resnet50.summary()
```

```
In [ ]: _y_train = utils.to_categorical(y_train)
        _y_test = utils.to_categorical(y_test)
```

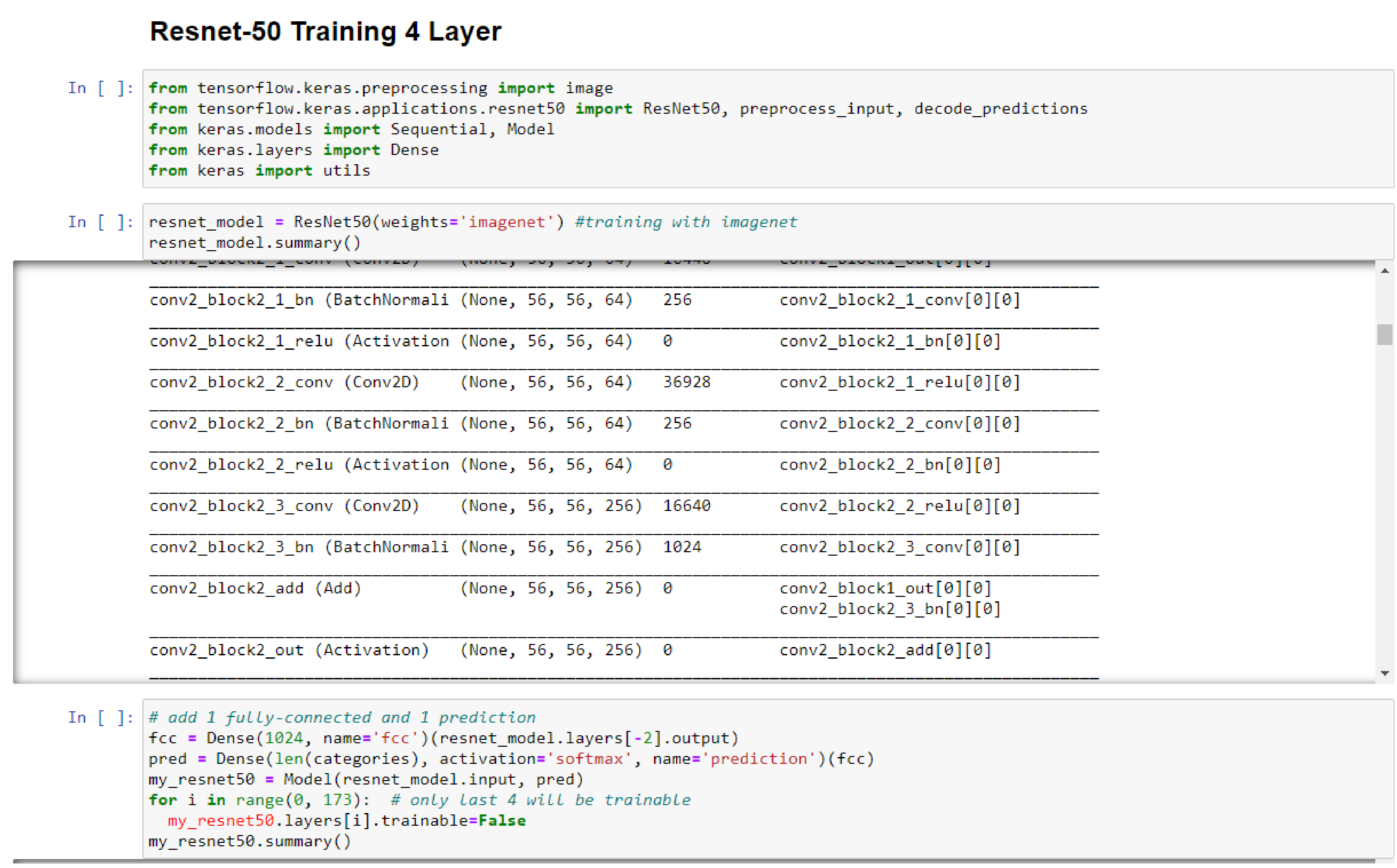
```
In [ ]: #compile model
        my_resnet50.compile(optimizer='adam',
                     loss='binary_crossentropy',
                     metrics=["accuracy"])

        my_resnet50.fit(X_train, _y_train, batch_size=32, epochs=10)
```

Confusion matrix results:



```
In [ ]: #import confusion matrix and get scores
        from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix

        y_pred = my_resnet50.predict(X_test)
        y_pred = np.argmax(y_pred, axis = 1)
        _y_test = np.argmax(_y_test, axis = 1)

        print(confusion_matrix(_y_test,y_pred))
        print(classification_report(_y_test,y_pred))
```

```
[[ 96 214 190]
 [  6 403  91]
 [  4 115 381]]
              precision    recall  f1-score   support

           0       0.91      0.19      0.32       500
           1       0.55      0.81      0.65       500
           2       0.58      0.76      0.66       500

    accuracy                           0.59      1500
   macro avg       0.68      0.59      0.54      1500
weighted avg       0.68      0.59      0.54      1500
```



Resnet-50 Training 4 Layer

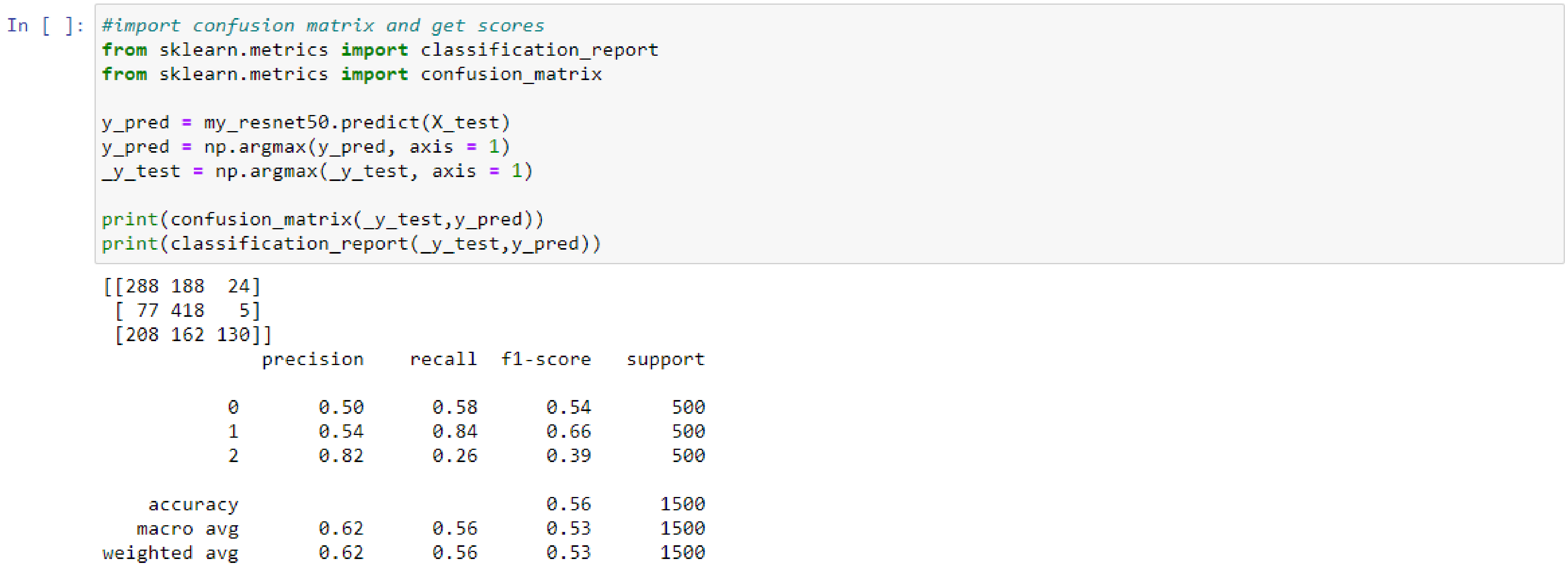
```
In [ ]: from tensorflow.keras.preprocessing import image
        from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input, decode_predictions
        from keras.models import Sequential, Model
        from keras.layers import Dense
        from keras import utils
```

```
In [ ]: resnet_model = ResNet50(weights='imagenet') #training with imagenet
        resnet_model.summary()
```

```
conv2_block2_1_conv (Conv2D)    (None, 56, 56, 64)   16448       conv2_block1_out[0][0]
__________________________________________________________________________________________________
conv2_block2_1_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block2_1_conv[0][0]
__________________________________________________________________________________________________
conv2_block2_1_relu (Activation (None, 56, 56, 64)   0           conv2_block2_1_bn[0][0]
__________________________________________________________________________________________________
conv2_block2_2_conv (Conv2D)    (None, 56, 56, 64)   36928       conv2_block2_1_relu[0][0]
__________________________________________________________________________________________________
conv2_block2_2_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block2_2_conv[0][0]
__________________________________________________________________________________________________
conv2_block2_2_relu (Activation (None, 56, 56, 64)   0           conv2_block2_2_bn[0][0]
__________________________________________________________________________________________________
conv2_block2_3_conv (Conv2D)    (None, 56, 56, 256)  16640       conv2_block2_2_relu[0][0]
__________________________________________________________________________________________________
conv2_block2_3_bn (BatchNormali (None, 56, 56, 256)  1024        conv2_block2_3_conv[0][0]
__________________________________________________________________________________________________
conv2_block2_add (Add)          (None, 56, 56, 256)  0           conv2_block1_out[0][0]
                                                                 conv2_block2_3_bn[0][0]
__________________________________________________________________________________________________
conv2_block2_out (Activation)   (None, 56, 56, 256)  0           conv2_block2_add[0][0]
__________________________________________________________________________________________________
```

```
In [ ]: # add 1 fully-connected and 1 prediction
        fcc = Dense(1024, name='fcc')(resnet_model.layers[-2].output)
        pred = Dense(len(categories), activation='softmax', name='prediction')(fcc)
        my_resnet50 = Model(resnet_model.input, pred)
        for i in range(0, 173):  # only last 4 will be trainable
          my_resnet50.layers[i].trainable=False
        my_resnet50.summary()
```

Confusion matrix results:



```
In [ ]: #import confusion matrix and get scores
        from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix

        y_pred = my_resnet50.predict(X_test)
        y_pred = np.argmax(y_pred, axis = 1)
        _y_test = np.argmax(_y_test, axis = 1)

        print(confusion_matrix(_y_test,y_pred))
        print(classification_report(_y_test,y_pred))
```

```
[[288 188  24]
 [ 77 418   5]
 [208 162 130]]
              precision    recall  f1-score   support

           0       0.50      0.58      0.54       500
           1       0.54      0.84      0.66       500
           2       0.82      0.26      0.39       500

    accuracy                           0.56      1500
   macro avg       0.62      0.56      0.53      1500
weighted avg       0.62      0.56      0.53      1500
```

Model Weight Load

VGG H5 Model Weight Load

In [ ]:

```
#Load Weights
VGG16_H_PATH = ""
VGG16_model = model.load_weights(VGG16_H_PATH)

VGG16_model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=["accuracy"])

VGG16_model.fit(X_train, _y_train, batch_size=32, epochs=10)

# Re-evaluate the model
loss, acc = VGG16_model.evaluate(test_images, test_labels, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```

ResNet50 Model Weight Load ¶

In [ ]:

```
#Load Weights
ResNet50_H_PATH = ""

ResNet_model = model.load_weights(ResNet50_H_PATH)

ResNet_model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=["accuracy"])

ResNet_model.fit(X_train, _y_train, batch_size=32, epochs=10)

# Re-evaluate the model
loss, acc = ResNet_model.evaluate(_y_test, test_labels, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```