

**IMAGE CLASSIFICATION OF**

**CATS AND DOGS**

Project Report

Deep Learning 2019

**K.P. PERAMUNUGAMA –** IT16122024

**BSc. Special (Honors) Degree in Information Technology Specializing**

**Software Engineering**

**Sri Lanka Institute of Information Technology**

**Sri Lanka**

**September 2019**

Table of Contents

[1 INTRODUCTION 2](#_Toc20228850)

[1.1 PROBLEM 2](#_Toc20228851)

[1.2 SOLUTION 2](#_Toc20228852)

[1.3 DATASET OVERVIEW 2](#_Toc20228853)

[1.4 DATASET SOURCE 2](#_Toc20228854)

[2 APPLICATION OF THE APPROPRIATE LEARNING ALGORITHM 3](#_Toc20228855)

[2.1 WHAT IS DEEP LEARNING 3](#_Toc20228856)

[2.1.1 HOW IT WORKS 3](#_Toc20228857)

[2.2 INTRODUCTION AND BACKGROUND OF THE ALGORITHM 3](#_Toc20228858)

[2.2.1 WHY CNN? 3](#_Toc20228859)

[2.2.2 HOW DO HUMANS SEE? 4](#_Toc20228860)

[2.2.3 HOW DO COMPUTERS SEE? 4](#_Toc20228861)

[2.3 CNNS ARCHITECTURE 6](#_Toc20228862)

[2.4 BASIC REQUIREMENTS FOR ALGORITHM 7](#_Toc20228863)

[2.4.1 TENSFLOW BACKEND 7](#_Toc20228864)

[2.4.2 KERAS 7](#_Toc20228865)

[3 ALGORITHM 8](#_Toc20228866)

[3.1 IMPORT LIBRARY 8](#_Toc20228867)

[3.2 DEFINE CONSTANTS 8](#_Toc20228868)

[3.3 PREPARE TRANING DATA 8](#_Toc20228869)

[3.4 DATA-FRAME (HEAD & TAIL ) 9](#_Toc20228870)

[3.5 SEE TOTAL IN COUNT (USING BAR CHART) 9](#_Toc20228871)

[3.6 THE SAMPLE IMAGE 10](#_Toc20228872)

[3.7 BUILD MODEL 10](#_Toc20228873)

[3.8 CALLBACKS FUNCTION 11](#_Toc20228874)

[3.9 PREPARE DATASET 12](#_Toc20228875)

[3.10 TRANING GENERATOR 13](#_Toc20228876)

[3.11 VALIDATION GENERATOR 13](#_Toc20228877)

[4 RESULTS 14](#_Toc20228878)

[4.1 SEE HOW THE GENERATOR WORK 14](#_Toc20228879)

[4.2 FIT MODEL 15](#_Toc20228880)

[4.3 SAVE MODEL 16](#_Toc20228881)

[4.4 VIRTUALIZE TRAINING 16](#_Toc20228882)

[4.5 VIRTAULIZE RESULT 18](#_Toc20228883)

[4.6 FINAL PREDICTED RESULT WITH IMAGES 19](#_Toc20228884)

[5 SUBMISSION 21](#_Toc20228885)

[6 APPENDIX 22](#_Toc20228886)

[7 REFERENCES 29](#_Toc20228887)

# INTRODUCTION

The ultimate goal of this project is to form a system that may notice cats and dogs. while our goal is extremely specific (Cats vs Dogs), Image Classifier will notice something that's tangible with an adequate dataset.

The training archive contains 25,000 pictures of dogs and cats. Train this formula on these files and predict the labels for test1.zip (1 = dog, 0 = cat).

## PROBLEM

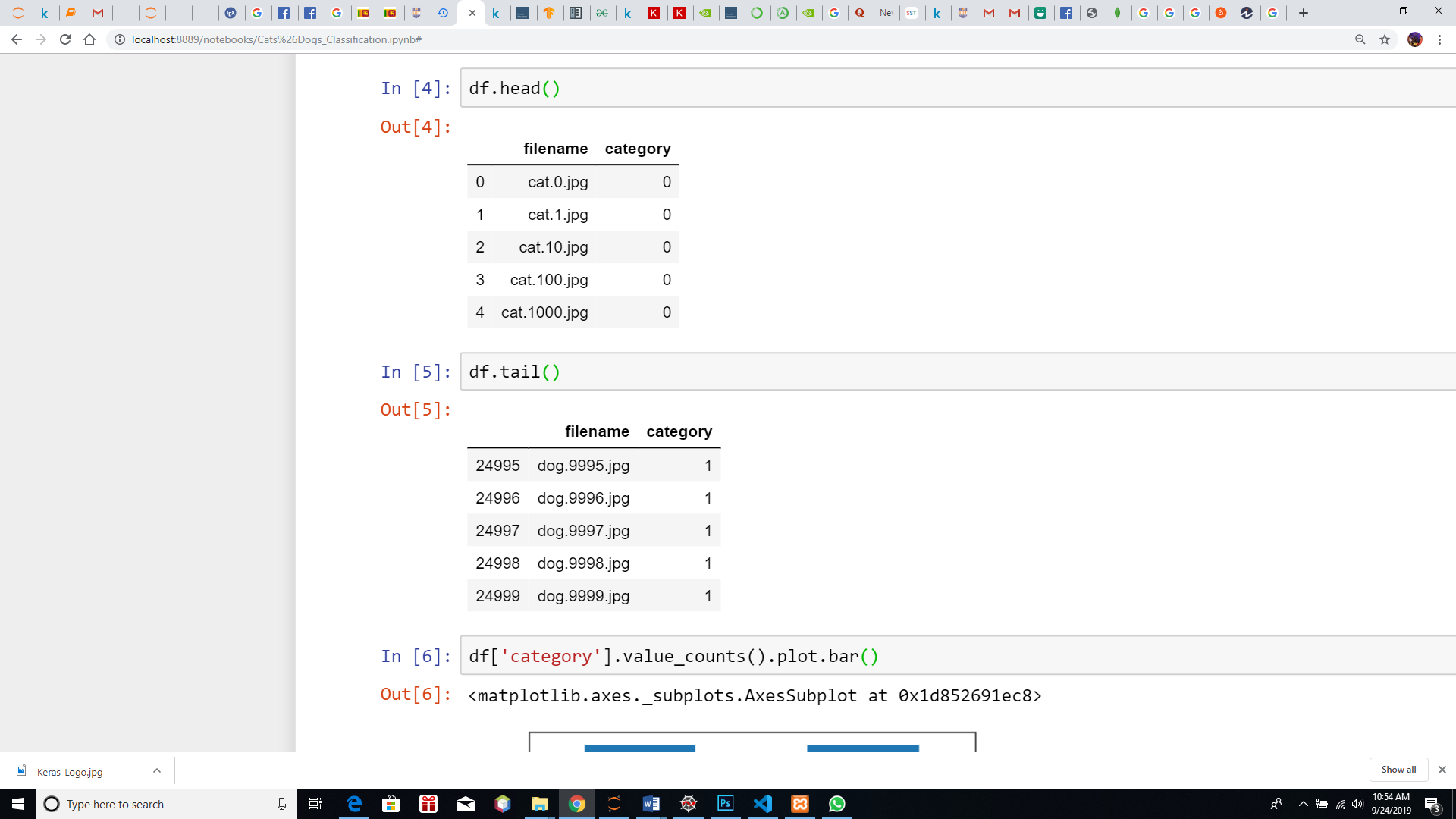
We need to distinguish dogs from cats in large number of dataset. When we put set of images of cats & dogs, we need to identify cats and dogs and labeled it.

## SOLUTION

In this case we have to use deep learning algorithm. Because it’s a neural learning and it has artificial intelligence (AI) therefore we need to use deep learning algorithm to solve this. I choose CNN algorithm to solve this. Because, for these image classification and detection purposes we need to use CNN deep learning algorithm.

## DATASET OVERVIEW

There are 25 000 images of cats and dogs in the train dataset. Labeled as (1 = dog, 0 = cat).



## DATASET SOURCE

Source: Kaggle

Name: Dogs vs. Cats

URL: <https://www.kaggle.com/c/dogs-vs-cats/data>

# APPLICATION OF THE APPROPRIATE LEARNING ALGORITHM

## WHAT IS DEEP LEARNING

Deep learning is an artificial intelligence perform that imitates the workings of the human brain in process information and making patterns to be used in the higher cognitive processes. Deep learning could be a set of machine learning in artificial intelligence (AI) that has networks capable of learning unattended from information that's unstructured or unlabeled. conjointly called deep neural learning or deep neural network.

### HOW IT WORKS

Deep learning has evolved hand-in-hand with the digital era, which has led to Associate in Nursing explosion of knowledge all told forms and from each region of the planet. This data, known merely as huge information, is drawn from sources like social media, web search engines, e-commerce platforms, and on-line cinemas, among others. This monumental quantity of information is quickly accessible and may be shared through fintech applications like cloud computing.

However, the data, that ordinarily is unstructured, is therefore large that it might take decades for humans to grasp it and extract relevant info. firms notice the unimaginable potential that may result from unraveling this wealth of knowledge and are progressively adapting to AI systems for machine-driven support.

## INTRODUCTION AND BACKGROUND OF THE ALGORITHM

CNNs area unit deep learning models suited to analyzing visual imaging. they're heavily influenced by however we have a tendency to - humans, see the encompassing world. Before we have a tendency to proceed to CNNs in order to analyze however computers see, let’s specialize in however humans are able to do it.

### WHY CNN?

There are plenty of algorithms that folks used for image classification before CNN became widespread. people used to produce features from pictures so feed those features into some classification algorithmic rule like SVM. Some algorithmic rule also used the pixel level values of pictures as a feature vector too. to offer an Associate in Nursing example, you may train a SVM with 784 options wherever every feature is that the element price for a 28x28 image.

So why CNN and why they work such a lot better?

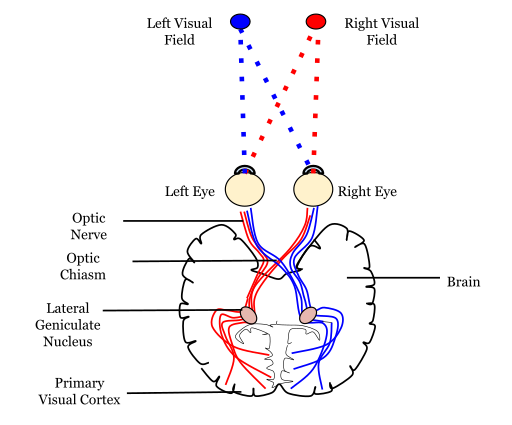
CNNs are often thought of as automatic feature extractors from the image. whereas if I take advantage of an algorithmic rule with element vector I lose plenty of spatial interaction between pixels, CNN effectively uses adjacent element info to effectively down sample the image initial by convolution so uses a prediction layer at the top.

This concept was initially given by Yann le cun autoimmune disorder in 1998 for digit classification wherever he used one convolution layer. it had been later popularized by Alex net in 2012 that used multiple convolution layers to attain state of the art on ImageNet. so creating them Associate in Nursing algorithmic rule of alternative for image classification challenges henceforth.

Convolution Neural Networks (CNN) are special variety of Feed-Forward Artificial Neural Networks that are usually used for image detection tasks. It accepts a giant array of pixels as input to the network. The result's a matrix referred to as the Convolved Feature Map.

In this problem we have to create an algorithm to distinguish dogs from cats. Therefore, we need to classify the images of our dataset. For these image classification and detection purposes we need to use CNN deep learning algorithm.

### HOW DO HUMANS SEE?



When we see an object, the light receptors in your eyes send signals via the nervus opticus to the first visual Cortex, wherever the input is being processed. the first visual Cortex is sensible of what the eye sees.

The deeply advanced hierarchical data structure of neurons and connections within the brain plays a serious role during this method of remembering and labeling objects.

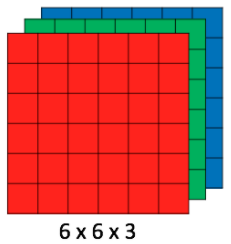
Without going into any details, the human brain analyzes pictures within the layers of accelerating quality. the primary layer distinguishes basic attributes like lines and curves. At higher levels, the brain acknowledges that a mix of edges and colors is, for instance, a train or a dog.

Individual animal tissue neurons reply to stimuli solely in an exceedingly restricted region of the field of vision referred to as the receptive field. The receptive fields of various neurons partly overlap such they cowl the whole field of vision.

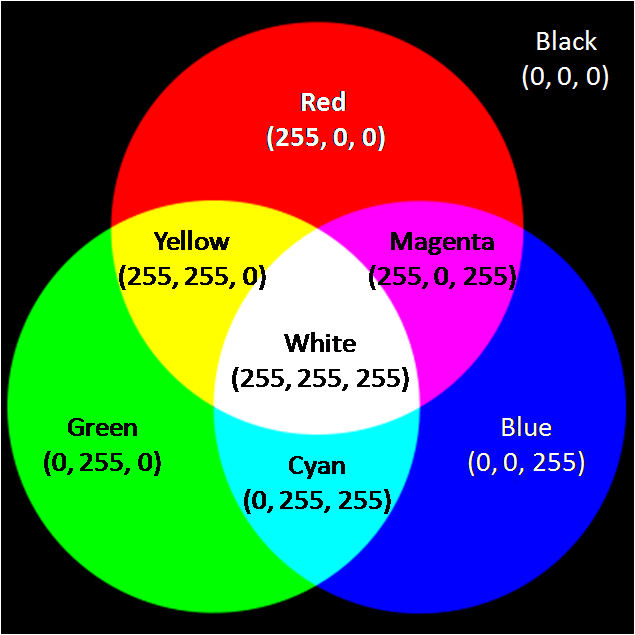
### HOW DO COMPUTERS SEE?

For a computer, an image is just an array of values. Typically, it’s a 3-dimensional (RGB) matrix of pixel values.

For example, a 6x6 RGB abstract image representation would look like this.



Where each pixel has a specific value of red, green and blue that represents the color of a given pixel.



CNN processes pictures using matrixes of weights known as filters (features) that find specific attributes like vertical edges, horizontal edges, etc. Moreover, as the image progresses through every layer, the filters are able to recognize a lot of advanced attributes. Final goal of CNN is to find what's happening within the scene.

## CNNS ARCHITECTURE

We’ve established before that similarly to humans, computers are able to analyze visual imagery using deeply layered systems.

* INPUT

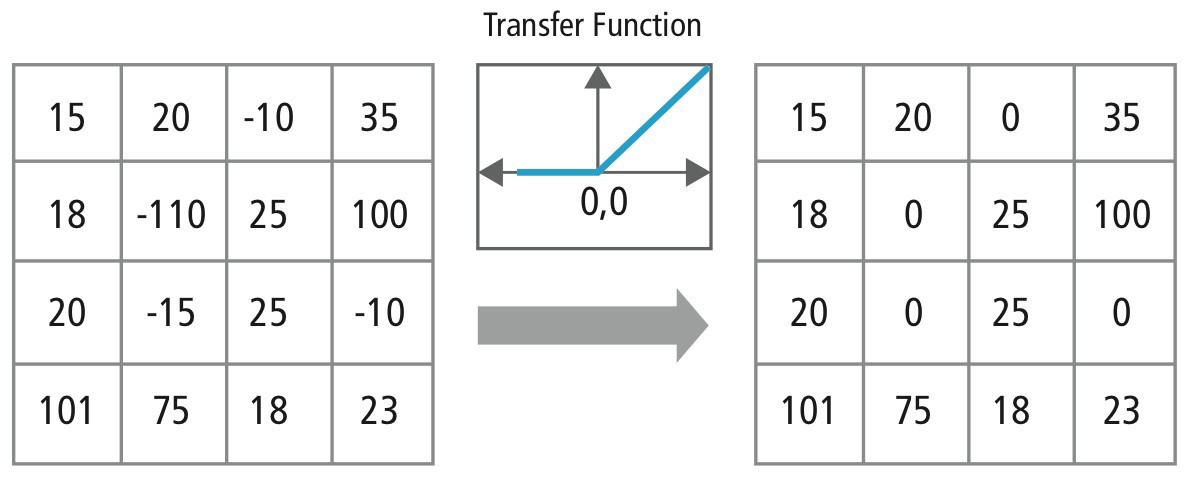
A Matrix of pixel values in the shape of [WIDTH, HEIGHT, CHANNELS].

* CONVOLUTION

The ultimate purpose of this layer is to receive a **feature map**. Usually, we start with low number of filters for low-level feature detection. The deeper we go into the CNN, the more filters (usually they are also smaller) we use to detect high-level features.

* ACTIVATION

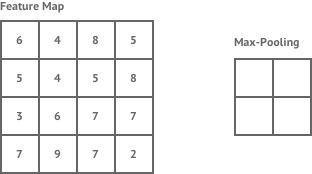
Without going into further details, we will use ReLU activation function that returns 0 for every negative value in the input image while it returns the same value for every positive value.



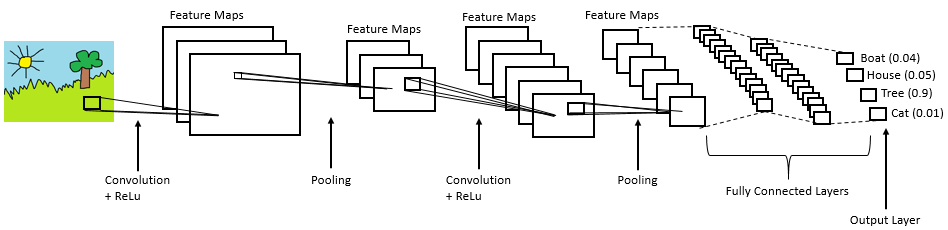
* POOLING

The goal of this layer is to supply spatial variance, that merely means the system is capable of recognizing AN object as an object even once its look varies in how.

The pooling layer can perform a downsampling operation on the spatial dimensions (width, height), leading to output like [16x16x12] for pooling\_size=(2, 2).



* FULLY CONNECTED



## BASIC REQUIREMENTS FOR ALGORITHM

### TENSFLOW BACKEND

TensorFlow may be a free and open-source software system library for dataflow and differentiable programming across a spread of tasks. it's a symbolic science library and is additionally used for machine learning applications like neural networks.

### KERAS

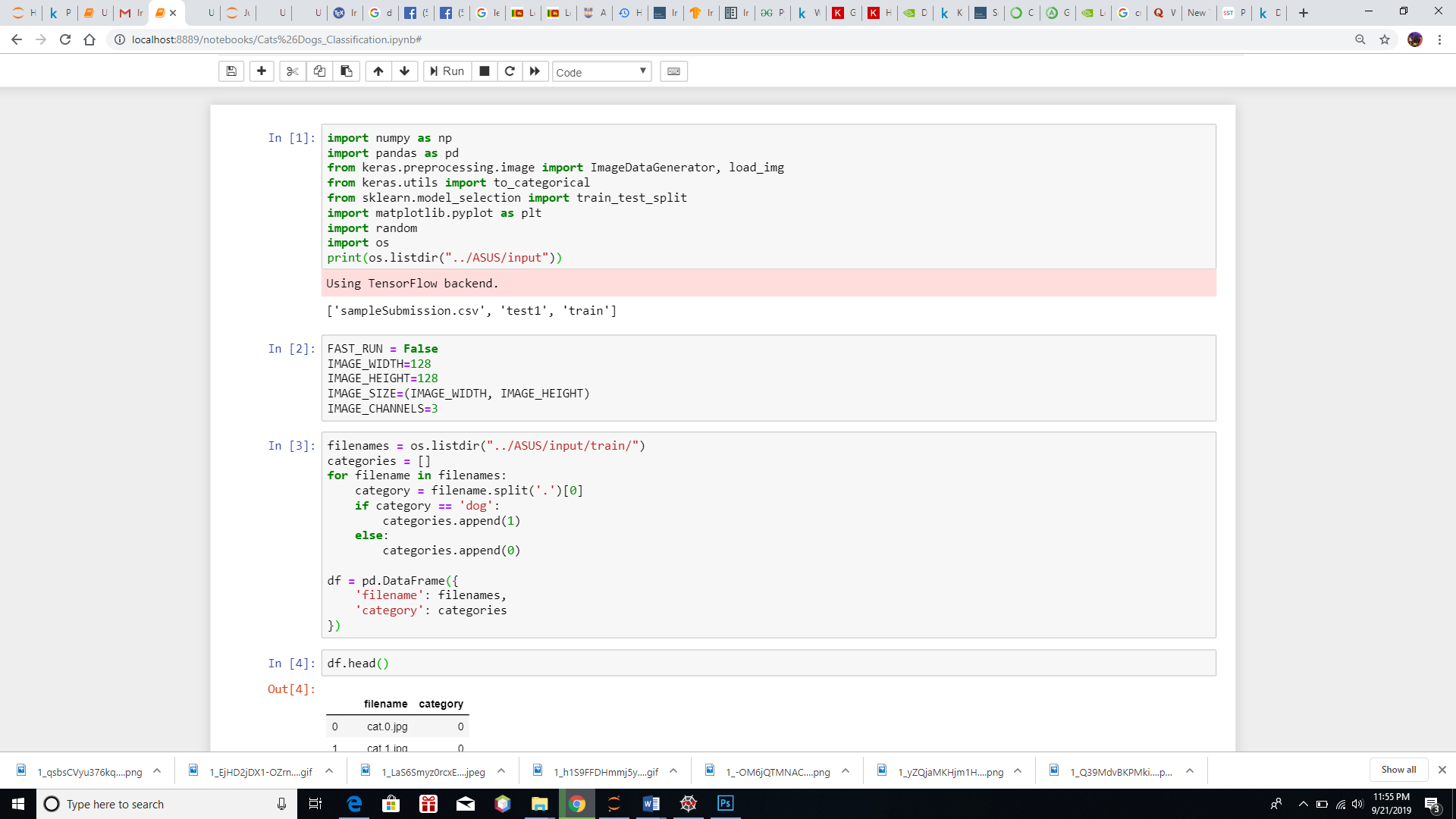
Keras is an Open-source Neural Network library written in Python that runs on prime of Theano or Tensorflow. it's designed to be standard, quick and straightforward to use. it absolutely was developed by François Chollet, a Google engineer.

Keras does not handle low-level computation. Instead, it uses another library to try and do it, referred to as the "Backend. thus Keras is a high-level API wrapper for the low-level API, capable of running on prime of TensorFlow, CNTK, or Theano.

Keras High-Level API handles the approach we tend to create models, process layers, or established multiple input-output models. during this level, Keras additionally compiles our model with loss and optimizer functions, training method with work operate. Keras does not handle Low-Level API like creating the computational graph, creating tensors or different variables as a result of it's been handled by the "backend" engine.

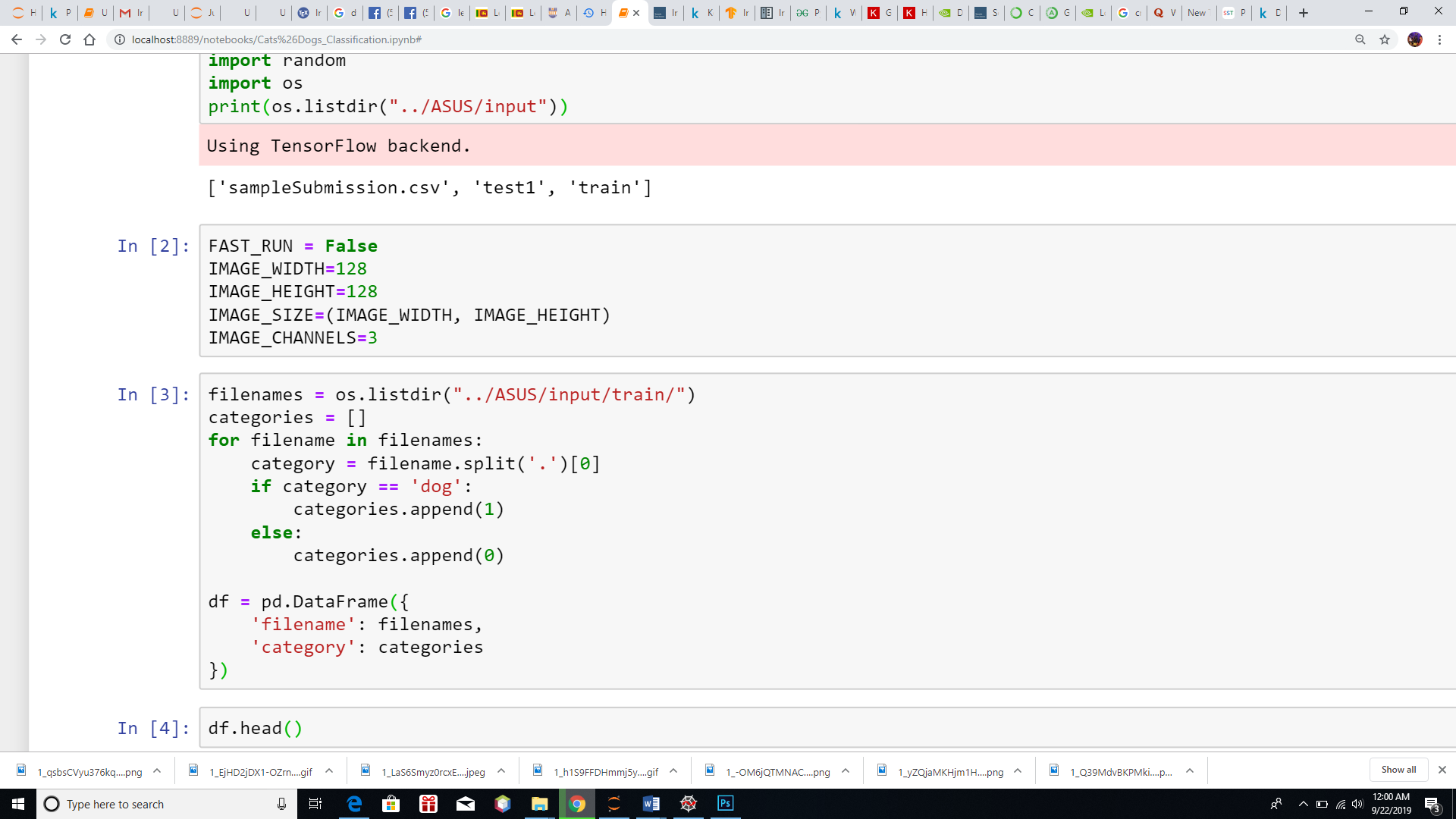
# ALGORITHM

## IMPORT LIBRARY

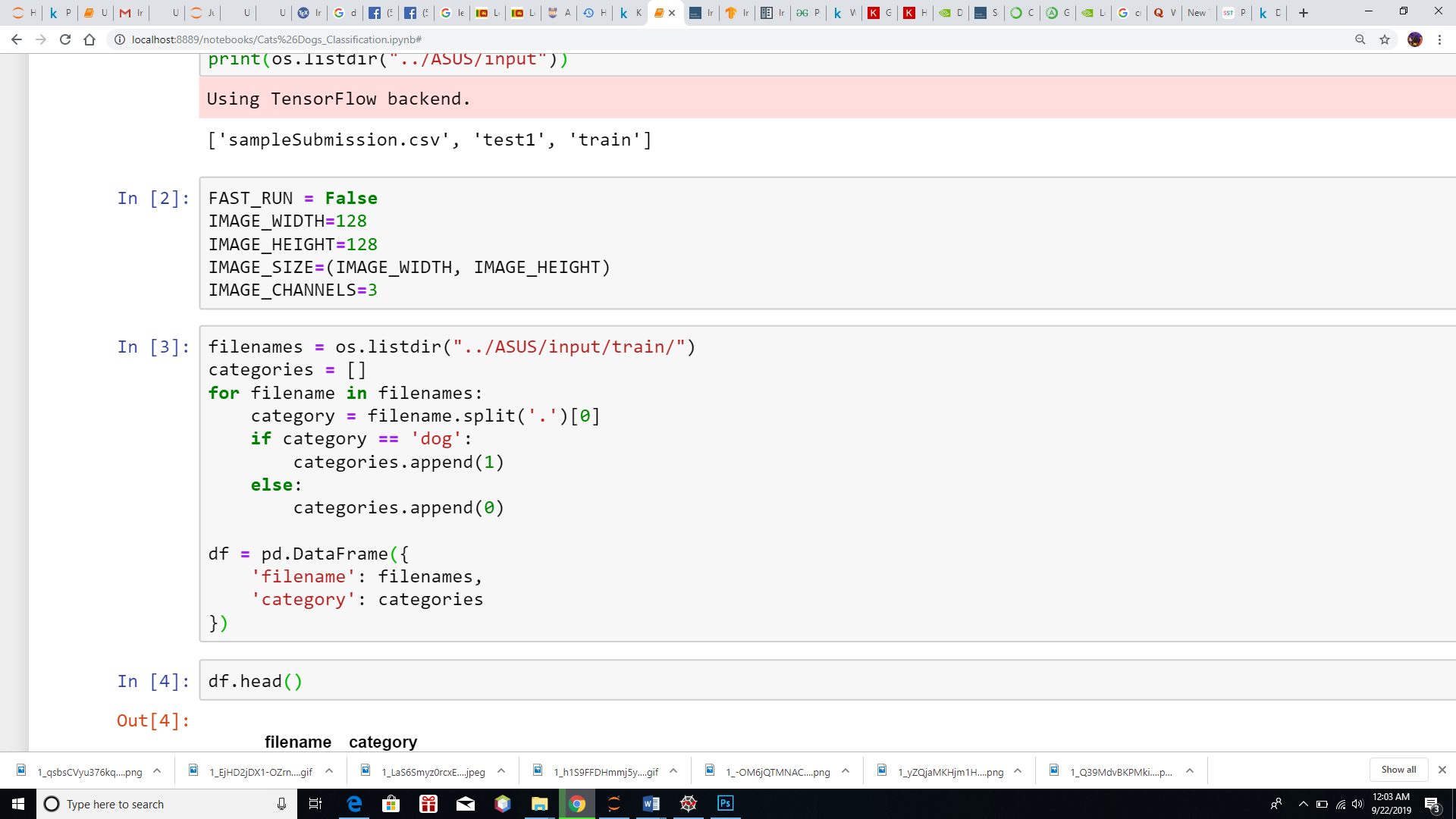


First of all, we must import all of the above libraries

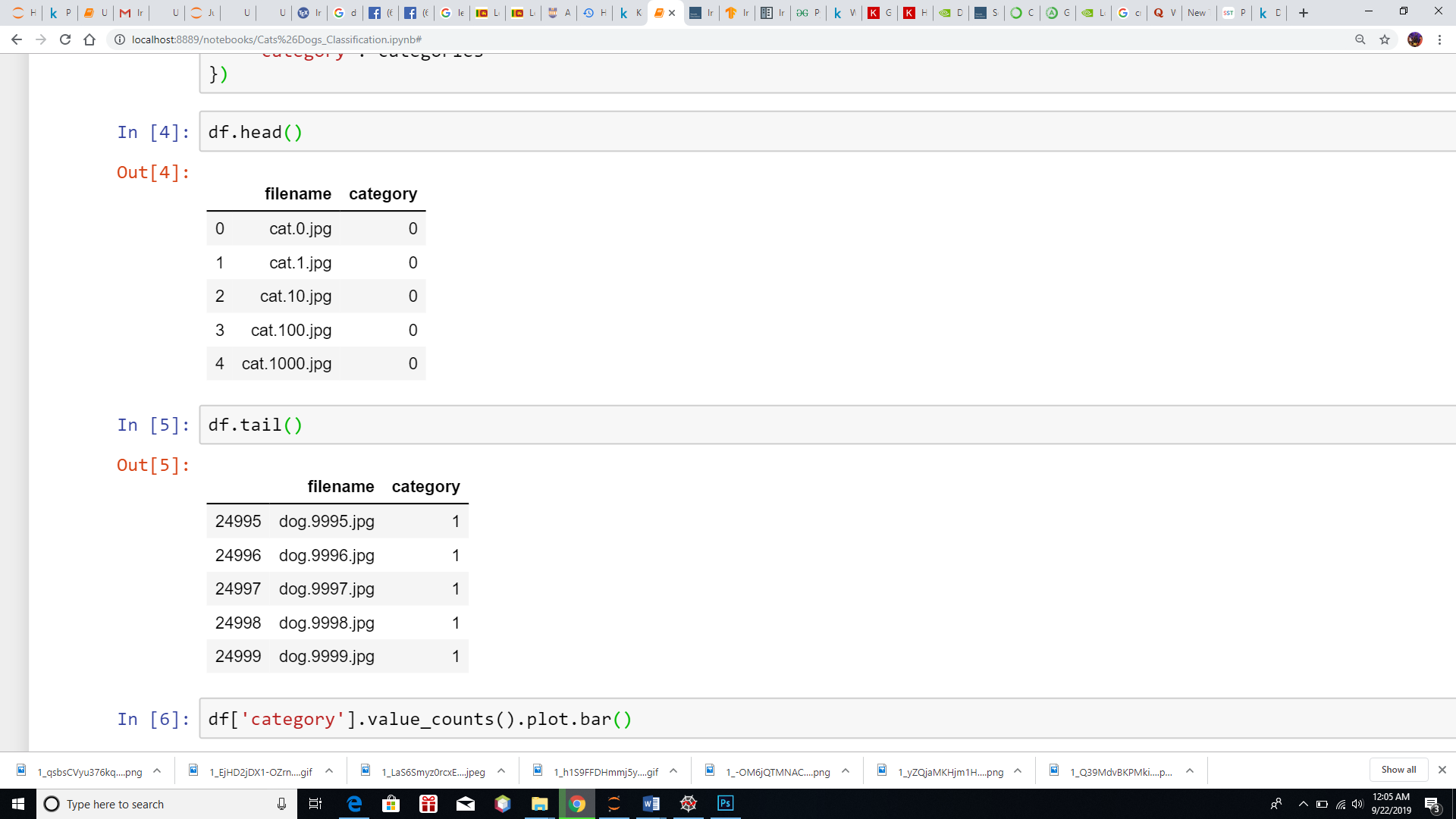
## DEFINE CONSTANTS



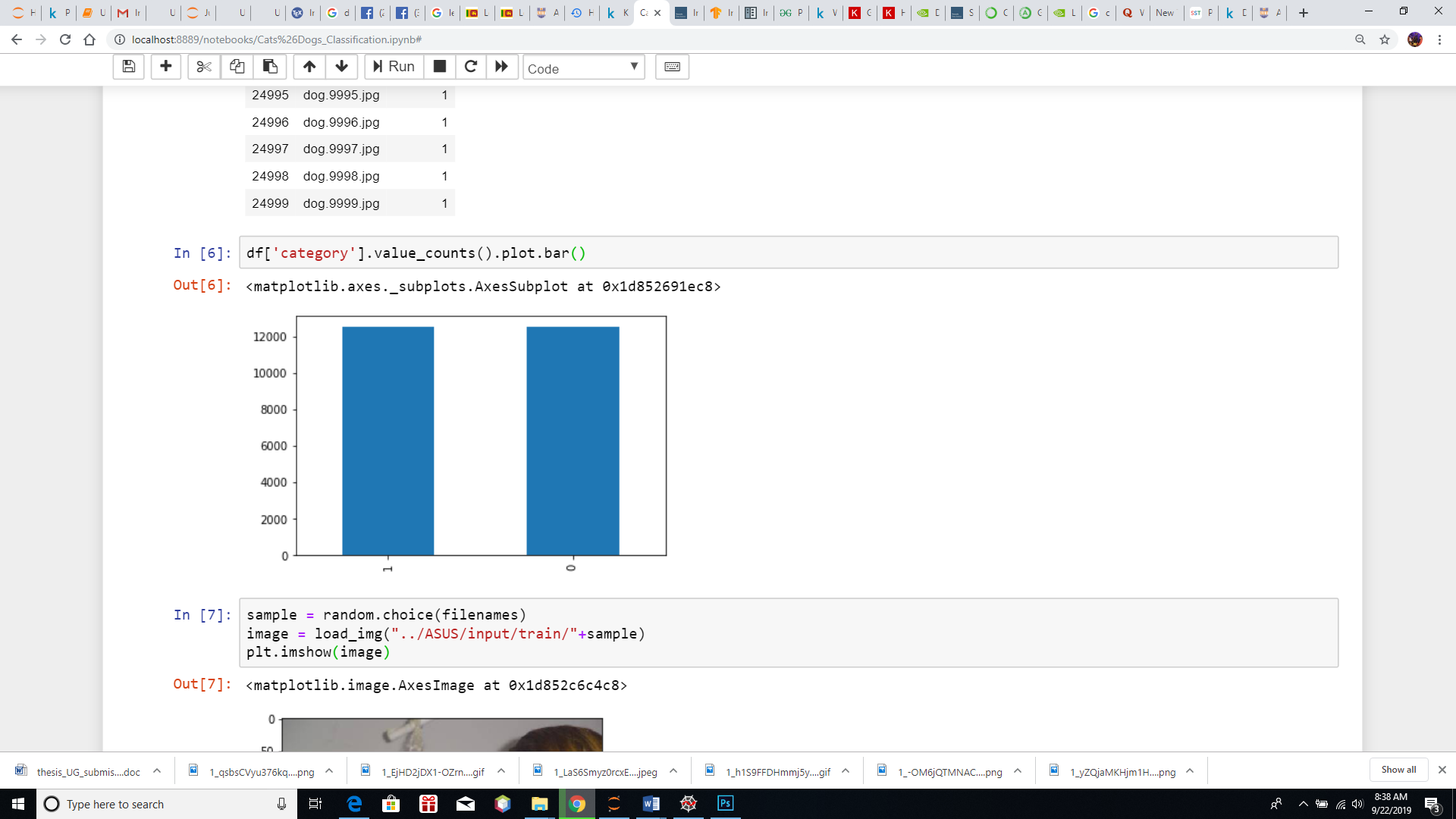
## PREPARE TRANING DATA



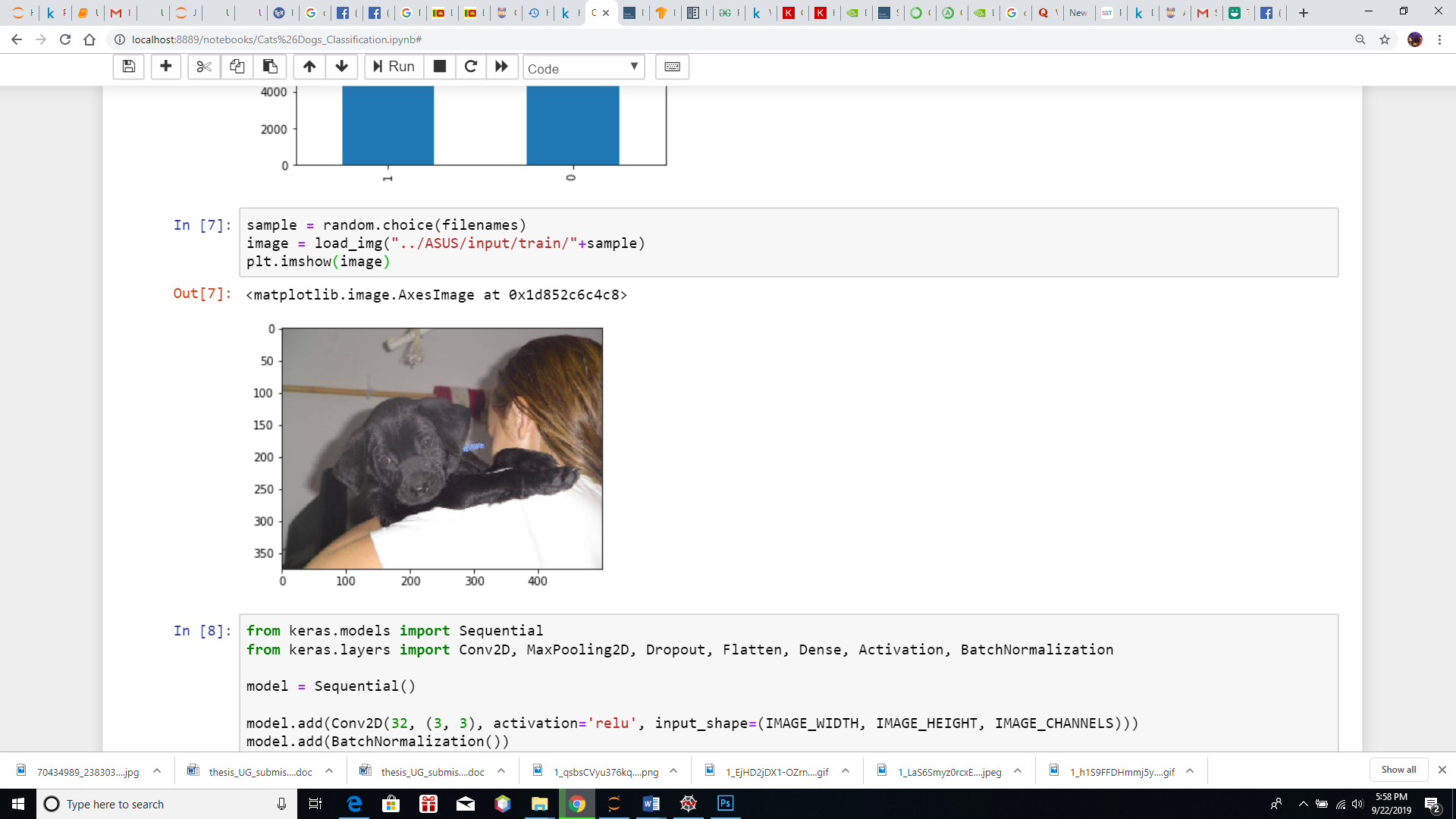
## DATA-FRAME (HEAD & TAIL )



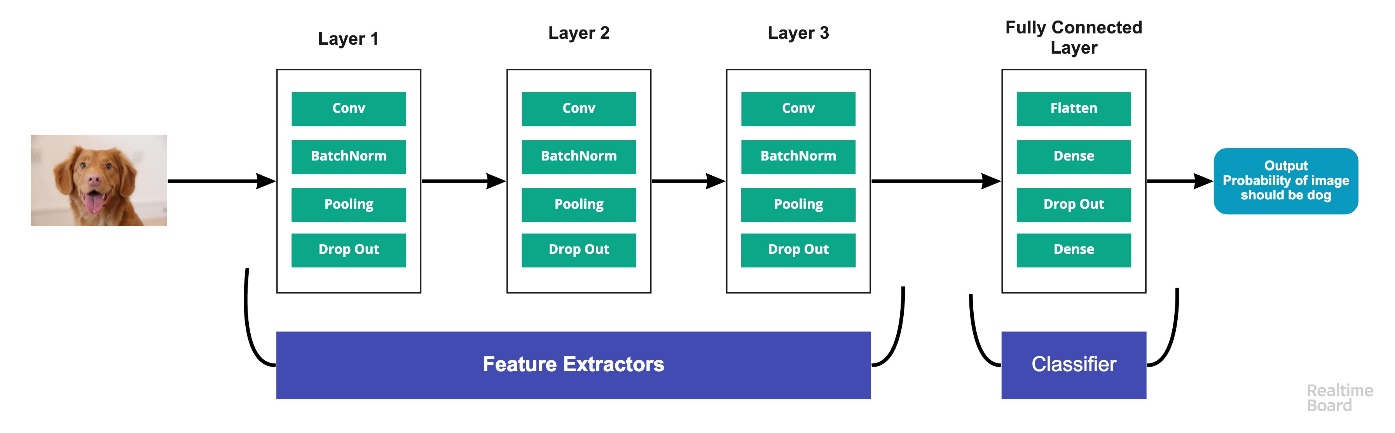
## SEE TOTAL IN COUNT (USING BAR CHART)



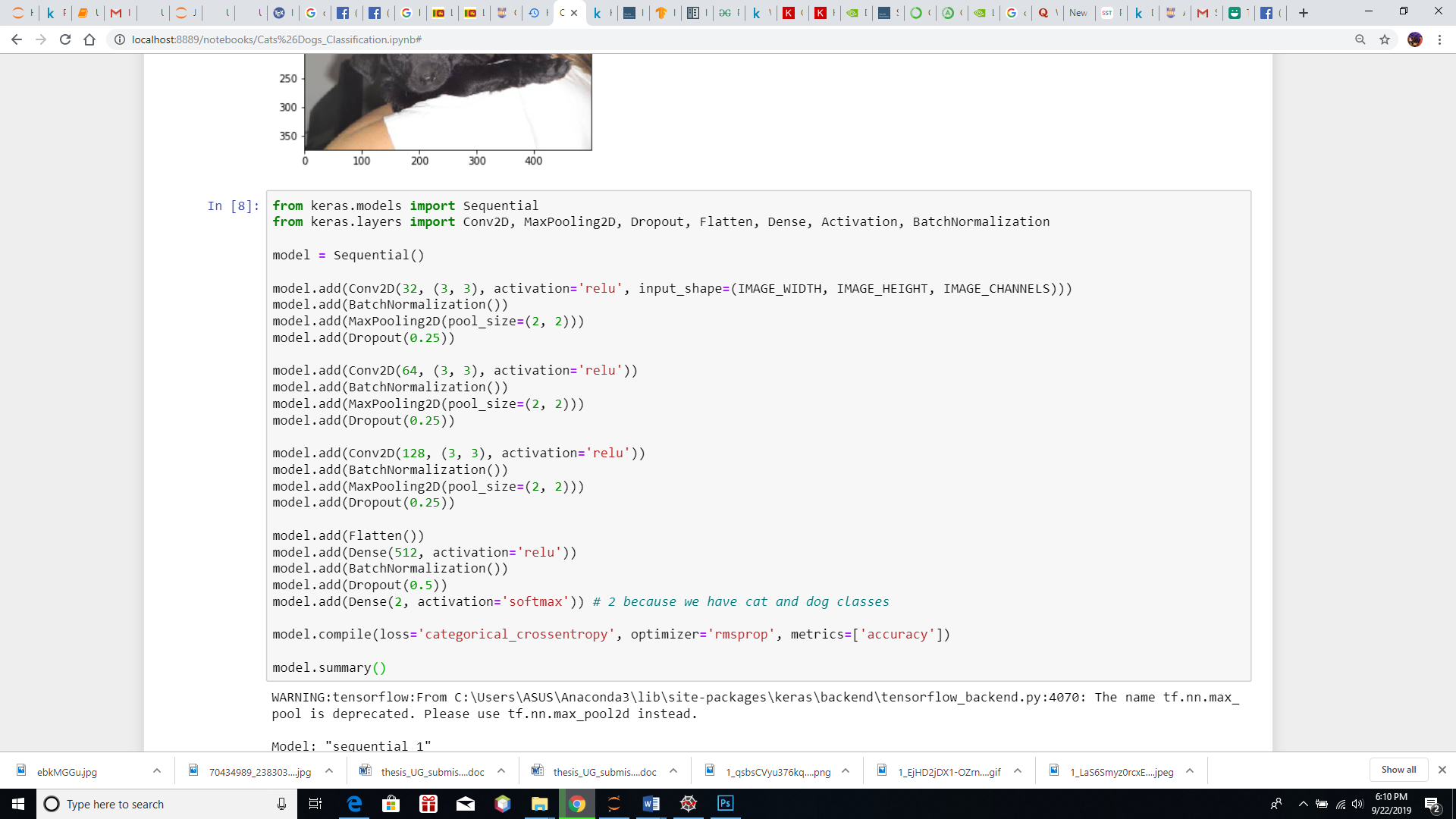
## THE SAMPLE IMAGE

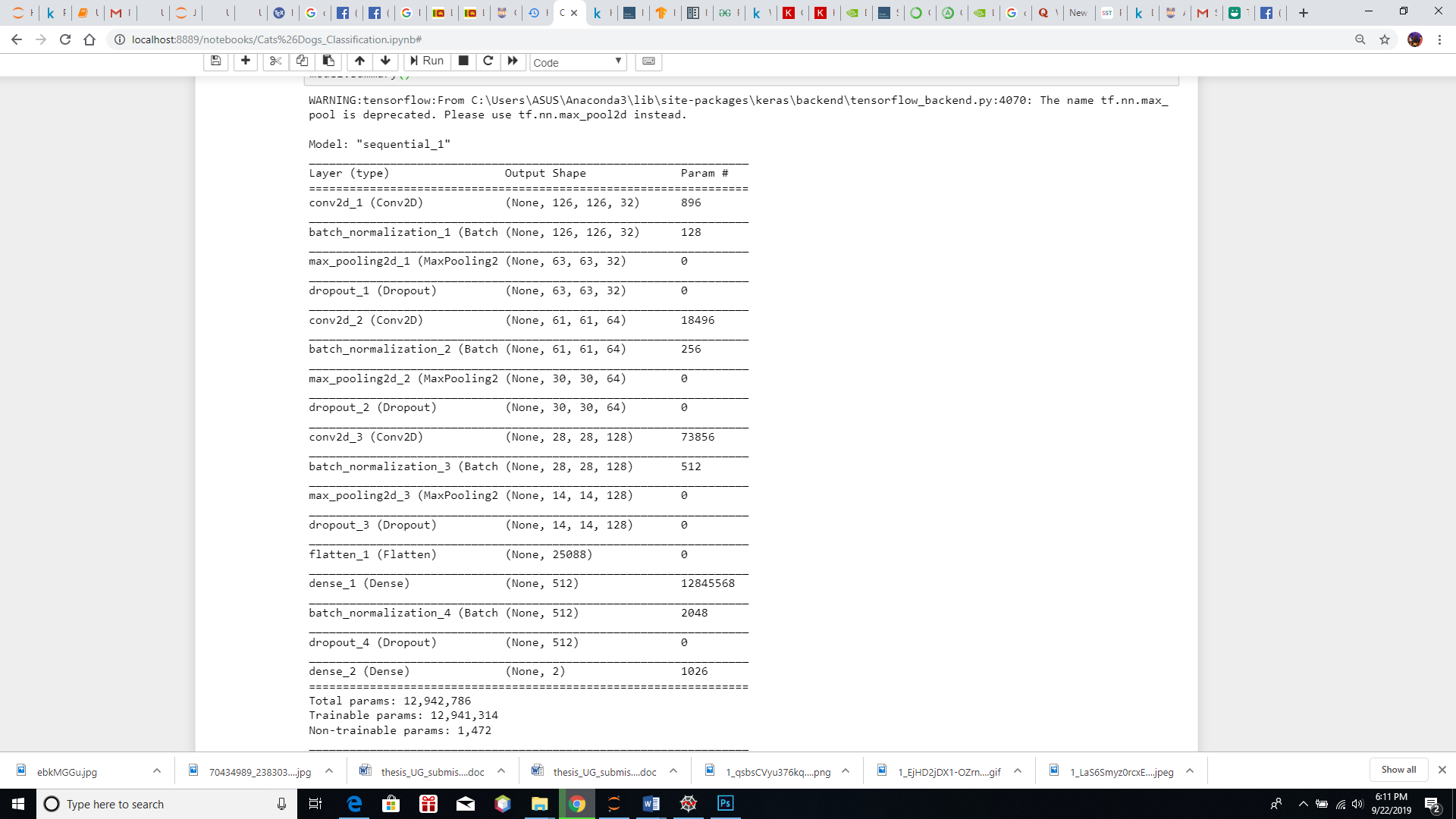


## BUILD MODEL



* **Input Layer**: It represent input image data. It will reshape image into single dimension array. Example your image is 64x64 = 4096, it will convert to (4096,1) array.
* **Conv Layer**: This layer will extract features from image.
* **Pooling Layer**: This layer reduces the spatial volume of input image after convolution.
* **ully Connected Layer**: It connect the network from a layer to another layer
* **Output Layer**: It is the predicted values layer.

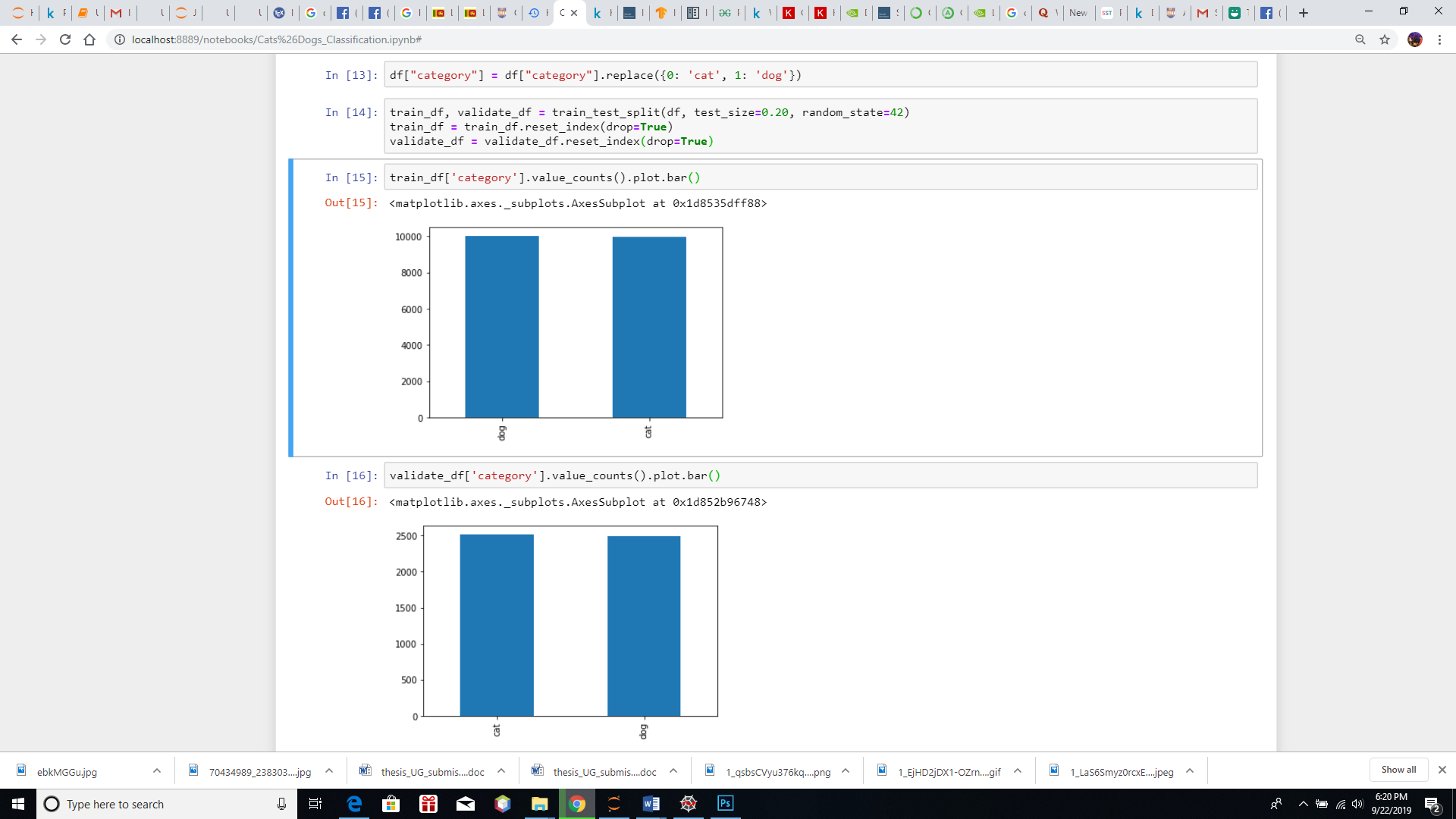




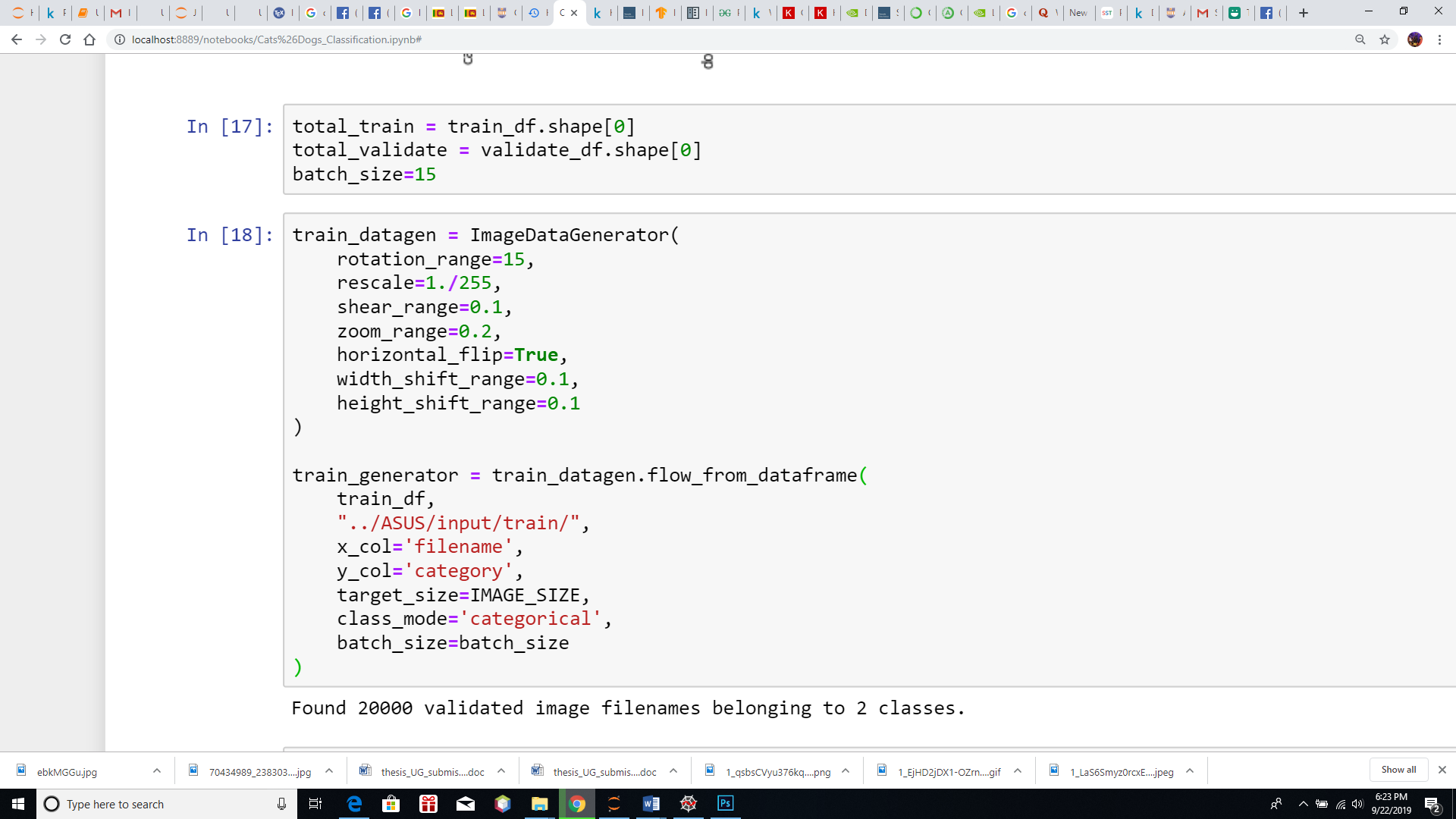
## CALLBACKS FUNCTION

## PREPARE DATASET

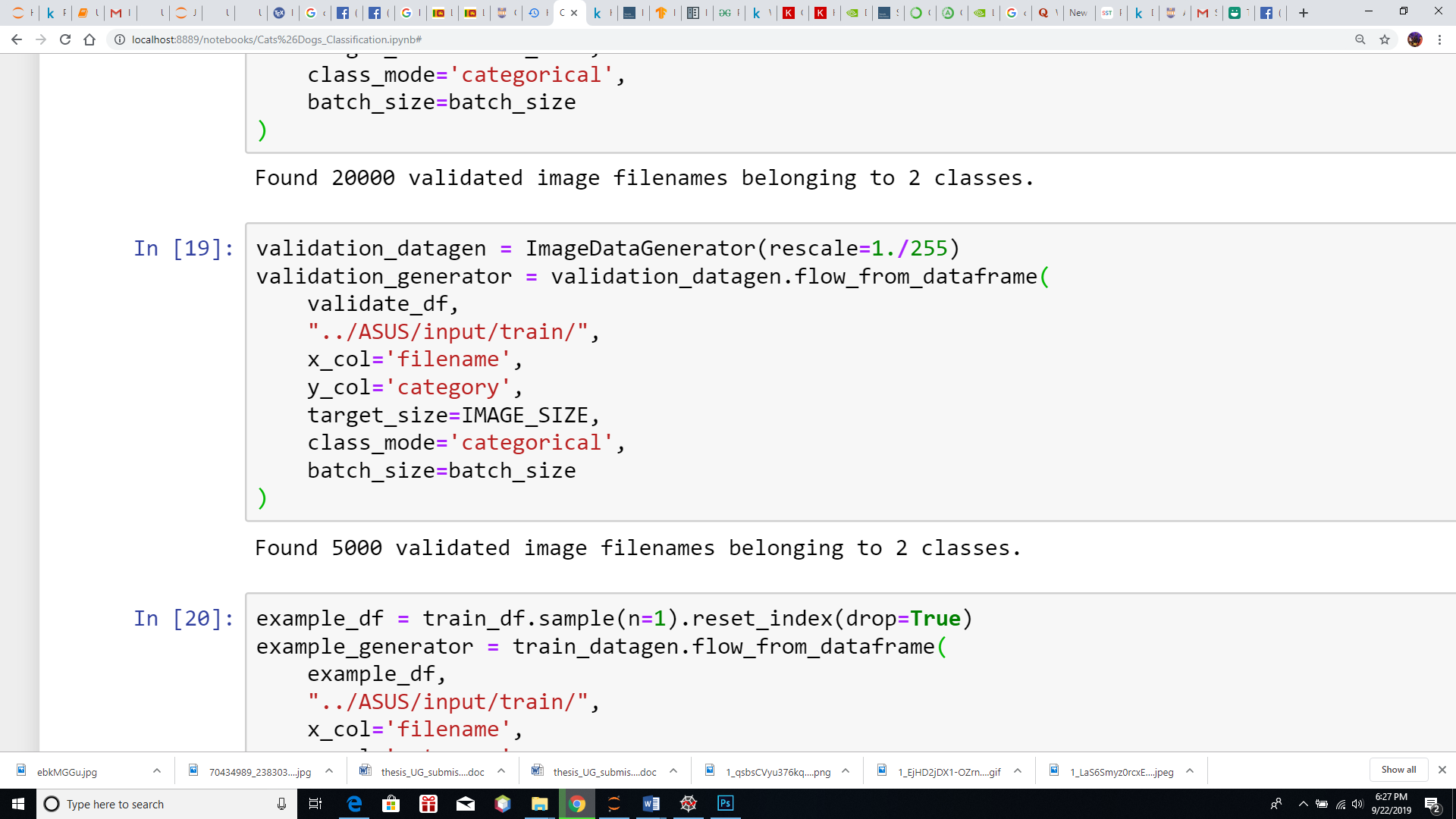
Because we are going to use an image generator with class\_mode="categorical". we want to convert column class into a string. Then image generator can convert it one-hot encoding that is nice for our classification. So we are going to convert 1 to dog and 0 to cat.



## TRANING GENERATOR

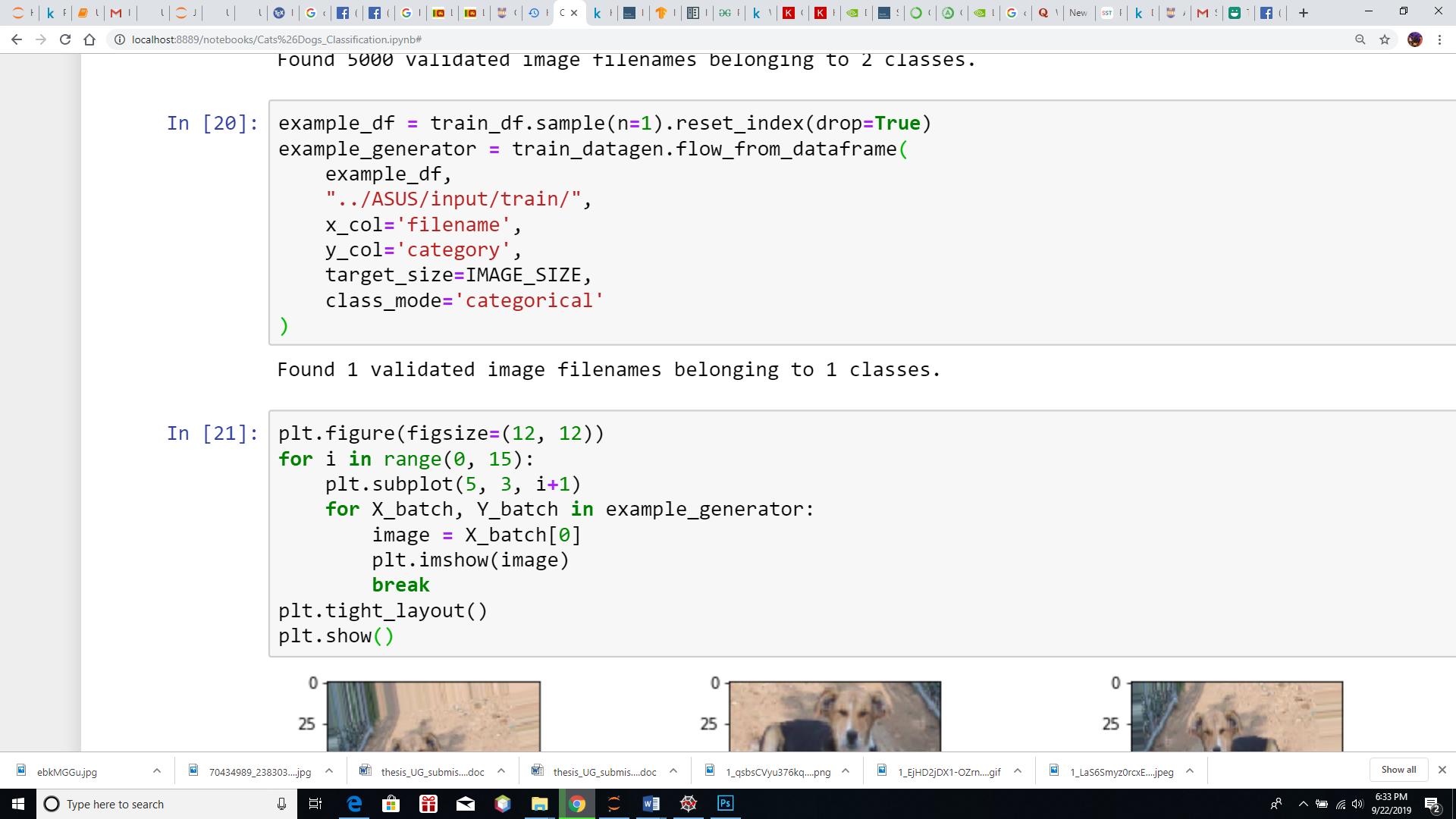


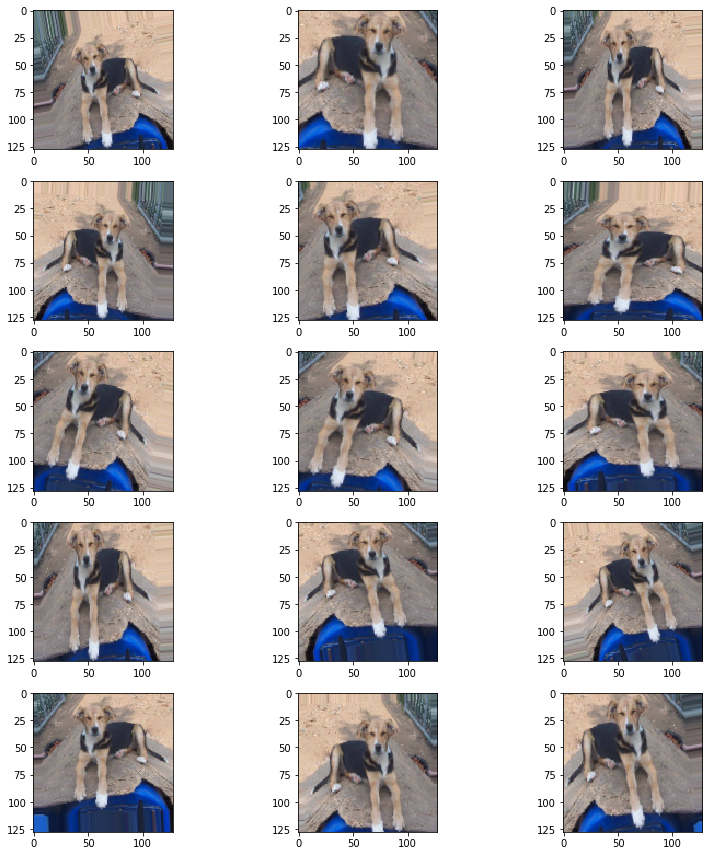
## VALIDATION GENERATOR



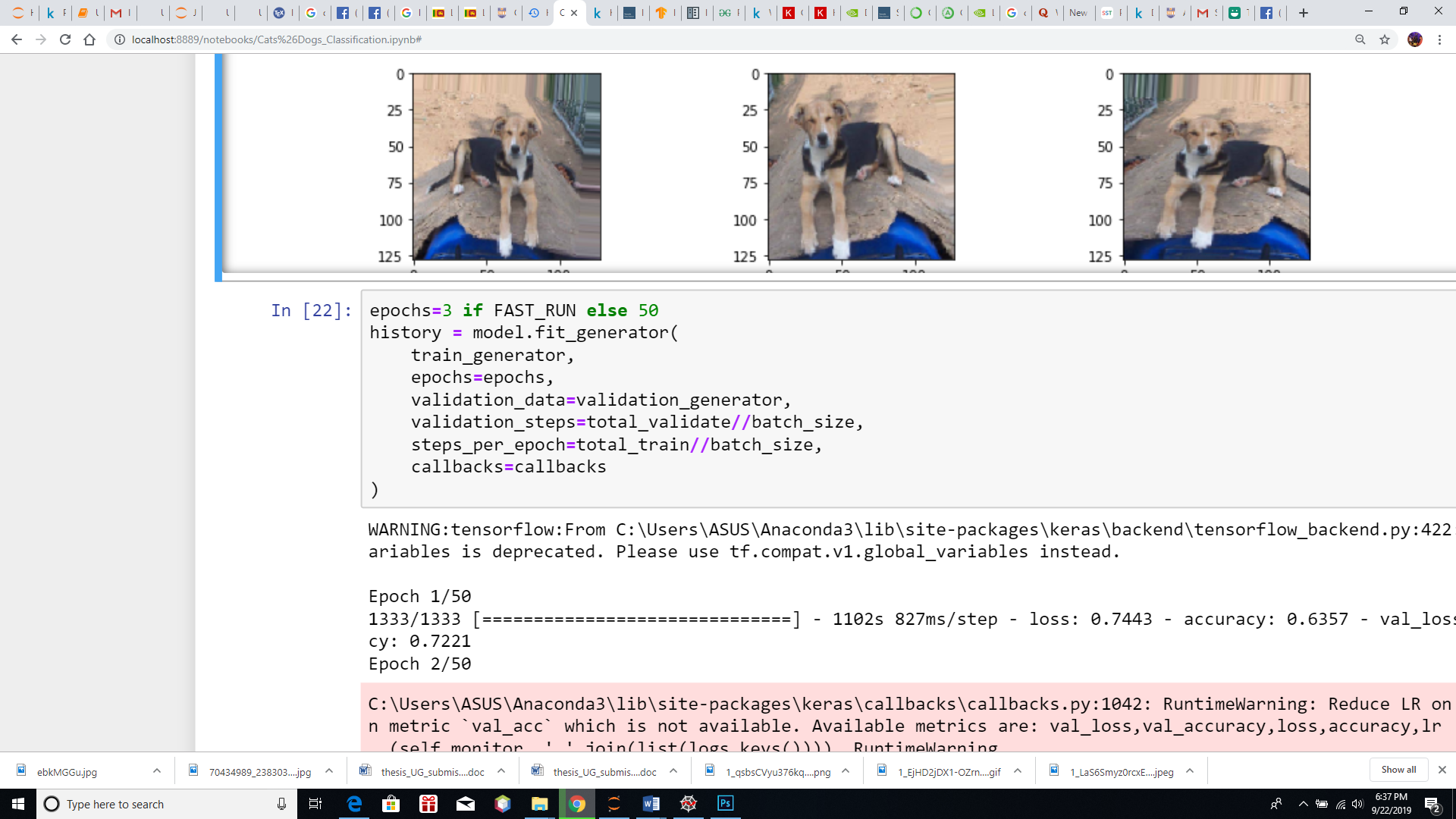
# RESULTS

## SEE HOW THE GENERATOR WORK





## FIT MODEL



WARNING:tensorflow:From C:\\Users\\ASUS\\Anaconda3\\lib\\site-packages\\keras\\backend\\tensorflow\_backend.py:422: The name tf.global\_variables is deprecated. Please use tf.compat.v1.global\_variables instead.\n\nEpoch 1/50\n1333/1333 [==============================] - 1102s 827ms/step - loss: 0.7443 - accuracy: 0.6357 - val\_loss: 0.8280 - val\_accuracy: 0.7221\nEpoch 2/50\n","name":"stdout"},{"output\_type":"stream","text":"C:\\Users\\ASUS\\Anaconda3\\lib\\site-packages\\keras\\callbacks\\callbacks.py:1042: RuntimeWarning: Reduce LR on plateau conditioned on metric `val\_acc` which is not available. Available metrics are: val\_loss,val\_accuracy,loss,accuracy,lr\n (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning\n","name":"stderr"},{"output\_type":"stream","text":"1333/1333 [==============================] - 1113s 835ms/step - loss: 0.5509 - accuracy: 0.7275 - val\_loss: 0.2706 - val\_accuracy: 0.7655\nEpoch 3/50\n1333/1333 [==============================] - 1101s 826ms/step - loss: 0.5022 - accuracy: 0.7631 - val\_loss: 0.4750 - val\_accuracy: 0.7803\nEpoch 4/50\n1333/1333 [==============================] - 1100s 825ms/step - loss: 0.4722 - accuracy: 0.7831 - val\_loss: 0.6564 - val\_accuracy: 0.7773\nEpoch 5/50\n1333/1333 [==============================] - 1098s 823ms/step - loss: 0.4439 - accuracy: 0.7985 - val\_loss: 0.2684 - val\_accuracy: 0.8455\nEpoch 6/50\n1333/1333 [==============================] - 1091s 819ms/step - loss: 0.4221 - accuracy: 0.8094 - val\_loss: 0.3970 - val\_accuracy: 0.8058\nEpoch 7/50\n1333/1333 [==============================] - 1091s 818ms/step - loss: 0.4075 - accuracy: 0.8189 - val\_loss: 0.3570 - val\_accuracy: 0.7751\nEpoch 8/50\n1333/1333 [==============================] - 1090s 818ms/step - loss: 0.3850 - accuracy: 0.8303 - val\_loss: 0.3029 - val\_accuracy: 0.7809\nEpoch 9/50\n1333/1333 [==============================] - 1090s 817ms/step - loss: 0.3748 - accuracy: 0.8380 - val\_loss: 0.1707 - val\_accuracy: 0.8381\nEpoch 10/50\n1333/1333 [==============================] - 1090s 818ms/step - loss: 0.3671 - accuracy: 0.8410 - val\_loss: 0.3872 - val\_accuracy: 0.8558\nEpoch 11/50\n1333/1333 [==============================] - 1094s 820ms/step - loss: 0.3579 - accuracy: 0.8439 - val\_loss: 0.3092 - val\_accuracy: 0.8217\nEpoch 12/50\n1333/1333 [==============================] - 1091s 818ms/step - loss: 0.3485 - accuracy: 0.8502 - val\_loss: 0.3228 - val\_accuracy: 0.8859\nEpoch 13/50\n1333/1333 [==============================] - 1095s 821ms/step - loss: 0.3475 - accuracy: 0.8521 - val\_loss: 0.4675 - val\_accuracy: 0.8491\nEpoch 14/50\n1333/1333 [==============================] - 1088s 816ms/step - loss: 0.3403 - accuracy: 0.8536 - val\_loss: 0.3442 - val\_accuracy: 0.8086\nEpoch 15/50\n1333/1333 [==============================] - 1086s 815ms/step - loss: 0.3318 - accuracy: 0.8578 - val\_loss: 0.2731 - val\_accuracy: 0.8929\nEpoch 16/50\n1333/1333 [==============================] - 1087s 815ms/step - loss: 0.3296 - accuracy: 0.8602 - val\_loss: 0.1314 - val\_accuracy: 0.8650\nEpoch 17/50\n1333/1333 [==============================] - 1089s 817ms/step - loss: 0.3265 - accuracy: 0.8636 - val\_loss: 0.1971 - val\_accuracy: 0.8949\nEpoch 18/50\n1333/1333 [==============================] - 1086s 815ms/step - loss: 0.3249 - accuracy: 0.8614 - val\_loss: 0.2574 - val\_accuracy: 0.8483\nEpoch 19/50\n1333/1333 [==============================] - 1087s 816ms/step - loss: 0.3226 - accuracy: 0.8630 - val\_loss: 0.1106 - val\_accuracy: 0.8786\nEpoch 20/50\n1333/1333 [==============================] - 1086s 815ms/step - loss: 0.3129 - accuracy: 0.8675 - val\_loss: 0.1447 - val\_accuracy: 0.8818\nEpoch 21/50\n1333/1333 [==============================] - 1085s 814ms/step - loss: 0.3133 - accuracy: 0.8678 - val\_loss: 0.4479 - val\_accuracy: 0.8800\nEpoch 22/50\n1333/1333 [==============================] - 1086s 815ms/step - loss: 0.3108 - accuracy: 0.8661 - val\_loss: 0.2316 - val\_accuracy: 0.8804\nEpoch 23/50\n1333/1333 [==============================] - 1088s 816ms/step - loss: 0.3072 - accuracy: 0.8707 - val\_loss: 0.1633 - val\_accuracy: 0.8881\nEpoch 24/50\n1333/1333 [==============================] - 1123s 843ms/step - loss: 0.3005 - accuracy: 0.8730 - val\_loss: 0.2487 - val\_accuracy: 0.8959\nEpoch 25/50\n1333/1333 [==============================] - 1073s 805ms/step - loss: 0.3060 - accuracy: 0.8695 - val\_loss: 0.1352 - val\_accuracy: 0.9021\nEpoch 26/50\n1333/1333 [==============================] - 1077s 808ms/step - loss: 0.2891 - accuracy: 0.8775 - val\_loss: 0.3021 - val\_accuracy: 0.9023\nEpoch 27/50\n1333/1333 [==============================] - 1070s 803ms/step - loss: 0.3003 - accuracy: 0.8748 - val\_loss: 0.5724 - val\_accuracy: 0.9101\nEpoch 28/50\n1333/1333 [==============================] - 1069s 802ms/step - loss: 0.2912 - accuracy: 0.8757 - val\_loss: 0.1978 - val\_accuracy: 0.8987\nEpoch 29/50\n1333/1333 [==============================] - 1067s 800ms/step - loss: 0.2867 - accuracy: 0.8798 - val\_loss: 0.2349 - val\_accuracy: 0.8963\n","name":"stdout"}]}]

## SAVE MODEL

model.save\_weights("model.h5")

## VIRTUALIZE TRAINING

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))

ax1.plot(history.history['loss'], color='b', label="Training loss")

ax1.plot(history.history['val\_loss'], color='r', label="validation loss")

ax1.set\_xticks(np.arange(1, epochs, 1))

ax1.set\_yticks(np.arange(0, 1, 0.1))

ax2.plot(history.history['accuracy'], color='b', label="Training accuracy")

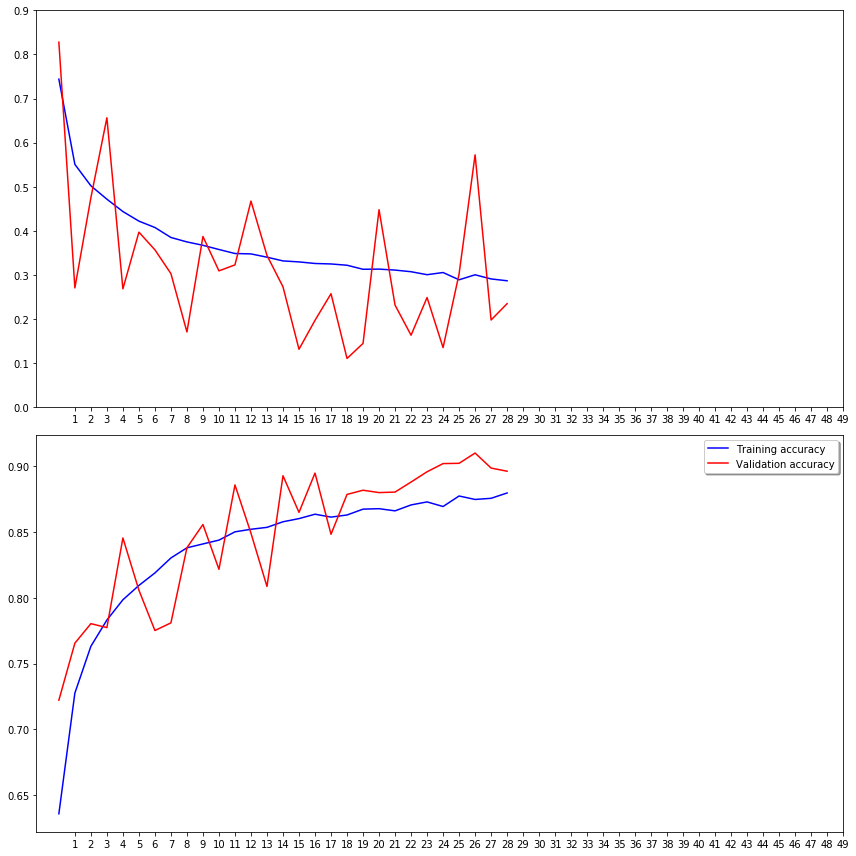
ax2.plot(history.history['val\_accuracy'], color='r',label="Validation accuracy")

ax2.set\_xticks(np.arange(1, epochs, 1))

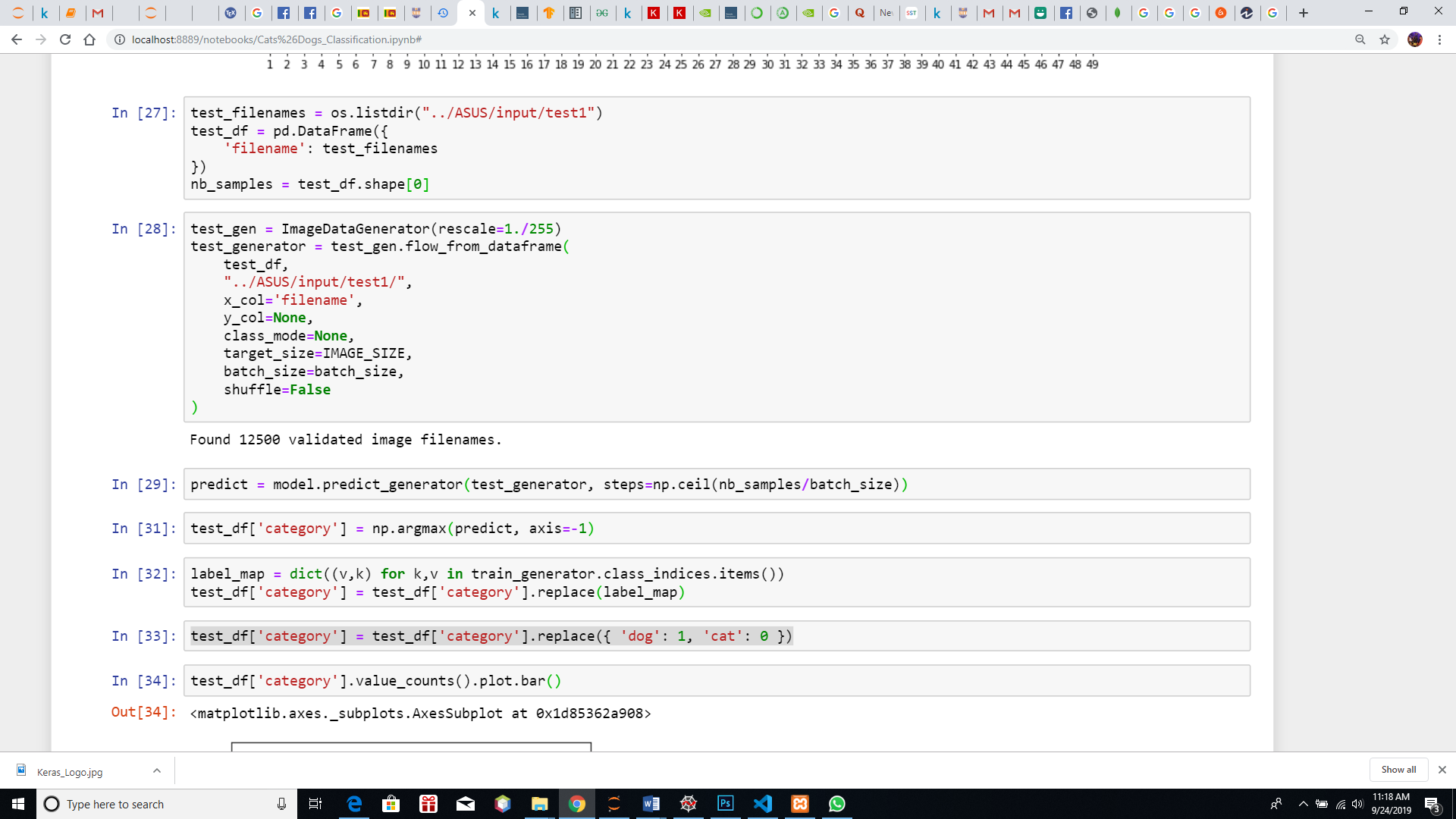
legend = plt.legend(loc='best', shadow=True)

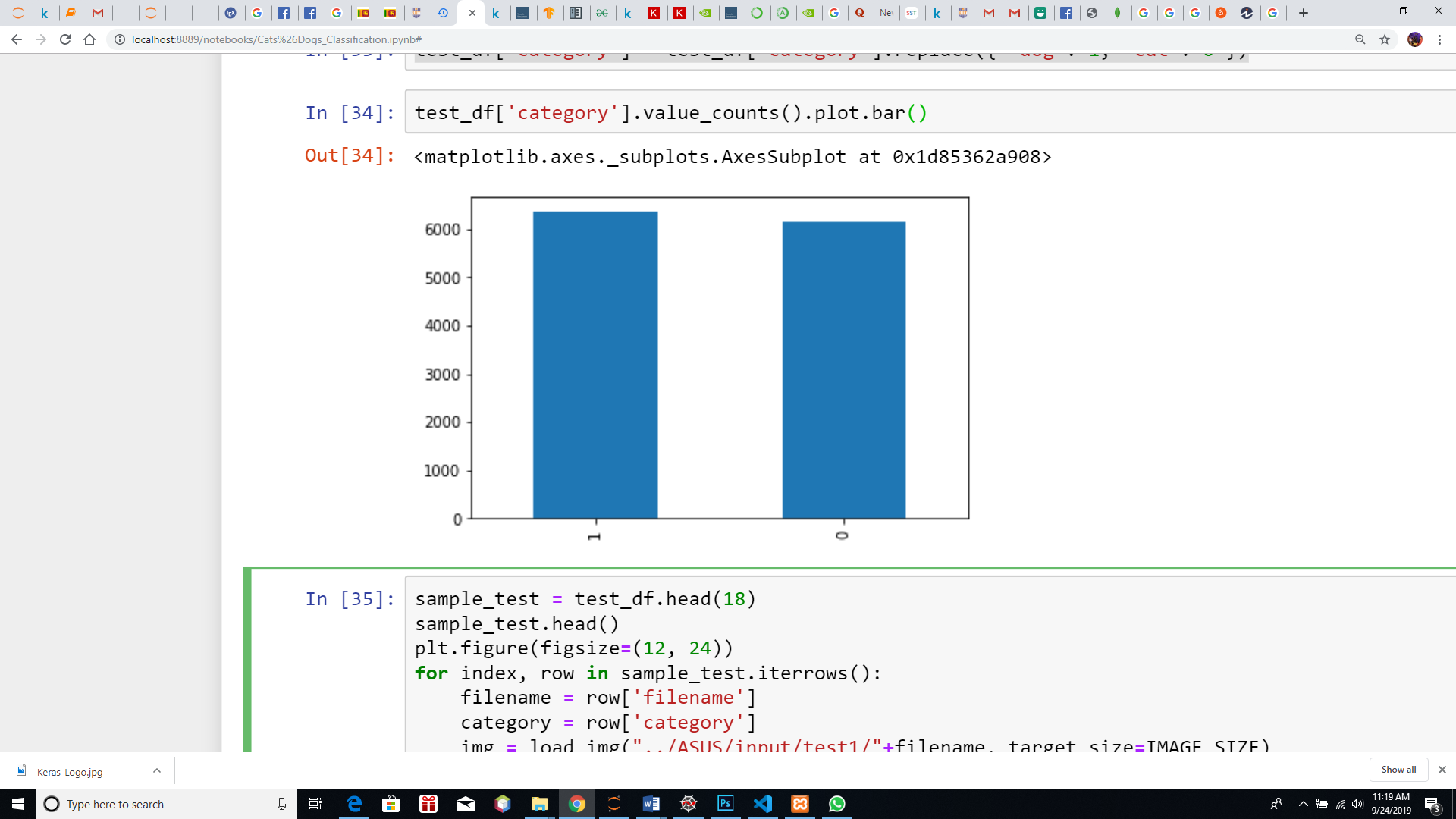
plt.tight\_layout()

plt.show()



## VIRTAULIZE RESULT





## FINAL PREDICTED RESULT WITH IMAGES

sample\_test = test\_df.head(18)

sample\_test.head()

plt.figure(figsize=(12, 24))

for index, row in sample\_test.iterrows():

filename = row['filename']

category = row['category']

img = load\_img("../ASUS/input/test1/"+filename, target\_size=IMAGE\_SIZE)

plt.subplot(6, 3, index+1)

plt.imshow(img)

plt.xlabel(filename + '(' + "{}".format(category) + ')' )

plt.tight\_layout()

plt.show()

In finally we can see the results of classified images. All images include the label. If a ‘filename (0)’ and if a dog ‘filename (1). All the images labeled as cats for (0) and dogs for (1). Using this algorithm, we can insert a large number of images of cats and dogs randomly, and the algorithm will identify who are the cats, and others will be dogs and classify the images and labeled.

# SUBMISSION

submission\_df = test\_df.copy()

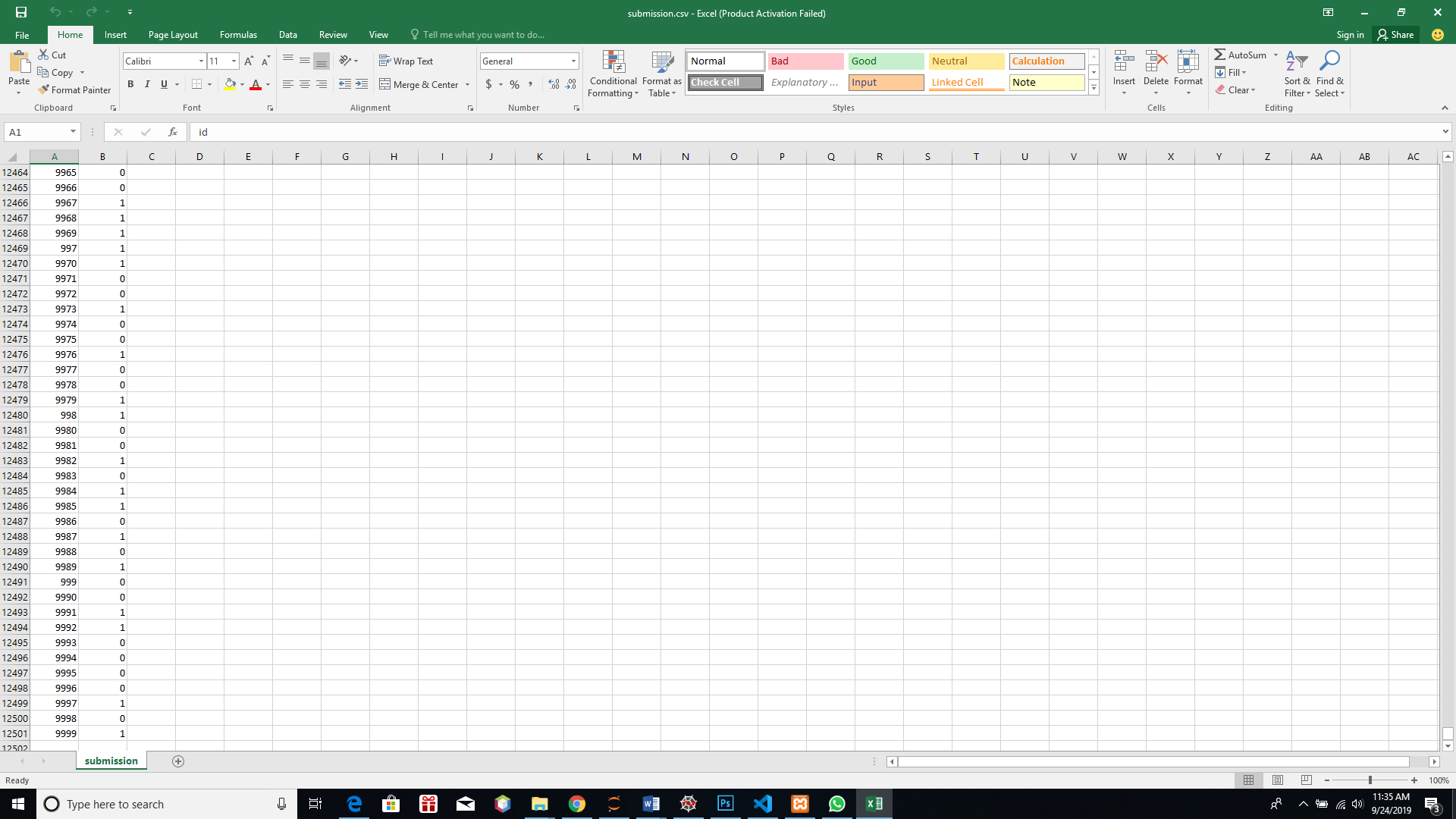
submission\_df['id'] = submission\_df['filename'].str.split('.').str[0]

submission\_df['label'] = submission\_df['category']

submission\_df.drop(['filename', 'category'], axis=1, inplace=True)

submission\_df.to\_csv('submission.csv', index=False)

Final result set of 12 500 test images.



# APPENDIX

#!/usr/bin/env python

# coding: utf-8

# In[1]: Import Library

import numpy as np

import pandas as pd

from keras.preprocessing.image import ImageDataGenerator, load\_img

from keras.utils import to\_categorical

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

import random

import os

print(os.listdir("../ASUS/input"))

# In[2]: Define Constant Values

FAST\_RUN = False

IMAGE\_WIDTH=128

IMAGE\_HEIGHT=128

IMAGE\_SIZE=(IMAGE\_WIDTH, IMAGE\_HEIGHT)

IMAGE\_CHANNELS=3

# In[3]: Prepare Traning Data

filenames = os.listdir("../ASUS/input/train/")

categories = []

for filename in filenames:

category = filename.split('.')[0]

if category == 'dog':

categories.append(1)

else:

categories.append(0)

df = pd.DataFrame({

'filename': filenames,

'category': categories

})

# In[4]: Data-Frame head

df.head()

# In[5]: Data-Frame Tail

df.tail()

# In[6]: See Total In count (Using bar diagram)

df['category'].value\_counts().plot.bar()

# In[7]: See sample image

sample = random.choice(filenames)

image = load\_img("../ASUS/input/train/"+sample)

plt.imshow(image)

# In[8]: Build Model

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(IMAGE\_WIDTH, IMAGE\_HEIGHT, IMAGE\_CHANNELS)))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax')) # 2 because we have cat and dog classes

model.compile(loss='categorical\_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

model.summary()

# In[9]: Callbacks

from keras.callbacks import EarlyStopping, ReduceLROnPlateau

# In[10]:

earlystop = EarlyStopping(patience=10)

# In[11]:

learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_acc',

patience=2,

verbose=1,

factor=0.5,

min\_lr=0.00001)

# In[12]:

callbacks = [earlystop, learning\_rate\_reduction]

# In[13]: Prepare data

df["category"] = df["category"].replace({0: 'cat', 1: 'dog'})

# In[14]:

train\_df, validate\_df = train\_test\_split(df, test\_size=0.20, random\_state=42)

train\_df = train\_df.reset\_index(drop=True)

validate\_df = validate\_df.reset\_index(drop=True)

# In[15]:

train\_df['category'].value\_counts().plot.bar()

# In[16]:

validate\_df['category'].value\_counts().plot.bar()

# In[17]:

total\_train = train\_df.shape[0]

total\_validate = validate\_df.shape[0]

batch\_size=15

# In[18]: Traning Generator

train\_datagen = ImageDataGenerator(

rotation\_range=15,

rescale=1./255,

shear\_range=0.1,

zoom\_range=0.2,

horizontal\_flip=True,

width\_shift\_range=0.1,

height\_shift\_range=0.1

)

train\_generator = train\_datagen.flow\_from\_dataframe(

train\_df,

"../ASUS/input/train/",

x\_col='filename',

y\_col='category',

target\_size=IMAGE\_SIZE,

class\_mode='categorical',

batch\_size=batch\_size

)

# In[19]: Validation Generator

validation\_datagen = ImageDataGenerator(rescale=1./255)

validation\_generator = validation\_datagen.flow\_from\_dataframe(

validate\_df,

"../ASUS/input/train/",

x\_col='filename',

y\_col='category',

target\_size=IMAGE\_SIZE,

class\_mode='categorical',

batch\_size=batch\_size

)

# In[20]: See how the generator work

example\_df = train\_df.sample(n=1).reset\_index(drop=True)

example\_generator = train\_datagen.flow\_from\_dataframe(

example\_df,

"../ASUS/input/train/",

x\_col='filename',

y\_col='category',

target\_size=IMAGE\_SIZE,

class\_mode='categorical'

)

# In[21]:

plt.figure(figsize=(12, 12))

for i in range(0, 15):

plt.subplot(5, 3, i+1)

for X\_batch, Y\_batch in example\_generator:

image = X\_batch[0]

plt.imshow(image)

break

plt.tight\_layout()

plt.show()

# In[22]: Fit Model

epochs=3 if FAST\_RUN else 50

history = model.fit\_generator(

train\_generator,

epochs=epochs,

validation\_data=validation\_generator,

validation\_steps=total\_validate//batch\_size,

steps\_per\_epoch=total\_train//batch\_size,

callbacks=callbacks

)

# In[23]: Save Model

model.save\_weights("model.h5")

# In[25]: Virtualize Training

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))

ax1.plot(history.history['loss'], color='b', label="Training loss")

ax1.plot(history.history['val\_loss'], color='r', label="validation loss")

ax1.set\_xticks(np.arange(1, epochs, 1))

ax1.set\_yticks(np.arange(0, 1, 0.1))

ax2.plot(history.history['accuracy'], color='b', label="Training accuracy")

ax2.plot(history.history['val\_accuracy'], color='r',label="Validation accuracy")

ax2.set\_xticks(np.arange(1, epochs, 1))

legend = plt.legend(loc='best', shadow=True)

plt.tight\_layout()

plt.show()

# In[27]: Prepare Testing Data

test\_filenames = os.listdir("../ASUS/input/test1")

test\_df = pd.DataFrame({

'filename': test\_filenames

})

nb\_samples = test\_df.shape[0]

# In[28]: Create Testing Generator

test\_gen = ImageDataGenerator(rescale=1./255)

test\_generator = test\_gen.flow\_from\_dataframe(

test\_df,

"../ASUS/input/test1/",

x\_col='filename',

y\_col=None,

class\_mode=None,

target\_size=IMAGE\_SIZE,

batch\_size=batch\_size,

shuffle=False

)

# In[29]: Predict

predict = model.predict\_generator(test\_generator, steps=np.ceil(nb\_samples/batch\_size))

# In[31]:

test\_df['category'] = np.argmax(predict, axis=-1)

# In[32]:

label\_map = dict((v,k) for k,v in train\_generator.class\_indices.items())

test\_df['category'] = test\_df['category'].replace(label\_map)

# In[33]:

test\_df['category'] = test\_df['category'].replace({ 'dog': 1, 'cat': 0 })

# In[34]: Virtaulize Result

test\_df['category'].value\_counts().plot.bar()

# In[35]: See predicted result with images

sample\_test = test\_df.head(18)

sample\_test.head()

plt.figure(figsize=(12, 24))

for index, row in sample\_test.iterrows():

filename = row['filename']

category = row['category']

img = load\_img("../ASUS/input/test1/"+filename, target\_size=IMAGE\_SIZE)

plt.subplot(6, 3, index+1)

plt.imshow(img)

plt.xlabel(filename + '(' + "{}".format(category) + ')' )

plt.tight\_layout()

plt.show()

# In[36]: Submission

submission\_df = test\_df.copy()

submission\_df['id'] = submission\_df['filename'].str.split('.').str[0]

submission\_df['label'] = submission\_df['category']

submission\_df.drop(['filename', 'category'], axis=1, inplace=True)

submission\_df.to\_csv('submission.csv', index=False)

# REFERENCES

1. “Dogs vs. Cats,” *Kaggle*. [Online]. Available: https://www.kaggle.com/c/dogs-vs-cats/data. [Accessed: 10-Sep-2019].
2. “Getting started with the Keras Sequential model,” *Guide to the Sequential model - Keras Documentation*. [Online]. Available: https://keras.io/getting-started/sequential-model-guide/. [Accessed: 12-Sep-2019].
3. Medium. (2019). *Setup a Python Environment for Machine Learning and Deep Learning*. [online] Available at: https://towardsdatascience.com/setup-an-environment-for-machine-learning-and-deep-learning-with-anaconda-in-windows-5d7134a3db10 [Accessed 24 Aug. 2019].
4. Medium. (2019). *Image Classifier - Cats🐱 vs Dogs🐶*. [online] Available at: https://towardsdatascience.com/image-classifier-cats-vs-dogs-with-convolutional-neural-networks-cnns-and-google-colabs-4e9af21ae7a8 [Accessed 16 Sep. 2019].
5. Investopedia. (2019). *How Deep Learning Can Help Prevent Financial Fraud*. [online] Available at: https://www.investopedia.com/terms/d/deep-learning.asp [Accessed 11 Sep. 2019].
6. “Install TensorFlow with pip  :   TensorFlow,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/install/pip. [Accessed: 14-Sep-2019].
7. “Getting started with Anaconda¶,” *Getting started with Anaconda - Anaconda 2.0 documentation*. [Online]. Available: https://docs.anaconda.com/anaconda/user-guide/getting-started/. [Accessed: 08-Sep-2019].