# USAGE

# python simple\_neural\_network.py --dataset kaggle\_dogs\_vs\_cats --model output/simple\_neural\_network.hdf5

# import the necessary packages

from sklearn.preprocessing import LabelEncoder

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

from keras.models import Sequential

from keras.layers import Activation

from keras.optimizers import SGD

from keras.layers import Dense

from keras.utils import np\_utils

from imutils import paths

import numpy as np

import argparse

import cv2

import os

def image\_to\_feature\_vector(image, size=(32, 32)):

# resize the image to a fixed size, then flatten the image into

# a list of raw pixel intensities

return cv2.resize(image, size).flatten()

# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-d", "--dataset", required=True,

help="path to input dataset")

ap.add\_argument("-m", "--model", required=True,

help="path to output model file")

args = vars(ap.parse\_args())

# grab the list of images that we'll be describing

print("[INFO] describing images...")

imagePaths = list(paths.list\_images(args["dataset"]))

# initialize the data matrix and labels list

data = []

labels = []

# loop over the input images

for (i, imagePath) in enumerate(imagePaths):

# load the image and extract the class label (assuming that our

# path as the format: /path/to/dataset/{class}.{image\_num}.jpg

image = cv2.imread(imagePath)

label = imagePath.split(os.path.sep)[-1].split(".")[0]

# construct a feature vector raw pixel intensities, then update

# the data matrix and labels list

features = image\_to\_feature\_vector(image)

data.append(features)

labels.append(label)

# show an update every 1,000 images

if i > 0 and i % 1000 == 0:

print("[INFO] processed {}/{}".format(i, len(imagePaths)))

# encode the labels, converting them from strings to integers

le = LabelEncoder()

labels = le.fit\_transform(labels)

# scale the input image pixels to the range [0, 1], then transform

# the labels into vectors in the range [0, num\_classes] -- this

# generates a vector for each label where the index of the label

# is set to `1` and all other entries to `0`

data = np.array(data) / 255.0

labels = np\_utils.to\_categorical(labels, 2)

# partition the data into training and testing splits, using 75%

# of the data for training and the remaining 25% for testing

print("[INFO] constructing training/testing split...")

(trainData, testData, trainLabels, testLabels) = train\_test\_split(

data, labels, test\_size=0.25, random\_state=42)

# define the architecture of the network

model = Sequential()

model.add(Dense(768, input\_dim=3072, init="uniform",

activation="relu"))

model.add(Dense(384, activation="relu", kernel\_initializer="uniform"))

model.add(Dense(2))

model.add(Activation("softmax"))

# train the model using SGD

print("[INFO] compiling model...")

sgd = SGD(lr=0.01)

model.compile(loss="binary\_crossentropy", optimizer=sgd,

metrics=["accuracy"])

model.fit(trainData, trainLabels, epochs=50, batch\_size=128,

verbose=1)

# show the accuracy on the testing set

print("[INFO] evaluating on testing set...")

(loss, accuracy) = model.evaluate(testData, testLabels,

batch\_size=128, verbose=1)

print("[INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,

accuracy \* 100))

# dump the network architecture and weights to file

print("[INFO] dumping architecture and weights to file...")

model.save(args["model"])