Image classifier for identifying cat-vs-dogs using TFLearn in Python.

**Machine Learning** is now one of the hottest topics around the world. Well, it can even be said of the new electricity in today’s world. But to be precise what is Machine Learning, well it’s just one way of teaching the machine by feeding a large amount of data. To know more about Machine learning and its algorithms you can refer to some links that are provided in the Reference sections of this article.

Today, we will create an Image Classifier of our own that can distinguish whether a **given pic is of a dog or cat** or something else depending upon your fed data. To achieve our goal, we will use one of the famous machine learning algorithms out there which are used for Image Classification i.e. Convolutional Neural Network(or CNN).

So basically what is CNN – as we know it’s a machine learning algorithm for machines to understand the features of the image with foresight and remember the features to guess whether the name of the new image is fed to the machine. Since it’s not an article explaining CNN so I’ll add some links in the end if you guys are interested in how CNN works and behaves.   
So after going through all those links let us see how to create our very own cat-vs-dog image classifier. For the dataset we will use the Kaggle dataset of cat-vs-dog:

Now after getting the data set, we need to preprocess the data a bit and provide labels to each of the images given there during training the data set. To do so we can see that name of each image of the training data set is either start with “cat” or “dog” so we will use that to our advantage then we use one hot encoder for the machine to understand the labels(cat[1, 0] or dog[0, 1]).

def label\_img(img):

word\_label = img.split('.')[-3]

# DIY One hot encoder

if word\_label == 'cat': return [1, 0]

elif word\_label == 'dog': return [0, 1]

**Libraries Required:**

* + **TFLearn** – Deep learning library featuring a higher-level API for TensorFlow used to create layers of our CNN
  + **tqdm** – Instantly make your loops show a smart progress meter, just for simple design sake
  + [**numpy**](https://www.geeksforgeeks.org/numpy-in-python-set-1-introduction/)– To process the image matrices
  + **open-cv** – To process the image like converting them to grayscale and etc.
  + **os** – To access the file system to read the image from the train and test directory from our machines
  + **random** – To shuffle the data to overcome the biasing
  + **matplotlib** – To display the result of our predictive outcome.
  + **TensorFlow** – Just to use the tensorboard to compare the loss and adam curve our result data or obtained log.

TRAIN\_DIR and TEST\_DIR should be set according to the user’s convenience and play with the basic hyperparameters like an epoch, learning rate, etc to improve the accuracy. I have converted the image to grayscale so that we will only have to deal with a 2-d matrix otherwise 3-d matrix is tough to directly apply CNN

Python3

|  |
| --- |
| # Python program to create |

# Python program to create

# Image Classifier using CNN

# Importing the required libraries

import cv2

import os

import numpy as np

from random import shuffle

from tqdm import tqdm

'''Setting up the env'''

TRAIN\_DIR = 'E:/dataset / Cats\_vs\_Dogs / train'

TEST\_DIR = 'E:/dataset / Cats\_vs\_Dogs / test1'

IMG\_SIZE = 50

LR = 1e-3

'''Setting up the model which will help with tensorflow models'''

MODEL\_NAME = 'dogsvscats-{}-{}.model'.format(LR, '6conv-basic')

'''Labelling the dataset'''

def label\_img(img):

word\_label = img.split('.')[-3]

# DIY One hot encoder

if word\_label == 'cat': return [1, 0]

elif word\_label == 'dog': return [0, 1]

'''Creating the training data'''

def create\_train\_data():

# Creating an empty list where we should store the training data

# after a little preprocessing of the data

training\_data = []

# tqdm is only used for interactive loading

# loading the training data

for img in tqdm(os.listdir(TRAIN\_DIR)):

# labeling the images

label = label\_img(img)

path = os.path.join(TRAIN\_DIR, img)

# loading the image from the path and then converting them into

# grayscale for easier covnet prob

img = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

# resizing the image for processing them in the covnet

img = cv2.resize(img, (IMG\_SIZE, IMG\_SIZE))

# final step-forming the training data list with numpy array of the images

training\_data.append([np.array(img), np.array(label)])

# shuffling of the training data to preserve the random state of our data

shuffle(training\_data)

# saving our trained data for further uses if required

np.save('train\_data.npy', training\_data)

return training\_data

'''Processing the given test data'''

# Almost same as processing the training data but

# we dont have to label it.

def process\_test\_data():

testing\_data = []

for img in tqdm(os.listdir(TEST\_DIR)):

path = os.path.join(TEST\_DIR, img)

img\_num = img.split('.')[0]

img = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

img = cv2.resize(img, (IMG\_SIZE, IMG\_SIZE))

testing\_data.append([np.array(img), img\_num])

shuffle(testing\_data)

np.save('test\_data.npy', testing\_data)

return testing\_data

'''Running the training and the testing in the dataset for our model'''

train\_data = create\_train\_data()

test\_data = process\_test\_data()

# train\_data = np.load('train\_data.npy')

# test\_data = np.load('test\_data.npy')

'''Creating the neural network using tensorflow'''

# Importing the required libraries

import tflearn

from tflearn.layers.conv import conv\_2d, max\_pool\_2d

from tflearn.layers.core import input\_data, dropout, fully\_connected

from tflearn.layers.estimator import regression

import tensorflow as tf

tf.compat.v1.reset\_default\_graph()

convnet = input\_data(shape =[None, IMG\_SIZE, IMG\_SIZE, 1], name ='input')

convnet = conv\_2d(convnet, 32, 5, activation ='relu')

convnet = max\_pool\_2d(convnet, 5)

convnet = conv\_2d(convnet, 64, 5, activation ='relu')

convnet = max\_pool\_2d(convnet, 5)

convnet = conv\_2d(convnet, 128, 5, activation ='relu')

convnet = max\_pool\_2d(convnet, 5)

convnet = conv\_2d(convnet, 64, 5, activation ='relu')

convnet = max\_pool\_2d(convnet, 5)

convnet = conv\_2d(convnet, 32, 5, activation ='relu')

convnet = max\_pool\_2d(convnet, 5)

convnet = fully\_connected(convnet, 1024, activation ='relu')

convnet = dropout(convnet, 0.8)

convnet = fully\_connected(convnet, 2, activation ='softmax')

convnet = regression(convnet, optimizer ='adam', learning\_rate = LR,

loss ='categorical\_crossentropy', name ='targets')

model = tflearn.DNN(convnet, tensorboard\_dir ='log')

# Splitting the testing data and training data

train = train\_data[:-500]

test = train\_data[-500:]

'''Setting up the features and labels'''

# X-Features & Y-Labels

X = np.array([i[0] for i in train]).reshape(-1, IMG\_SIZE, IMG\_SIZE, 1)

Y = np.array([i[1] for i in train])

test\_x = np.array([i[0] for i in test]).reshape(-1, IMG\_SIZE, IMG\_SIZE, 1)

test\_y = np.array([i[1] for i in test])

'''Fitting the data into our model'''

# epoch = 5 taken

model.fit({'input': X}, {'targets': Y}, n\_epoch = 5,

validation\_set =({'input': test\_x}, {'targets': test\_y}),

snapshot\_step = 500, show\_metric = True, run\_id = MODEL\_NAME)

model.save(MODEL\_NAME)

'''Testing the data'''

import matplotlib.pyplot as plt

# if you need to create the data:

# test\_data = process\_test\_data()

# if you already have some saved:

test\_data = np.load('test\_data.npy')

fig = plt.figure()

for num, data in enumerate(test\_data[:20]):

# cat: [1, 0]

# dog: [0, 1]

img\_num = data[1]

img\_data = data[0]

y = fig.add\_subplot(4, 5, num + 1)

orig = img\_data

data = img\_data.reshape(IMG\_SIZE, IMG\_SIZE, 1)

# model\_out = model.predict([data])[0]

model\_out = model.predict([data])[0]

if np.argmax(model\_out) == 1: str\_label ='Dog'

else: str\_label ='Cat'

y.imshow(orig, cmap ='gray')

plt.title(str\_label)

y.axes.get\_xaxis().set\_visible(False)

y.axes.get\_yaxis().set\_visible(False)

plt.show()