תרגיל 3.5 – ראייה ממוחשבת

מעיין שטרית – 315512715, דוד ניר – 203487293

לצורך הנוחות הקוד המלא של כל התרגיל נמצא בדרייב שבקישור הבא, ויופיע גם בסוף הפתרון:

https://drive.google.com/drive/folders/1z0PE-sLbSCyuBd3H8O81zdOZSypBeWdP?usp=sharing

כמו כן, פתרון תרגיל 4 יופיע גם בהמשך של קובץ זה.

בשאלה זו התבקשנו לממש רשת $VGG16$ שמקבלת תמונות $MNIST$ שחור-לבן של ספרות 0,1 בכתב יד ופולטת פיצ'רים שמתארים את התמונות. לאחר מכן התבקשנו לאמן מודל KNN שיודע לקבל את הפיצ'רים האלה ולהגיד איזו ספרה הפיצ'רים מזהים.

מימוש הרשת שלנו:

הרשת שלנו בנויה מ-4 בלוקי קונבולוציה מהצורה $Conv \rightarrow Conv \rightarrow MaxPool$ ועוד שתי שכבות של $FullyConnected$, היא מקבלת תמונות 28×28 ומחזירה על כל תמונה 1024 פיצ'רים.

בכל בלוק השתמשנו בקונבולוציה בגודל שונה, בסה"כ השתמשנו בגדלים 64,128,256,512 (64 בהתחלה ו512 בסוף).

השכבה הראשונה של ה$Fully\ Connected$ בגודל 4096 והשכבה השניה (ובעצם האחרונה ברשת) בגודל 1024.

להלן הקוד שמייצר את המודל שלנו (בעמוד הבא ה$model\ summary$):

```
model = Sequential()
model.add(Input((28,28,1)))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(MaxPool2D(strides=(2, 2)))
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(MaxPool2D(strides=(2, 2)))
model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
model.add(MaxPool2D(strides=(2, 2)))
model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dense(1024, activation='softmax'))
adam = Adam(lr=1e-4, decay=1e-6)
model.compile(adam, 'categorical_crossentropy', metrics=['accuracy'])
```

להלן ה*model summary*:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 64)        640
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 128)       147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 256)         295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 3, 3, 512)         1180160
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 3, 3, 512)         2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 1, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 4096)              2101248
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              4195328
=================================================================
Total params: 10,980,800
Trainable params: 10,980,800
Non-trainable params: 0
```

לאחר שיצרנו את הרשת, אנחנו מייצרים פיצ'רים ומאמנים עליהם את הknn שלנו:

```
# we use the model as a feature creating function
features = model.predict(train_X)
# create the knn and train it on the generated features and the labels
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(features,train_y)
```

כדי לבדוק את התפקוד של המודל המשולב שלנו ($VGG16 + knn$), אנחנו מייצרים פיצ'רים מה$test\ set$ שלנו ואז מייצרים מהם פרדיקציה באמצעות מודל הknn שלנו. לבסוף משווים מול האמת ובודקים את ה$error\ rate$ שלנו:

```
# extract features from the test data
features = model.predict(test_X)
# predict on the features using knn
knn_prediction = knn.predict(features)
knn_prediction = tf.reshape(knn_prediction, shape=[-1, 1])
assert knn_prediction.shape == test_y.shape
# compute the error rate
err_rate = np.sum((knn_prediction != test_y)) / len(test_y)
return err_rate
```

חזרנו על התהליך 5 פעמים וחישבנו ממוצע של השגיאות ($\sim 0.002\ error\ rate$):

```
Average knn prediction error in 5 attempts: 0.002458628841607565
```

נספח – הקוד המלא לחלק הראשון:

```
import os
from mlxtend.data import loadlocal_mnist
from pandas import DataFrame
import pandas as pd
from matplotlib import pyplot as plt

# disable tensorflow log spamming, will only output errors
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Input

LABEL_COLUMN_NAME = 'label'
LABEL_COL_HEADER = 'labels'


def load_MNIST_data_into_dataframe(MNIST_paths, verbose=False):
    """ Given two paths which point to two files describing MNIST data in the form of X and y,
    opens it and reads it into usable DataFrames, returns said DataFrames.
    Assumes valid input."""
    if verbose:
        print("MNIST data paths:")
        print("\tData:  \t" + MNIST_paths[0])
        print("\tLabels:\t" + MNIST_paths[1])
    # load the data into numpy arrays
    X, y = loadlocal_mnist(MNIST_paths[0], MNIST_paths[1])
    # convert the numpy arrays into DataFrames
    X_df = DataFrame(X)
    y_df = DataFrame(y)
    if verbose:
        print("Loaded MNIST data shapes:")
        print("\t" + str(X_df.shape))
        print("\t" + str(y_df.shape))
    return X_df, y_df


def plot_black_white_image(sample, ax, label=""):
    """ Receives a sample in the form of a matrix which represents an arrangement of black and
    white pixels, and plots it into given plt."""
    ax.imshow(sample, cmap='Greys')
    ax.set_title(label)


def reshape_sample_as_DF(sample_original, shape):
    """ Reshapes a sample to some new given shape and returns the new one. """
    new_sample = pd.DataFrame(sample_original.reshape(shape[0], shape[1]))
    return new_sample
```

```python
def plot_multiple_bw_images(X, y, cols, rows, label_prefix="", im_x=28, im_y=28):
    """ Plots a grid of images from given data.
    Adds label_prefix+y[i] as the title for the i'th image.
    If in total it needs to plot k images, it will always take the first k from X."""
    fig, axs = plt.subplots(cols, rows)
    cur = 0
    for i in range(rows):
        for j in range(cols):
            # reshape so that we have a 28x28 data instead of 1x768 data
            # this is done so that we can plot it as an image
            sample = reshape_sample_as_DF(X.values[cur], (im_x, im_y))
            label = y.values[cur]
            plot_black_white_image(sample, axs[i, j], label_prefix + str(label))
            cur += 1
    plt.show()


def plot_random_3x3_samples(X, y, n=1, sample_set_size=9):
    """ Plots some random subset of size 9 from given X in a 3x3 grid. """
    rand_samples, rand_labels = None, None
    X[LABEL_COL_HEADER] = y
    for i in range(n):
        rand_samples = X.sample(sample_set_size)
        rand_labels = pd.DataFrame(rand_samples.pop(LABEL_COL_HEADER))
        plot_multiple_bw_images(rand_samples, rand_labels, 3, 3)
    X.pop(LABEL_COL_HEADER)
    return rand_samples, rand_labels


def plot_single_gs_img(img, title):
    plt.imshow(img, cmap='Greys')


def drop_all_non_zero_or_one_values_from_mnist_data(X, y, verbose=False):
    """
    Drops all data where the label is not 0 or 1
    """
    X[LABEL_COLUMN_NAME] = y
    if verbose:
        print("dropping all images with labels not 0 or 1")
        print(X.shape)
        zeroes = X[X[LABEL_COLUMN_NAME] == 0]
        ones = X[X[LABEL_COLUMN_NAME] == 1]
        print('\tzeroes', zeroes.shape)
        print('\tones', ones.shape)
    relevant_data = X.drop(X[X[LABEL_COLUMN_NAME] > 1].index, inplace=False)
    if verbose:
        print('\tshape after filtering:', relevant_data.shape)
    y = pd.DataFrame(relevant_data.pop(LABEL_COLUMN_NAME))
    if verbose:
        print('\tsplit data into X and y of shapes:', relevant_data.shape, y.shape)
    return relevant_data, y
```

```python
def reshape_df_into_2d_28_square_imgs(data, verbose=False):
    """
    Assumes given data is a dataframe with 28x28 images flattened.
    Returns the same images reshaped into 28x28.
    """
    if verbose:
        print("Before and after reshape: {}".format(data.shape), end="")
    reshaped_data = tf.reshape(data, shape=[-1, 28, 28, 1])
    if verbose:
        print(" --> {}".format(reshaped_data.shape))
    return reshaped_data


def extract_features_and_train_knn(model, train_X, train_y, verbose=True):
    """
    Creates a knn model that is trained on given model output as data and actual given MNIST labels.
    """
    # we use the model as a feature creating function
    features = model.predict(train_X)
    if verbose:
        print("Extracting features:")
        print("\tX shape: {}\n\tFeatures shape: {}".format(train_X.shape, features.shape))
    # create the knn and train it on the generated features and the labels
    knn = KNeighborsClassifier(n_neighbors=1)
    knn.fit(features, train_y)
    return knn


def test_error_rate(knn, model, test_X, test_y):
    """
    Self explanatory.
    """
    # extract features from the test data
    features = model.predict(test_X)
    # predict on the features using knn
    knn_prediction = knn.predict(features)
    knn_prediction = tf.reshape(knn_prediction, shape=[-1, 1])
    assert knn_prediction.shape == test_y.shape
    # compute the error rate
    err_rate = np.sum((knn_prediction != test_y)) / len(test_y)
    return err_rate
```

```python
def create_vgg16_model(show_model_summary=False):
    """
    Architecture:
    Input (MNIST shaped images)
    2xConv (64 layers, relu)
    MaxPool
    2xConv (128 layers, relu)
    MaxPool
    2xConv (256 layers, relu)
    MaxPool
    2xConv (512 layers, relu)
    MaxPool
    Flatten()
    Dense (4096, relu)
    Dense (1024, softmax)
    """
    model = Sequential()
    model.add(Input((28, 28, 1)))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(MaxPool2D(strides=(2, 2)))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(MaxPool2D(strides=(2, 2)))
    model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
    model.add(MaxPool2D(strides=(2, 2)))
    model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
    model.add(MaxPool2D(strides=(2, 2)))
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(1024, activation='softmax'))
    adam = Adam(lr=1e-4, decay=1e-6)
    model.compile(adam, 'categorical_crossentropy', metrics=['accuracy'])
    if show_model_summary:
        model.summary()
    return model
```

```python
if __name__ == '__main__':

    # MNIST datasets downloaded from: http://yann.lecun.com/exdb/mnist/
    TRAIN_X = "data/train-images.idx3-ubyte"
    TRAIN_Y = "data/train-labels.idx1-ubyte"
    TEST_X = "data/t10k-images.idx3-ubyte"
    TEST_Y = "data/t10k-labels.idx1-ubyte"

    # load all the MNIST train/test data
    train_X, train_y = load_MNIST_data_into_dataframe((TRAIN_X, TRAIN_Y), verbose=True)
    test_X, test_y = load_MNIST_data_into_dataframe((TEST_X, TEST_Y), verbose=True)

    # drop all values non 0 or 1 from the train/test data
    train_X, train_y = drop_all_non_zero_or_one_values_from_mnist_data(train_X, train_y, verbose=True)
    test_X, test_y = drop_all_non_zero_or_one_values_from_mnist_data(test_X, test_y, verbose=True)

    # reshape our data to 2D shape
    train_X = reshape_df_into_2d_28_square_imgs(train_X, verbose=True)
    test_X = reshape_df_into_2d_28_square_imgs(test_X, verbose=True)
    print('\n-- Finished Preparing Data --\n')

    create_vgg16_model(show_model_summary=True)

    print('\n-- Computing Average Error Rate --\n')

    train_sample_num = 500
    attempts = 5
    errs = []
    for i in range(attempts):
        vgg16_model = create_vgg16_model()
        knn = extract_features_and_train_knn(vgg16_model, train_X[:train_sample_num],
train_y[:train_sample_num],
                                             verbose=False)
        err = test_error_rate(knn, vgg16_model, test_X, test_y)
        errs.append(err)
    avg_err = sum(errs) / attempts
    print("\n-- Finished Creation/Prediction Step --\n")
    print("Average knn prediction error in {} attempts: {}".format(attempts, avg_err))
```

<u>תרגיל 4 – ראייה ממוחשבת</u>

<u>מעיין שטרית – 315512715, דוד ניר – 203487293</u>

בתרגיל זה בחרנו לממש מערכת המזהה מתי כלב עולה על ספה. הרעיון נולד מתוך צורך אמיתי לזיהוי כשכלב עולה על הספה בזמן שהבעלים שלו לא בבית. חשבנו שניתן להרחיב קוד זה ובזמן זיהוי של כלב על הספה להפעיל אזעקה שתבהיל את הכלב, או הקלטה של הבעלים עם המילים "רדי מהספה!!", וכך הבעלים יוכל לצאת מהבית ללא חשש. כמובן עם הצבת מצלמה המחוברת לאלגוריתם שלנו.

לצורך ניסוי ראשוני, צילמנו את הכלבה לילי קופצת על הספה ואז יורדת ממנה.

בדרייב שבקישור הבא:

https://drive.google.com/drive/folders/1z0PE-sLbSCyuBd3H8O81zdOZSypBeWdP?usp=sharing

נמצא הסרטון המקורי שבו ניתן לראות את לילי קופת על הספה, וגם הסרטון שאחרי העיבוד עם *yolo*, שבו ניתן לראות התראה כאשר לילי עולה על הספה.

בעיבוד שלנו ההתראה הינה "$Lily\ on\ sofa$" על מנת להתאים את המצב, כמובן שזה ניתן לשינוי (כלומר לשים שם אחר או אפילו להכליל ולכתוב "$Dog\ on\ sofa$").

כעת נרחיב על הפרטים הטכניים בתרגיל שלנו:

<u>הקלט:</u>

path לסרטון שבו ניתן לראות את לילי קופצת על ספה.
באלגוריתם שלנו תחילה אנו מפצלים את הסרטון ל *frames* שונים, ואז מריצים עיבוד על כל *frame* בנפרד.
בכל *frame* אנו מחפשים אובייקטים שהינם כלב או ספה, ולאחר שמצאנו אנו בודקים האם יש חיתוך בין האובייקטים.

אחד הקשיים שנתקלנו בהם הינו – מה בידיוק נחשב חפיפה, בסופו של דבר החלטנו שבשלב ראשוני נבדוק האם יש חפיפה בצורה בסיסית.
התנהלות זו עלולה ליצור *false alarms* במידה ולילי תעמוד ליד הספה ולא על הספה.
ניתן להתמודד עם זה על ידי כך שבמקום בדיקה בסיסית אם קיימת חפיפה בין האובייקטים, נגדיר שטח חפיפה מסוים, כך שאם שטח החפיפה בתמונה גדול ממנו – האלגוריתם יתריע שיש כלב על הספה, ואם שטח החפיפה קטן ממנו – האלגוריתם לא יתריע.

<u>התוצאות:</u>

התוצאה הינה סרטון עם זיהויים של האובייקטים הרלונטים, ובנוסף, בפינה הימנית של הסרטון תופיע התראה כאשר לילי באמת על הספה.

האלגוריתם בפייתון – קיים גם בקובץ *py* בדרייב :

```
# import the necessary packages
import numpy as np
import cv2


def load_yolo():
    """
    Load and create YOLO net
    :return:
    """
    net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
    classes = []
    with open("coco.names", "r") as f:
        classes = [line.strip() for line in f.readlines()]
    layers_names = net.getLayerNames()
    output_layers = [layers_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
    colors = np.random.uniform(0, 255, size=(len(classes), 3))
    return net, classes, colors, output_layers


def load_image(img_path):
    """
    Loading image by path
    :param img_path: path to image
    :return:
    """
    img = cv2.imread(img_path)
    img = cv2.resize(img, None, fx=0.4, fy=0.4)
    height, width, channels = img.shape
    return img, height, width, channels


def detect_objects(img, net, outputLayers):
    """
    Detect objects in the given img using the given net
    :return:
    """
    blob = cv2.dnn.blobFromImage(img, scalefactor=0.00392, size=(320, 320), mean=(0, 0, 0), swapRB=True,
crop=False)
    net.setInput(blob)
    outputs = net.forward(outputLayers)
    return blob, outputs
```

```python
def get_box_dimensions(outputs, height, width):
    """
    calculate and return box dimensions
    """
    boxes = []
    confs = []
    class_ids = []
    for output in outputs:
        for detect in output:
            scores = detect[5:]
            class_id = np.argmax(scores)
            conf = scores[class_id]
            if conf > 0.3:
                center_x = int(detect[0] * width)
                center_y = int(detect[1] * height)
                w = int(detect[2] * width)
                h = int(detect[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confs.append(float(conf))
                class_ids.append(class_id)
    return boxes, confs, class_ids


def draw_labels(boxes, confs, colors, class_ids, classes, img):
    indexes = cv2.dnn.NMSBoxes(boxes, confs, 0.5, 0.4)
    font = cv2.FONT_HERSHEY_PLAIN
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            color = colors[i]
            cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
            cv2.putText(img, label, (x, y - 5), font, 1, color, 1)
    cv2.imshow("Image", img)


def image_detect(img_path):
    model, classes, colors, output_layers = load_yolo()
    image, height, width, channels = load_image(img_path)
    blob, outputs = detect_objects(image, model, output_layers)
    boxes, confs, class_ids = get_box_dimensions(outputs, height, width)
    add_text = detect_dog_on_sofa(boxes, class_ids)
    if add_text:
        new_frame = addText(image, 'Lily On Sofa!')
    else:
        new_frame = image
    draw_labels(boxes, confs, colors, class_ids, classes, new_frame)
    while True:
        key = cv2.waitKey(1)
        if key == 27:
            break
```

```python
def start_video(video_path, out_path):
    """
    Main function - search for dof on sofa or ned and alaram about that!!
    :param video_path: path to video
    :return:
    """
    model, classes, colors, output_layers = load_yolo()
    cap = cv2.VideoCapture(video_path)
    video_frames = []
    size = None
    while True:
        _, frame = cap.read()
        try:
            frame = cv2.resize(frame, None, fx=0.4, fy=0.4)
        except:
            break
        height, width, channels = frame.shape
        size = (width, height)
        blob, outputs = detect_objects(frame, model, output_layers)
        boxes, confs, class_ids = get_box_dimensions(outputs, height, width)
        add_text = detect_dog_on_sofa(boxes, class_ids)
        if add_text:
            new_frame = addText(frame, 'Lily On Sofa!')
        else:
            new_frame = frame
        draw_labels(boxes, confs, colors, class_ids, classes, new_frame)

        video_frames.append(np.copy(new_frame))

        key = cv2.waitKey(1)
        if key == 27:
            break

    out = cv2.VideoWriter(out_path, cv2.VideoWriter_fourcc(*'mp4v'), 30, size)

    # out = cv2.VideoWriter(out_path, cv2.VideoWriter_fourcc(*'mp4v'), 30, size)
    for im in video_frames:
        # writing to a image array
        out.write(im)

    out.release()
    cap.release()
```

```python
def detect_dog_on_sofa(boxes, class_ids):
    """
    detect dogs on sofa given boxes of objects
    :return:
    """
    dog = 16
    sofa = 57
    chair = 56
    idx_array = np.array(class_ids)
    dog_indexes = np.where(idx_array == dog)[0]
    sofa_indexes = np.where(idx_array == sofa)[0]
    chair_indexes = np.where(idx_array == chair)[0]
    disallowed_indexes = np.concatenate([sofa_indexes, chair_indexes])

    for disallowed_index in disallowed_indexes:
        x0, y0, w0, h0 = boxes[disallowed_index]
        sofa_rect = {
            'top_left':
                {
                    'x': x0,
                    'y': y0
                },
            'bot_right':
                {
                    'x': x0 + w0,
                    'y': y0 + h0
                }
        }
        for dog_index in dog_indexes:
            x1, y1, w1, h1 = boxes[dog_index]
            dog_rect = {
                'top_left':
                    {
                        'x': x1,
                        'y': y1
                    },
                'bot_right':
                    {
                        'x': x1 + w1,
                        'y': y1 + h1
                    }
            }
            if are_intersect(sofa_rect, dog_rect):
                return True
    return False
```

```python
def are_intersect(rect_a, rect_b):
    """
    The function checks if the rect are intersect and return results
    :param rect_a: rect of one object
    :param rect_b: rect of second object
    :return:
    """
    # If one rectangle is on left side of other

    a_before_b = rect_b['top_left']['x'] >= rect_a['bot_right']['x']
    b_before_a = rect_a['top_left']['x'] >= rect_b['bot_right']['x']

    if a_before_b or b_before_a:
        return False

    a_above_b = rect_b['top_left']['y'] >= rect_a['bot_right']['y']
    b_above_a = rect_a['top_left']['y'] >= rect_b['bot_right']['y']

    # If one rectangle is above other
    if a_above_b or b_above_a:
        return False

    return True


def addText(frame, text):
    # font
    font = cv2.FONT_HERSHEY_SIMPLEX

    # org
    org = (50, 50)

    # fontScale
    font_scale = 1

    # Blue color in BGR
    color = (0, 0, 0)

    # Line thickness of 2 px
    thickness = 2

    # get the width and height of the text box
    (text_width, text_height) = cv2.getTextSize(text, font, fontScale=font_scale, thickness=1)[0]

    # set the text start position
    text_offset_x = org[0]
    text_offset_y = org[1]

    # make the coords of the box with a small padding of two pixels
    box_coords = ((text_offset_x, 0), (text_offset_x + text_width + 2, 100))

    rectangle_bgr = (255, 255, 255)

    cv2.rectangle(frame, box_coords[0], box_coords[1], rectangle_bgr, cv2.FILLED)
```

```
    # Using cv2.putText() method
    new_frame = cv2.putText(frame, text, org, font,
                            font_scale, color, thickness, cv2.LINE_AA)

    return new_frame


if __name__ == '__main__':
    # img_path = 'c:\\temp\\dogs\\oosterpark.jpg'
    # img_path = 'c:\\temp\\dogs\\dog-on-sofa.jpg'
    # image_detect(img_path)
    vid_path = 'c:\\temp\\dogs\\lily4.mp4'
    out_path = 'c:\\temp\\dogs\\lily_detect.mp4'
    start_video(vid_path, out_path)
```