mport os

import cv2

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import itemfreq

from color\_recognition\_api import knn\_classifier as knn\_classifier

def color\_histogram\_of\_test\_image(test\_src\_image):

# load the image

image = test\_src\_image

chans = cv2.split(image)

colors = ('b', 'g', 'r')

features = []

feature\_data = ''

counter = 0

for (chan, color) in zip(chans, colors):

counter = counter + 1

hist = cv2.calcHist([chan], [0], None, [256], [0, 256])

features.extend(hist)

import cv2

from color\_recognition\_api import color\_histogram\_feature\_extraction

from color\_recognition\_api import knn\_classifier

import os

import os.path

import sys

# read the test image

try:

source\_image = cv2.imread(sys.argv[1])

except:

source\_image = cv2.imread('black\_cat.jpg')

prediction = 'n.a.'

# checking whether the training data is ready

PATH = './training.data'

if os.path.isfile(PATH) and os.access(PATH, os.R\_OK):

print ('training data is ready, classifier is loading...')

else:

print ('training data is being created...')

open('training.data', 'w')

color\_histogram\_feature\_extraction.training()

import cv2

from color\_recognition\_api import color\_histogram\_feature\_extraction

from color\_recognition\_api import knn\_classifier

import os

import os.path

cap = cv2.VideoCapture(1)

(ret, frame) = cap.read()

prediction = 'n.a.'

# checking whether the training data is ready

PATH = './training.data'

if os.path.isfile(PATH) and os.access(PATH, os.R\_OK):

print ('training data is ready, classifier is loading...')

else:

print ('training data is being created...')

open('training.data', 'w')

color\_histogram\_feature\_extraction.training()

print ('training data is ready, classifier is loading...')

while True:

import csv

import random

import math

import operator

import cv2

# calculation of euclidead distance

def calculateEuclideanDistance(variable1, variable2, length):

distance = 0

for x in range(length):

distance += pow(variable1[x] - variable2[x], 2)

return math.sqrt(distance)

# get k nearest neigbors

def kNearestNeighbors(training\_feature\_vector, testInstance, k):

distances = []

length = len(testInstance)

for x in range(len(training\_feature\_vector)):

dist = calculateEuclideanDistance(testInstance,

training\_feature\_vector[x], length)

distances.append((training\_feature\_vector[x], dist))