IT Carlow - BSc.
Software development

Automatic Detection of Brand Logos Technical Manual

Student Name: Zhe Cui
Student Number: C00266169
Lecturer: Paul Barry
Date: 06.05.2021

Abstract

Automatic Detection of Brand Logos is a tool identifying brand logos in still and moving images, and calculate how long the logo is visible. This document provides detailed instructions for users.

Table of Contents

Table of Figures

1 Introduction

Automatic Detection of Brand Logos is a tool identifying brand logos in images and videos, and calculate how long the logo is visible. It can not only identify 10 different brands, 5 international brands and 5 Chinese brands namely Adidas, Kappa, New Balance, Nike, Puma, 361, Anta, Erke, Lining, Xtep, but also locate the logo that has not been seen before, the logo that has not been trained. Given the specific logo image, the system can mark the location of the logo in the pictures and videos. It can help evaluate marketing campaigns, capture user reviews of the product, counterfeit detection and protect brands' intellectual property, personalize product recommendations, improve search algorithms.

Automatic Detection of Brand Logos System includes two parts, the Web module and the Deep Learning module. When users visit the page, they can upload a local picture, complete the instructions according to the prompts on the webpage, and the page will finally display the predicted results of the picture. If the image contains a logo, the user can see the specific classification information and confidence. The design principle of the Web module is that after receiving the uploaded picture, the server waits for the instruction of the user to test the picture. After receiving the instruction, the server immediately runs the corresponding training script of the deep learning module, carries out the test based on the trained model, and returns the results to the front-end for display, as shown in Figure 1-1.

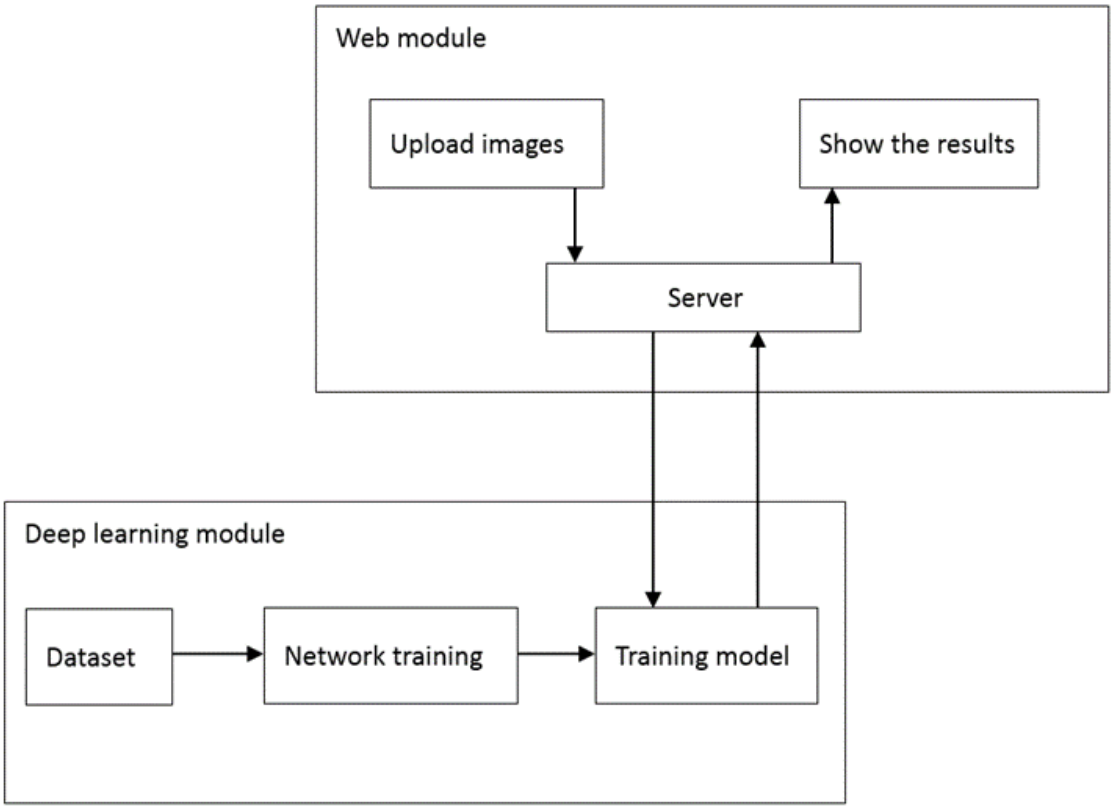

Figure 1-1 Automatic Detection of Brand Logos system flow chart

2 System environment

Python 3.6.5

Anaconda3-5.2.0

How to install: https://www.anaconda.com/distribution/

Download and install Anaconda from the official Anaconda website.

Select “Add Anaconda to my PATH environment variable” during installation.

Tensorflow 1.15.0

Once Anaconda is installed, the next step is to configure the TensorFlow environment, Win+R to start cmd, enter the following command at the command

prompt:

```
conda create -n tensorflow python=3.5

activate tensorflow

pip install tensorflow==1.15.0
```

Keras 2.1.5

```
pip install keras==2.1.5
```

Restart the computer after the installation is complete.

3 How to use Automatic Detection of Brand Logos system

3.1 Image detection

Run *serve.py*, then click the http://localhost:8000/. The initial page of the website is shown as Figure 3-2 and then click "Please choose an image", click "Submit", the system will detect the picture, and the detection result and confidence will be displayed on the page.


```
Colocations handled automatically by placer.
logs/ep032-loss14.261-val_loss13.502.h5 model, anchors, and classes loaded.
Development server is running at http://localhost:8000
Quit the server with Control-C
[I 210424 11:37:24 web:2243] 200 GET / (::1) 7.51ms
[W 210424 11:37:25 web:2243] 404 GET /favicon.ico (::1) 7.01ms
```

Figure 3-1 The output of serve.py

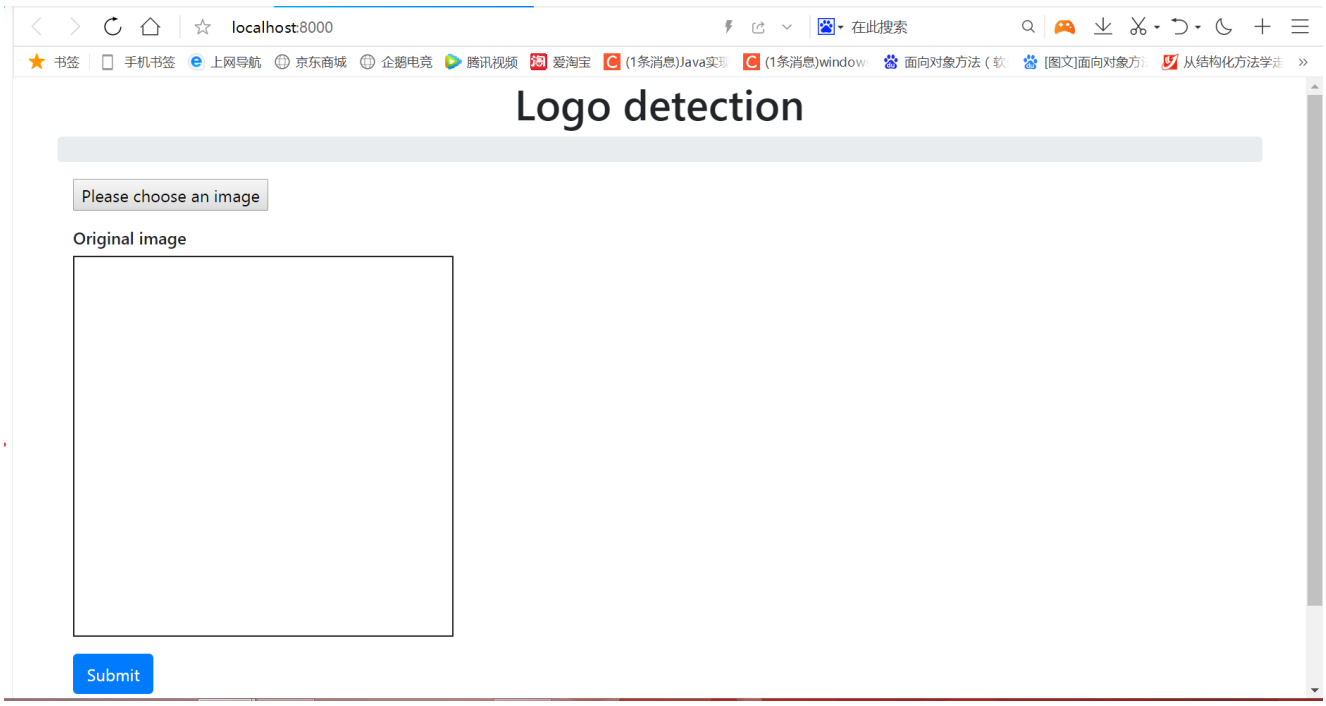

Figure 3-2 Website initial page

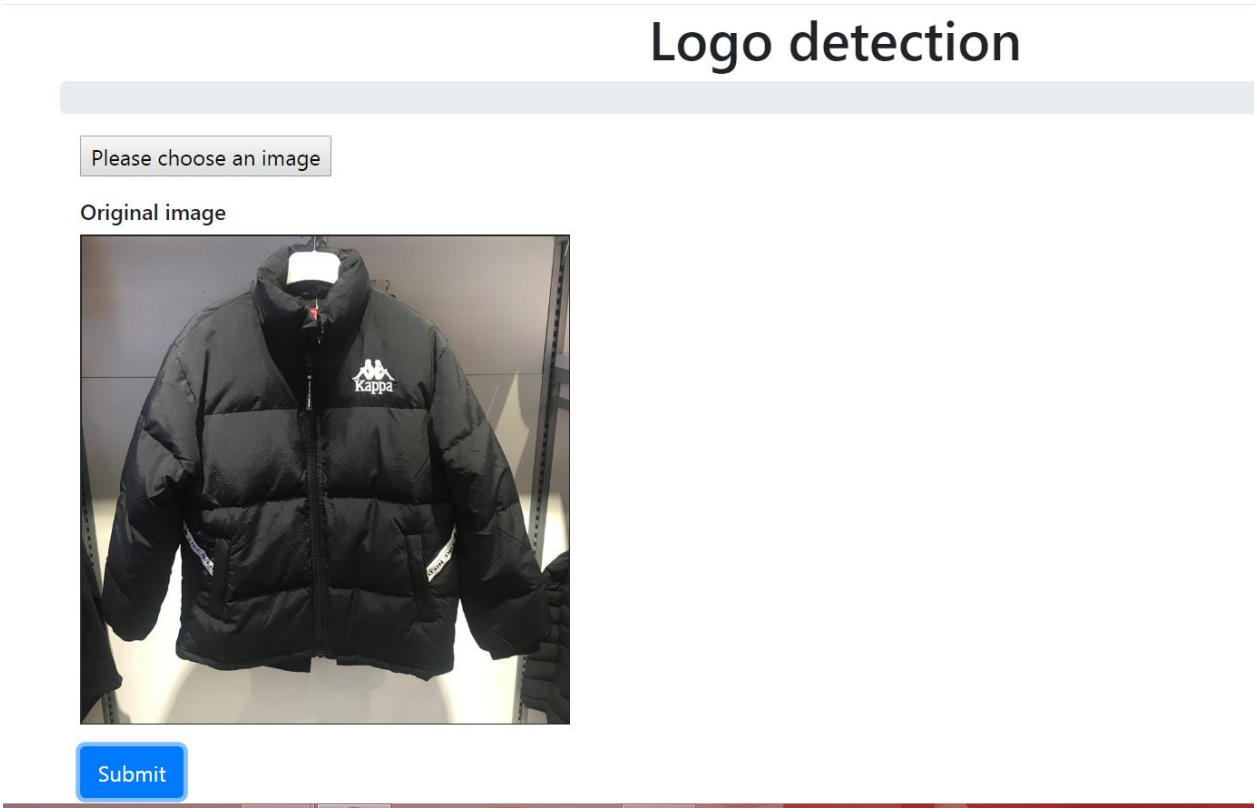

Figure 3-3 Choose an image

Detection result:

Figure 3-4 Detection result

3.2 Video detection

For video detection, find the following statement in *video.py* and change the parenthesis to the path of the video to be detected. When the test is complete, a window appears showing how long each brand is visible in the video.

```
capture=cv2.VideoCapture("nike.mp4")
```

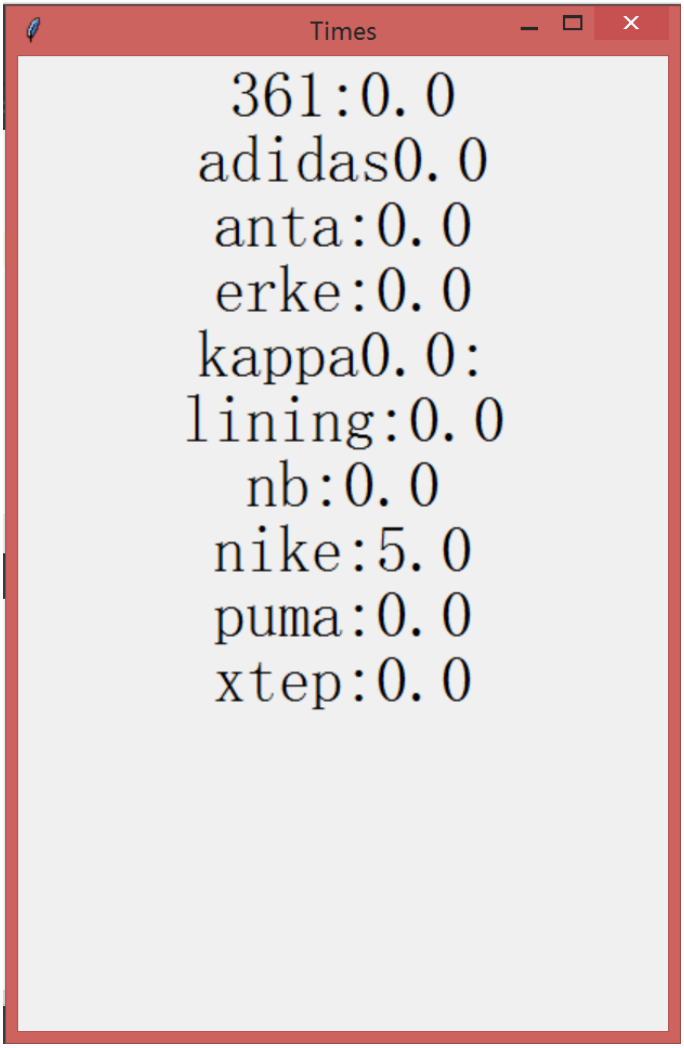

Figure 3-5 How long each brand are visible

3.3 Live camera detection

For real-time camera detection, run *camera.py* to call the camera for live detection. The system will call the camera, then detect the logo in the camera in real-time, and output the confidence.

The system can also send the live stream to an external web page, then detect the logos in the camera in real-time. Running the *server-live_camera.py*, click http://127.0.0.1:5000/, there will be a webpage that shows the live stream. If there is a logo in the camera, the system can detect the category and confidence of the logo.

server-live_camera.py

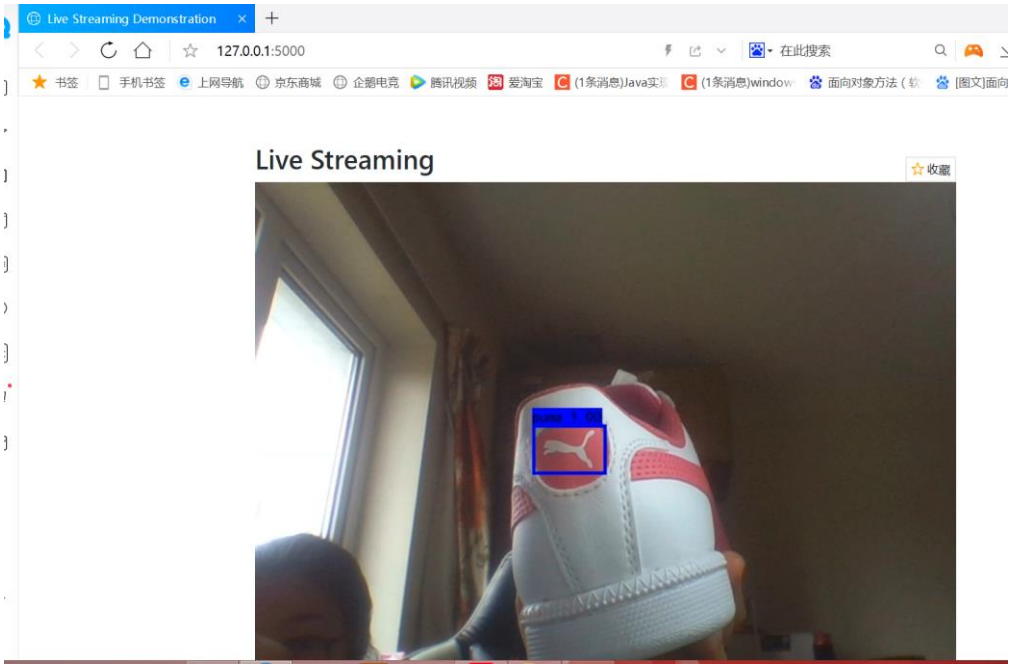

Figure 3-6 Real-time camera detection

Figure 3-7 Real-time camera detection

4 Locate a logo that has never been seen before

For the logo that is not in the data set and has not been trained, the image of the given logo needs to be placed in the target_img folder. The system will match the logo and locate the logo in the video and pictures. As shown in Figure 4-1, the Toyota logo isn't trained before, the system can locate it.

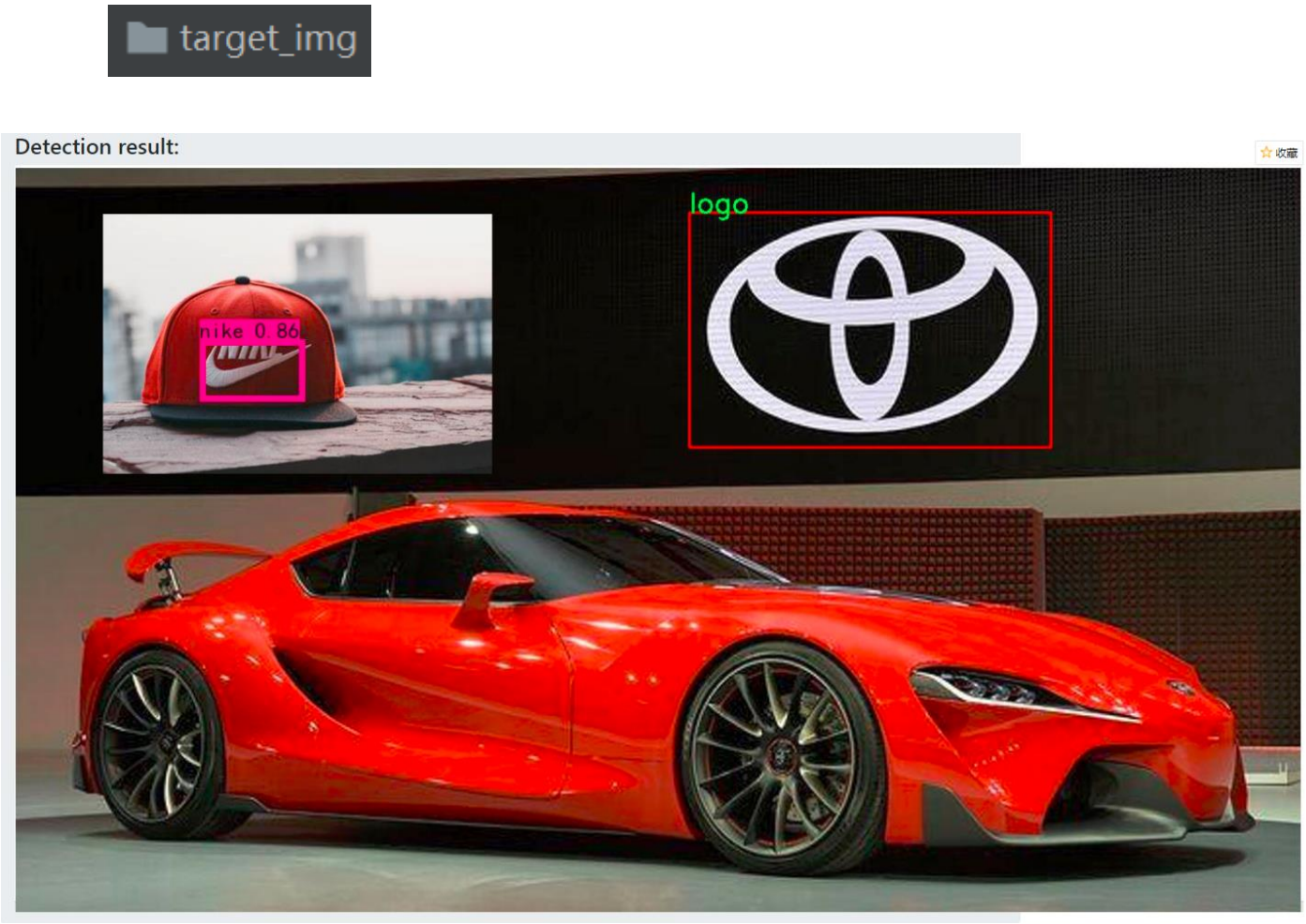

Figure 4-1 Locate a logo that has never been seen before

5 How to train your own Yolov3 model?

First, place the label files of the images in the Annotations file for Voc2007 folder, which is under the Vocdevkit folder, as shown in Figure 5-1. Place the images in the JPEGImages folder. Then run *voc2yolo3.py*, traversal the JPEGImages and Annotations folders to save the names of the images used for training on train.txt.

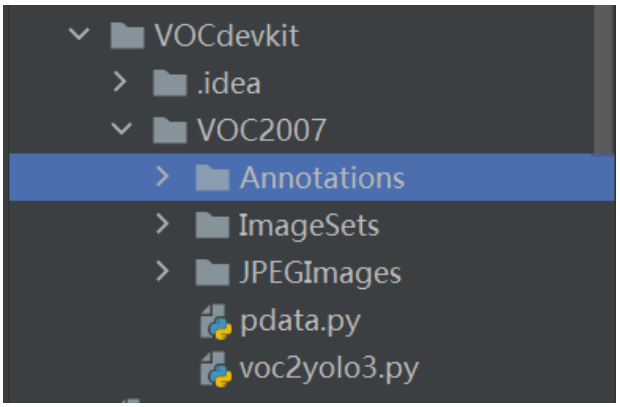

Figure 5-1 VOCdekit folder

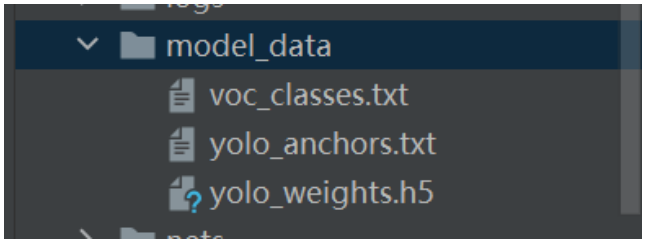

Figure 5-2 model_data folder

Changing the categories in voc_classes.txt to the categories are going to be trained. Then run *voc_annotation.py*, and change the classes to those that are going to be trained. Notice that the classes must correspond to those in voc_classes.txt. Then the 2007_train.txt is generated, each line corresponds to the path of the image and the position of the ground truth.

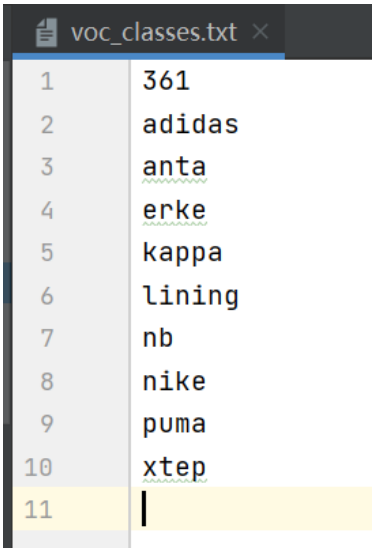

Figure 5-3 voc_classes.txt

```
6  classes = ["361", "adidas", "anta", "erke", "kappa", "lining", "nb", "nike", "puma", "xtep"]
```

Figure 5-4 voc_annotation.py

Start training by running *train.py*. As shown in Figure 5-5, the trained weight will be stored in the logs folder. Changing the model_path in *yolo.py* to the path of the trained weight, and image prediction can be performed.

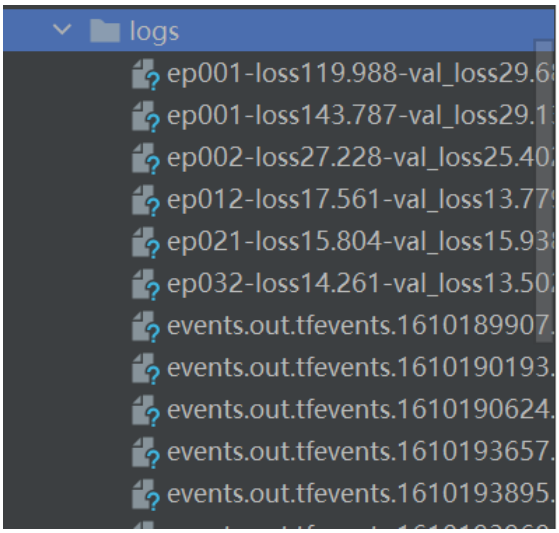

Figure 5-5 Weight

```
91  class YOLO(object):
92      _defaults = {
93          "model_path"        : 'logs/ep032-loss14.261-val_loss13.502.h5',
94          "anchors_path"      : 'model_data/yolo_anchors.txt',
95          "classes_path"      : 'model_data/voc_classes.txt',
96          "score"             : 0.7,
97          "iou"               : 0.1,
98          "max_boxes"         : 100,
99          "model_image_size"  : (416, 416)
100     }
```

Figure 5-6 Model path

6 How to train your own Faster-RCNN model?

First, place the label files of the images in the Annotations file for the Voc2007 folder, which is under the Vocdevkit folder, as shown in Figure 6-1. Place the images in the JPEGImages folder. Then run *voc2yolo3.py*, traversal the JPEGImages and Annotations folders to save the names of the images used for training on train.txt.

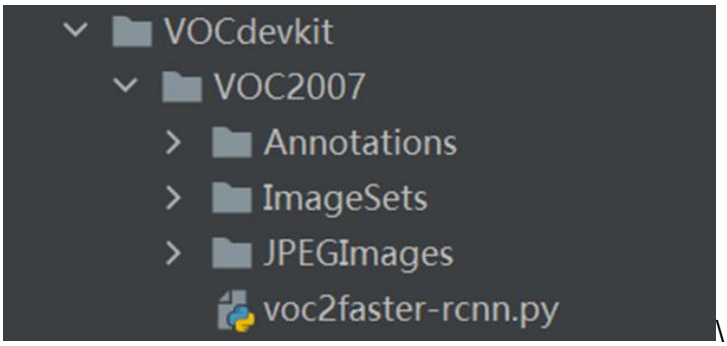

Figure 6-1 VOCdekit folder

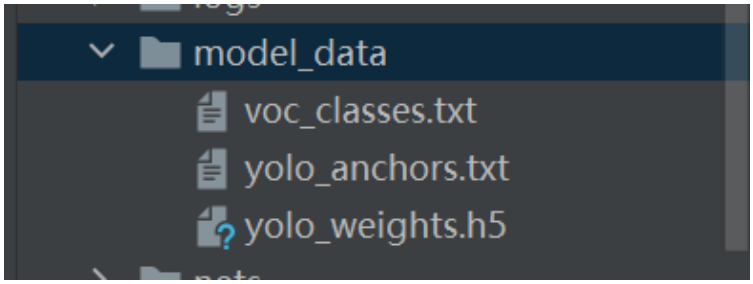

Figure 6-2 model_data folder

Changing the categories in voc_classes.txt to the categories are going to be trained. Then run *voc_annotation.py*, and change the classes to those that are going to be trained. Notice that the classes must correspond to those in voc_classes.txt. Then the 2007_train.txt is generated, each line corresponds to the path of the image and the position of the ground truth.

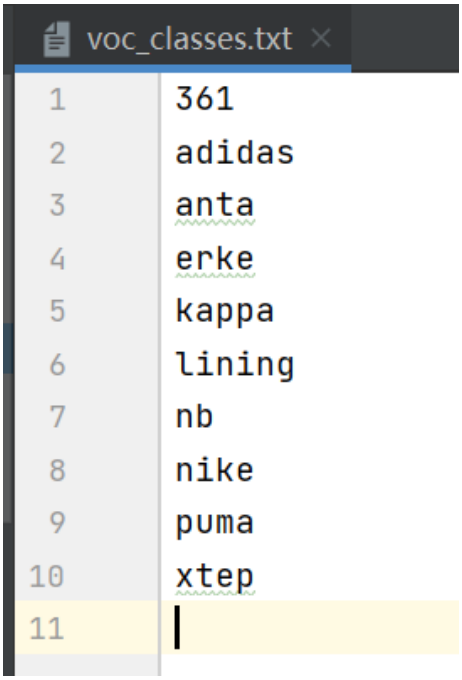

Figure 6-3 voc_classes.txt

```
6    classes = ["361", "adidas", "anta", "erke", "kappa", "lining", "nb", "nike", "puma", "xtep"]
```

Figure 6-4 voc_annotation.py

Start training by running *train.py*. As shown in Figure 6-5, the trained weight will be stored in the logs folder. Changing the model_path in *yolo.py* to the path of the trained weight, and image prediction can be performed. Note that when training Faster-RCNN, the "NUM_CLASSES =" in *train.py* must be changed to "*the number of classes are going to be trained + 1*". For example, if you want to identify 10 brands in this project, then NUM_CLASSES should be changed to 11.

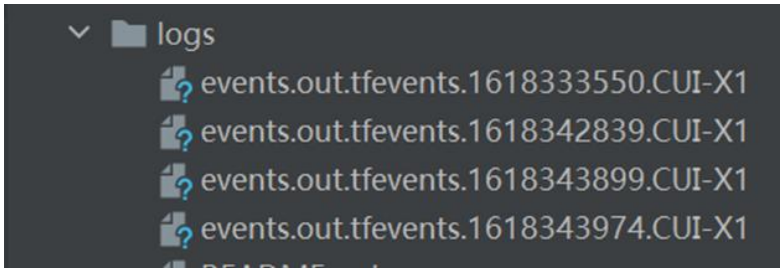

Figure 6-5 Weight

```
class FRCNN(object):
    _defaults = {
        "model_path"    : 'model_data/voc_weights.h5',
        "classes_path"  : 'model_data/voc_classes.txt',
        "confidence"    : 0.5,
        "iou"           : 0.3
    }
```

Figure 6-6 Model path

```
NUM_CLASSES = 11
```

7 How to train your own SSD model?

First, place the label files of the images in the Annotations file for the Voc2007 folder, which is under the Vocdevkit folder, as shown in Figure 7-1. Place the images in the JPEGImages folder. Then run *voc2ssd.py*, traversal the JPEGImages and Annotations folders to save the names of the images used for training on train.txt.

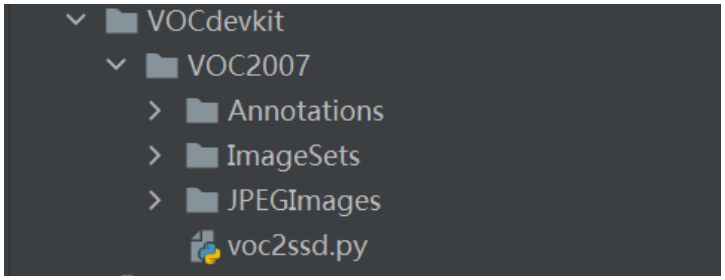

Figure 7-1 VOCdekit folder

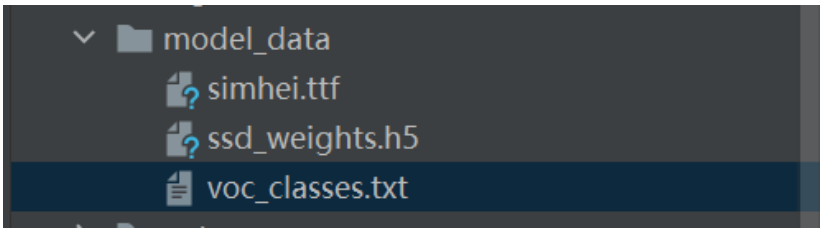

Figure 7-2 model_data folder

Changing the categories in voc_classes.txt to the categories are going to be trained. Then run *voc_annotation.py*, and change the classes to those that are going to be

trained. Notice that the classes must correspond to those in voc_classes.txt. Then the 2007_train.txt is generated, each line corresponds to the path of the image and the position of the ground truth.

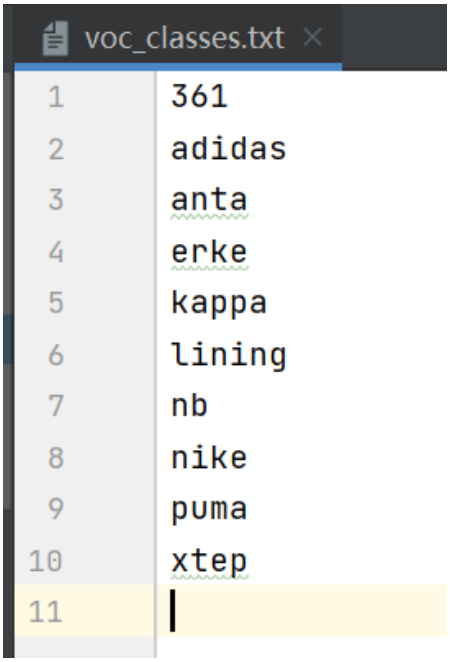

Figure 7-3 voc_classes.txt

```
6    classes = ["361", "adidas", "anta", "erke", "kappa", "lining", "nb", "nike", "puma", "xtep"]
```

Figure 7-4 voc_annotation.py

Start training by running *train.py*. As shown in Figure 6-5, the trained weight will be stored in the logs folder. Changing the model_path in *ssd.py* to the path of the trained weight, and image prediction can be performed. Note that when training Faster-RCNN, the "NUM_CLASSES =" in *train.py* must be changed to "*the number of classes are going to be trained + 1*". For example, if you want to identify 10 brands in this project, then NUM_CLASSES should be changed to 11.

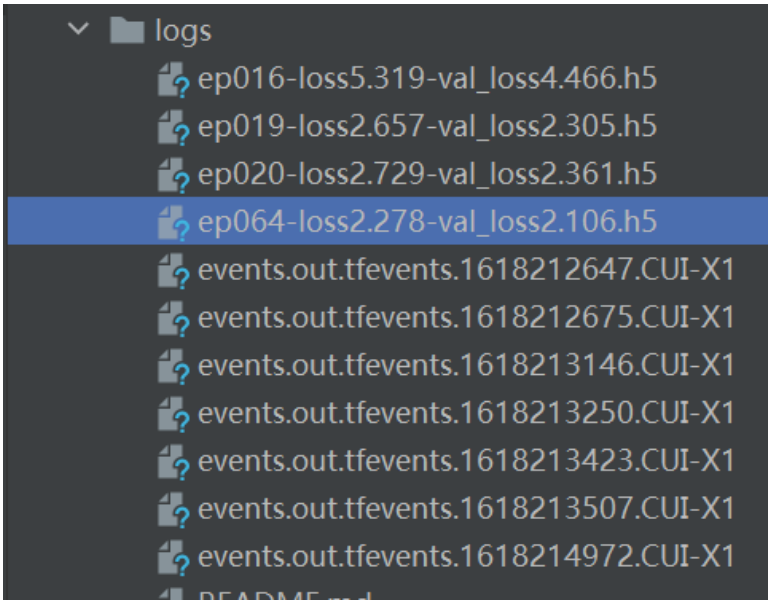

Figure 7-5 SSD weight

```
22  class SSD(object):
23      _defaults = {
24          "model_path"        : 'logs/ep064-loss2.278-val_loss2.106.h5',
25          "classes_path"      : 'model_data/voc_classes.txt',
26          "input_shape"       : (300, 300, 3),
27          "confidence"        : 0.5,
28          "nms_iou"           : 0.45,
29          'anchors_size'      : [30,60,111,162,213,264,315],
30          "letterbox_image"   : False,
31      }
```

Figure 7-6 Model path

```
124     NUM_CLASSES = 11
```

8 Project code

Keras-yolov3 source code:

https://github.com/mochenge/yolo3-serving-keras

https://github.com/qqwweee/keras-yolo3

darknet53.py

(The code which I adapted from other sources)

```
from functools import wraps
from keras.layers import Conv2D, Add, ZeroPadding2D, UpSampling2D, Conc
atenate, MaxPooling2D
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.normalization import BatchNormalization
from keras.regularizers import l2
from utils.utils import cobine_data


@wraps(Conv2D)
def Darknet_basic_Conv2D(*args, **kwargs):
    keys_dict = {}
    keys_dict['kernel_regularizer'] = l2(0.00001)
    if kwargs.get('strides')==(2,2):
        keys_dict['padding'] = 'valid'
    else:
        keys_dict['padding'] = 'same'
    keys_dict.update(kwargs)
    return Conv2D(*args, **keys_dict)

def Darknet_Conv2D_BatchNormal_Leaky_Relu(*args, **kwargs):
    keys_dict = {}
    keys_dict['use_bias'] = False
    keys_dict.update(kwargs)
    return cobine_data(
        Darknet_basic_Conv2D(*args, **keys_dict),
        BatchNormalization(),
        LeakyReLU(0.1))
```

```
def res_darknet_block(out, channels_num, num_layers):
    out = ZeroPadding2D(((1, 0),(1, 0)))(out)
    out = Darknet_Conv2D_BatchNormal_Leaky_Relu(channels_num, (3, 3), s
trides=(2, 2))(out)
    for num in range(num_layers):
        y_out = Darknet_Conv2D_BatchNormal_Leaky_Relu(channels_num//2,
(1,1))(out)
        y_out = Darknet_Conv2D_BatchNormal_Leaky_Relu(channels_num, (3,
3))(y_out)
        out = Add()([out,y_out])
    return out

def darknet53_bodys(x):
    out = Darknet_Conv2D_BatchNormal_Leaky_Relu(32, (3,3))(x)
    out = res_darknet_block(out, 64, 1)
    out = res_darknet_block(out, 128, 2)
    out = res_darknet_block(out, 256, 8)
    feat_11 = out
    out = res_darknet_block(out, 512, 8)
    feat_22 = out
    out = res_darknet_block(out, 1024, 4)
    feat_33 = out
    return feat_11,feat_22,feat_33
```

loss.py

(The code which I adapted from other sources)

```
import numpy as np
from keras.layers.normalization import BatchNormalization
import tensorflow as tf
from keras import backend as ks
from keras.layers.advanced_activations import LeakyReLU, PReLU, ELU
from keras.regularizers import l2
```

```
def yolo_head_loss(feat_nums, anchors_box, total_class, one_channels_sh
ape, jisuan_total_loss=False):
    number_a = len(anchors_box)

    anchors_tensors = ks.reshape(ks.constant(anchors_box), [1, 1, 1, nu
mber_a, 2])


    grids_shapes = ks.shape(feat_nums)[1:3]
    grids_y_shapes = ks.tile(ks.reshape(ks.arange(0, stop=grids_shapes[
0]), [-1, 1, 1, 1]),
                     [1, grids_shapes[1], 1, 1])
    grids_x_shapes = ks.tile(ks.reshape(ks.arange(0, stop=grids_shapes[
1]), [1, -1, 1, 1]),
                     [grids_shapes[0], 1, 1, 1])
    grids = ks.concatenate([grids_x_shapes, grids_y_shapes])
    grids = ks.cast(grids, ks.dtype(feat_nums))


    f_data = ks.reshape(feat_nums, [-
1, grids_shapes[0], grids_shapes[1], number_a, total_class + 5])


    boundingbox_xy_data = (ks.sigmoid(f_data[..., :2]) + grids) / ks.ca
st(grids_shapes[::-1], ks.dtype(f_data))
    boundingbox_wh_data = ks.exp(f_data[..., 2:4]) * anchors_tensors /
ks.cast(one_channels_shape[::-1], ks.dtype(f_data))
    boundingbox_con_score = ks.sigmoid(f_data[..., 4:5])
    boundingbox_class_data = ks.sigmoid(f_data[..., 5:])

    if jisuan_total_loss == True:
        return grids, f_data, boundingbox_xy_data, boundingbox_wh_data
    return boundingbox_xy_data, boundingbox_wh_data, boundingbox_con_sc
ore, boundingbox_class_data



def boxs_iou(b1_boundingbox, b2_boundingbox):

    bounding1_box = ks.expand_dims(b1_boundingbox, -2)
    bounding1_xy_data = bounding1_box[..., :2]
    bounding1_wh_data = bounding1_box[..., 2:4]
    bounding1_wh_part_data = bounding1_wh_data / 2.
```

```
    bounding1_min_data = bounding1_xy_data - bounding1_wh_part_data
    bounding1_max_data = bounding1_xy_data + bounding1_wh_part_data


    bounding2_box = ks.expand_dims(b2_boundingbox, 0)
    bounding2_xy_data = bounding2_box[..., :2]
    bounding2_wh_data = bounding2_box[..., 2:4]
    bounding2_wh_part_data = bounding2_wh_data / 2.
    bounding2_min_data = bounding2_xy_data - bounding2_wh_part_data
    bounding2_max_data = bounding2_xy_data + bounding2_wh_part_data


    p_min_data = ks.maximum(bounding1_min_data, bounding2_min_data)
    p_max_data = ks.minimum(bounding1_max_data, bounding2_max_data)
    p_wh_data = ks.maximum(p_max_data - p_min_data, 0.)
    p_total = p_wh_data[..., 0] * p_wh_data[..., 1]
    bounding1_a_data = bounding1_wh_data[..., 0] * bounding1_wh_data[..
., 1]
    bounding2_a_data = bounding2_wh_data[..., 0] * bounding2_wh_data[..
., 1]
    ious_boxs = p_total / (bounding1_a_data + bounding2_a_data - p_tota
l)

    return ious_boxs



def yolos_total_losss(kywags, anchors, number_labels, i_value=.5, count
_losss=False):
    number_network = len(anchors) // 3


    y_labels_data = kywags[number_network:]
    network_outputs_data = kywags[:number_network]
    if number_network == 3 :
        anchors_layer_m = [[6, 7, 8], [3, 4, 5], [0, 1, 2]]
    else:
        anchors_layer_m = [[3, 4, 5], [1, 2, 3]]

    one_channels_shape = ks.cast(ks.shape(network_outputs_data[0])[1:3]
 * 32, ks.dtype(y_labels_data[0]))
```

```
    grids_layer_s = [ks.cast(ks.shape(network_outputs_data[l])[1:3], ks
.dtype(y_labels_data[0])) for l in range(number_network)]
    losses = 0


    temp = ks.shape(network_outputs_data[0])[0]
    temp_i = ks.cast(temp, ks.dtype(network_outputs_data[0]))


    for i in range(number_network):

        current_layer_m = y_labels_data[i][..., 4:5]

        true_labels_scores = y_labels_data[i][..., 5:]


        grids, feats_data, box_xy_data, box_wh_data = yolo_head_loss(ne
twork_outputs_data[i],
                                                     anchors[anchors_la
yer_m[i]], number_labels, one_channels_shape, calc_loss=True)


        preds_boxs = ks.concatenate([box_xy_data, box_wh_data])


        currnet_layers_masks = tf.TensorArray(ks.dtype(y_labels_data[0]
), dynamic_size=True, size=1)
        current_layer_masks_panduan = ks.cast(current_layer_m, 'bool')


        def wh_bodys(bbox, mask_current):

            ground_boxs = tf.boolean_mask(y_labels_data[i][bbox, ..., 0
:4], current_layer_masks_panduan[bbox, ..., 0])

            ious = boxs_iou(preds_boxs[bbox], ground_boxs)


            first_ious = ks.max(ious, axis=-1)


            mask_current = mask_current.write(bbox, ks.cast(first_ious
< i_value, ks.dtype(ground_boxs)))
```

```
            return bbox + 1, mask_current


        _, currnet_layers_masks = ks.control_flow_ops.while_loop(lambda
 b, *args: b < temp, wh_bodys, [0, currnet_layers_masks])


        currnet_layers_masks = currnet_layers_masks.stack()

        currnet_layers_masks = ks.expand_dims(currnet_layers_masks, -1)


        bb_label_xx_data = y_labels_data[i][..., :2] * grids_layer_s[i]
[:] - grids
        bb_label_ww_data = ks.log(y_labels_data[i][..., 2:4] / anchors[
anchors_layer_m[i]] * one_channels_shape[::-1])


        bb_label_ww_data = ks.switch(current_layer_m, bb_label_ww_data,
 ks.zeros_like(bb_label_ww_data))
        boundingb_losss_value = 2 - y_labels_data[i][..., 2:3] * y_labe
ls_data[i][..., 3:4]

        bounding_xx_losses = current_layer_m * boundingb_losss_value *
ks.binary_crossentropy(bb_label_xx_data, feats_data[..., 0:2],

from_logits=True)
        bounding_ww_losses = current_layer_m * boundingb_losss_value *
0.5 * ks.square(bb_label_ww_data - feats_data[..., 2:4])


        true_loss_total = current_layer_m * ks.binary_crossentropy(curr
ent_layer_m, feats_data[..., 4:5], from_logits=True) + \
                          (1 - current_layer_m) * ks.binary_crossentrop
y(current_layer_m, feats_data[..., 4:5],
                                                                  fro
m_logits=True) * currnet_layers_masks

        label_total_loss = current_layer_m * ks.binary_crossentropy(tru
e_labels_scores, feats_data[..., 5:], from_logits=True)

        bounding_xx_losses = ks.sum(bounding_xx_losses) / temp_i
        bounding_ww_losses = ks.sum(bounding_ww_losses) / temp_i
```

```
        true_loss_total = ks.sum(true_loss_total) / temp_i
        label_total_loss = ks.sum(label_total_loss) / temp_i
        losses += bounding_xx_losses + bounding_ww_losses + true_loss_t
otal + label_total_loss
        if count_losss:
            losses = tf.Print(losses, [losses, bounding_xx_losses, boun
ding_ww_losses, true_loss_total, label_total_loss, ks.sum(currnet_layer
s_masks)],
                              message='loss: ')

    return losses
```

yolo3.py

(The code which I adapted from other sources)

```
from functools import wraps


import tensorflow as tf
from keras import backend as ks
from keras.layers import Conv2D,  UpSampling2D, Concatenate, AveragePoo
ling2D,MaxPooling2D
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.normalization import BatchNormalization
from keras.models import Model, Sequential
from keras.regularizers import l2
from nets.darknet53 import darknet53_bodys
from utils.utils import cobine_data,Image
from keras.layers.advanced_activations import LeakyReLU, PReLU, ELU
from keras.layers.normalization import BatchNormalization



@wraps(Conv2D)
def Darknet_basic_Conv2D(*args, **kwargs):
    keys_dict = {}
    keys_dict['kernel_regularizer'] = l2(0.00001)
    if kwargs.get('strides')==(2,2):
        keys_dict['padding'] = 'valid'
    else:
        keys_dict['padding'] = 'same'
```

```
    keys_dict.update(kwargs)
    return Conv2D(*args, **keys_dict)

def Darknet_Conv2D_BatchNormal_Leaky_Relu(*args, **kwargs):
    keys_dict = {}
    keys_dict['use_bias'] = False
    keys_dict.update(kwargs)
    return cobine_data(
        Darknet_basic_Conv2D(*args, **keys_dict),
        BatchNormalization(),
        LeakyReLU(0.1))

def make_bodys_layers(input, number_filter, output_filter):

    out_x = Darknet_Conv2D_BatchNormal_Leaky_Relu(number_filter, (1,1))
(input)
    out_x = Darknet_Conv2D_BatchNormal_Leaky_Relu(number_filter*2, (3,3
))(out_x)
    out_x = Darknet_Conv2D_BatchNormal_Leaky_Relu(number_filter, (1,1))
(out_x)
    out_x = Darknet_Conv2D_BatchNormal_Leaky_Relu(number_filter*2, (3,3
))(out_x)
    out_x = Darknet_Conv2D_BatchNormal_Leaky_Relu(number_filter, (1,1))
(out_x)


    out_y = Darknet_Conv2D_BatchNormal_Leaky_Relu(number_filter*2, (3,3
))(out_x)
    out_y = Darknet_basic_Conv2D(output_filter, (1,1))(out_y)

    return out_x, out_y


def networks_yolo(in_data_shape, number_anchors, number_classes):

    feat_11,feat_22,feat_33 = darknet53_bodys(in_data_shape)
    darknet53_network = Model(in_data_shape, feat_33)


    x_out, y1_out = make_bodys_layers(darknet53_network.output, 512, nu
mber_anchors*(number_classes+5))
```

```
    x_out = cobine_data(
            Darknet_Conv2D_BatchNormal_Leaky_Relu(256, (1, 1)),
            UpSampling2D(2))(x_out)
    x_out = Concatenate()([x_out, feat_22])

    x_out, y2_out = make_bodys_layers(x_out, 256, number_anchors*(numbe
r_classes+5))

    x_out = cobine_data(
            Darknet_Conv2D_BatchNormal_Leaky_Relu(128, (1,1)),
            UpSampling2D(2))(x_out)
    x_out = Concatenate()([x_out,feat_11])

    x_out, y3_out = make_bodys_layers(x_out, 128, number_anchors*(numbe
r_classes+5))

    return Model(in_data_shape, [y1_out,y2_out,y3_out])


def yolo_head(feat_input, anchors_box, number_classes, one_channels_sha
pe, jisuan_total_loss=False):
    number_a = len(anchors_box)

    anchors_tensors = ks.reshape(ks.constant(anchors_box), [1, 1, 1, nu
mber_a, 2])

    grids_shapes = ks.shape(feat_input)[1:3]
    grids_y_shapes = ks.tile(ks.reshape(ks.arange(0, stop=grids_shapes[
0]), [-1, 1, 1, 1]),
                             [1, grids_shapes[1], 1, 1])
    grids_x_shapes = ks.tile(ks.reshape(ks.arange(0, stop=grids_shapes[
1]), [1, -1, 1, 1]),
                             [grids_shapes[0], 1, 1, 1])
    grids = ks.concatenate([grids_x_shapes, grids_y_shapes])
    grids = ks.cast(grids, ks.dtype(feat_input))

    f_data = ks.reshape(feat_input, [-
1, grids_shapes[0], grids_shapes[1], number_a, number_classes + 5])

    boundingbox_xy_data = (ks.sigmoid(f_data[..., :2]) + grids) / ks.ca
st(grids_shapes[::-1], ks.dtype(f_data))
```

```
    boundingbox_wh_data = ks.exp(f_data[..., 2:4]) * anchors_tensors /
ks.cast(one_channels_shape[::-1],

        ks.dtype(f_data))
    boundingbox_con_score = ks.sigmoid(f_data[..., 4:5])
    boundingbox_class_data = ks.sigmoid(f_data[..., 5:])

    if jisuan_total_loss == True:
        return grids, f_data, boundingbox_xy_data, boundingbox_wh_data
    return boundingbox_xy_data, boundingbox_wh_data, boundingbox_con_sc
ore, boundingbox_class_data


def yolo_correct_layer_boxes(boundingb_xx_data, boundingb_ww_data, inpu
ts_shapes, images_shapes):
    boundingb_yy_data = boundingb_xx_data[..., ::-1]
    boundingb_hh_data = boundingb_ww_data[..., ::-1]

    inputs_shapes = ks.cast(inputs_shapes, ks.dtype(boundingb_yy_data))
    images_shapes = ks.cast(images_shapes, ks.dtype(boundingb_yy_data))

    news_shapes = ks.round(images_shapes * ks.min(inputs_shapes / image
s_shapes))
    offsets_value = (inputs_shapes - news_shapes) / 2. / inputs_shapes
    scales_value = inputs_shapes / news_shapes

    boxs_yx_data = (boundingb_yy_data - offsets_value) * scales_value
    boundingb_hh_data *= scales_value

    boxs_min_iou = boxs_yx_data - (boundingb_hh_data / 2.)
    boxs_max_iou = boxs_yx_data + (boundingb_hh_data / 2.)
    boundingboxs_total = ks.concatenate([
        boxs_min_iou[..., 0:1],
        boxs_min_iou[..., 1:2],
        boxs_max_iou[..., 0:1],
        boxs_max_iou[..., 1:2]
    ])

    boundingboxs_total *= ks.concatenate([images_shapes, images_shapes]
)
    return boundingboxs_total
```

```
def yolo_boundingboxs_value(feat_box, bounding_box, number_labels, inputs_shapes, images_shapes):

    boxs_xy_data, boxs_wh_data, boxs_confidence_score, boxs_class_probs_score = yolo_head(feat_box, bounding_box, number_labels, inputs_shapes)

    bounding_box_total = yolo_correct_layer_boxes(boxs_xy_data, boxs_wh_data, inputs_shapes, images_shapes)

    bounding_box_total = ks.reshape(bounding_box_total, [-1, 4])
    bounding_box_value = boxs_confidence_score * boxs_class_probs_score
    bounding_box_value = ks.reshape(bounding_box_value, [-1, number_labels])
    return bounding_box_total, bounding_box_value


def yolo_test(ff_input,
              bounding_box_a,
              number_labels,
              images_shapes,
              num_boundingb=20,
              values=.6,
              iou_vaule=.5):

    total_num_ff = len(ff_input)

    anchors_masks = [[6, 7, 8], [3, 4, 5], [0, 1, 2]]

    inputs_shapes = ks.shape(ff_input[0])[1:3] * 32
    bounding_box_total = []
    bounding_box_total_value = []

    for i in range(total_num_ff):
        _boundingb, _boundingb_value = yolo_boundingboxs_value(ff_input[i], bounding_box_a[anchors_masks[i]], number_labels, inputs_shapes,
                                                               images_shapes)
        bounding_box_total.append(_boundingb)
        bounding_box_total_value.append(_boundingb_value)

    boundingbs_total = ks.concatenate(bounding_box_total, axis=0)
```

```
    bounding_box_total_value = ks.concatenate(bounding_box_total_value,
 axis=0)

    masks_boundingb = bounding_box_total_value >= values
    boundingb_max_T = ks.constant(num_boundingb, dtype='int32')
    boundingbs_ss = []
    values_ss = []
    labels_ss = []
    for x in range(number_labels):

        labels_boundingbs_total = tf.boolean_mask(boundingbs_total, mas
ks_boundingb[:, x])
        labels_boundingbs_value_toatl = tf.boolean_mask(bounding_box_to
tal_value[:, x], masks_boundingb[:, x])


        nms_index = tf.image.non_max_suppression(
            labels_boundingbs_total, labels_boundingbs_value_toatl, bou
ndingb_max_T, iou_threshold=iou_vaule)


        labels_boundingbs_total = ks.gather(labels_boundingbs_total, nm
s_index)
        labels_boundingbs_value_toatl = ks.gather(labels_boundingbs_val
ue_toatl, nms_index)
        labels_num = ks.ones_like(labels_boundingbs_value_toatl, 'int32
') * x
        boundingbs_ss.append(labels_boundingbs_total)
        values_ss.append(labels_boundingbs_value_toatl)
        labels_ss.append(labels_num)
    boundingbs_ss = ks.concatenate(boundingbs_ss, axis=0)
    values_ss = ks.concatenate(values_ss, axis=0)
    labels_ss = ks.concatenate(labels_ss, axis=0)

    return boundingbs_ss, values_ss, labels_ss
```

yolo.py

(The code which I wrote and adapted from other sources)

```
import os
import numpy as np
import copy
import colorsys
from timeit import default_timer as timer
from keras import backend as K
from keras.models import load_model,save_model,model_from_json,clone_mo
del
from keras.layers import Input,core,average
from PIL import Image, ImageFont, ImageDraw, FontFile, SgiImagePlugin
from nets.yolo3 import networks_yolo,yolo_test
from utils.utils import get_boundingb_img
from io import BytesIO
import cv2


def nms(bounding_boxs, thresh):

    x11 = bounding_boxs[:, 0]
    y11 = bounding_boxs[:, 1]
    x22 = bounding_boxs[:, 2]
    y22 = bounding_boxs[:, 3]


    areasq = (y22 - y11 + 1) * (x22 - x11 + 1)


    scoresq = bounding_boxs[:, 4]
    index = scoresq.argsort()[::-1]


    resq = []

    while index.size > 0:
        iq = index[0]
        resq.append(iq)

        x11q = np.maximum(x11[iq], x11[index[1:]])
        y11q = np.maximum(y11[iq], y11[index[1:]])
```

```
        x22q = np.minimum(x22[iq], x22[index[1:]])
        y22q = np.minimum(y22[iq], y22[index[1:]])

        wq = np.maximum(0, x22q - x11q + 1)
        hq = np.maximum(0, y22q - y11q + 1)

        overlapsq = wq * hq
        iousq = overlapsq / (areasq[iq] + areasq[index[1:]] - overlapsq
) # index[1:]从下标 1 开始取到列表结束  最高分的面积加其余的面积

        idxq = np.where(iousq <= thresh)[0]
        index = index[idxq + 1]

    return resq

def mathc_img(image,Target,values_score):

    target_list = os.listdir(Target)
    for target_img_path in target_list:
        target_img = cv2.imread('./target_img/' + target_img_path)
        channels, width, height = target_img.shape[::-1]
        temp = cv2.matchTemplate(image,target_img,cv2.TM_CCOEFF_NORMED)
        res_value = values_score
        ll = np.where(temp >= res_value)
        Nims = len(ll[::-1][0])
        box = np.empty([Nims, 5])
        for i, point in enumerate(zip(*ll[::-1])):
            a = np.asarray([point[0],point[1], point[0] + width, point[
1] + height, 0.7])
            box[i] = a
        new_box = nms(box, 0.7)
        for x in new_box:
            point = (ll[1][x], ll[0][x])
            cv2.rectangle(image, point, (point[0] + width, point[1] + h
eight), (255, 0, 0), 2)
            cv2.putText(image,'logo',point,cv2.FONT_HERSHEY_SIMPLEX, 1,
 (0, 255, 0), 2)

    return image


class YOLO(object):
```

```
    _key_values = {
        "net_dir"        : 'logs/ep032-loss14.261-val_loss13.502.h5',
        "anchors_path"      : 'model_data/yolo_anchors.txt',
        "classes_path"      : 'model_data/voc_classes.txt',
        "score"             : 0.5,
        "iou"               : 0.1,
        "max_boxes"         : 100,
        "input_img_shape"   : (416, 416)
    }


    @classmethod
    def get_key_values(cls, x):
        if x in cls._key_values:
            return cls._key_values[x]
        else:
            return "name '" + x + "'"


    def __init__(self, **kwargs):
        self.__dict__.update(self._key_values)
        self.labels_name = self._give_labels()
        self.boundingbox_a = self._give_boundingbox()
        self.sess = K.get_session()
        self.boundingbox, self.values, self.labels = self.creat_model()
        self.band_dict = {
        '361': 0.0,
        'adidas': 0.0,
        'anta': 0.0,
        'erke': 0.0,
        'kappa': 0.0,
        'lining': 0.0,
        'nb': 0.0,
        'nike': 0.0,
        'puma': 0.0,
        'xtep': 0.0,
    }



    def _give_labels(self):
        class_dir = os.path.expanduser(self.classes_path)
```

```
        with open(class_dir) as ff:
            labels_number = ff.readlines()
        labels_number = [x.strip() for x in labels_number]
        return labels_number


    def _give_boundingbox(self):
        ancor_dir = os.path.expanduser(self.anchors_path)
        with open(ancor_dir) as file:
            ancor_num = file.readline()
        ancor_num = [float(i) for i in ancor_num.split(',')]
        return np.asarray(ancor_num).reshape(-1, 2)


    def creat_model(self):
        net_path = os.path.expanduser(self.net_dir)
        assert net_path.endswith('.h5'), 'Keras model or weights must b
e a .h5 file.'


        number_anchor = len(self.boundingbox_a)
        number_class = len(self.labels_name)


        try:
            self.networks = load_model(net_path, compile=False)
        except:
            self.networks = networks_yolo(Input((None,None, 3)), number
_anchor//3, number_class)
            self.networks.load_weights(self.net_dir)
        else:
            assert self.networks.layers[-1].output_shape[-1] == \
                number_anchor/len(self.networks.output) * (number_class
 + 5), \
                '...'

        print('{}'.format(net_path))


        color_value = [(i / len(self.labels_name), 1., 1.)
                       for i in range(len(self.labels_name))]
        self.convert_img = list(map(lambda temp: colorsys.hsv_to_rgb(*t
emp), color_value))
```

```
        self.convert_img = list(
            map(lambda temp: (int(temp[0] * 255), int(temp[1] * 255), i
nt(temp[2] * 255)),
                self.convert_img))


        np.random.seed(2421)
        np.random.shuffle(self.convert_img)
        np.random.seed(None)

        self.in_img_size = K.placeholder((2, ))

        box_value, score_value, class_value = yolo_test(self.networks.o
utput, self.boundingbox_a,
                number_class, self.in_img_size, num_boundingb = self.ma
x_boxes,
                values = self.score, iou_vaule = self.iou)
        return box_value, score_value, class_value


    def detecter_images(self, img):
        temp_img_shape = (self.input_img_shape[1],self.input_img_shape[
0])
        boundingbox_img = get_boundingb_img(img, temp_img_shape)
        img_coo= np.array(boundingbox_img, dtype='float32')
        img_coo /= 255.
        img_coo = np.expand_dims(img_coo, 0)  # Add batch dimension.

        count_boundingbox, count_values, count_labels = self.sess.run(
            [self.boundingbox, self.values, self.labels],
            feed_dict={
                self.networks.input: img_coo,
                self.in_img_size: [img.size[1], img.size[0]],
            })

        print('find {} box for {}'.format(len(count_boundingbox), 'pict
ure'))
        font = ImageFont.truetype(font='font/simhei.ttf',
                    size=np.floor(3e-
2 * img.size[1] + 0.5).astype('int32'))
        t_value = (img.size[0] + img.size[1]) // 300
```

```
        set_out_classes = count_labels
        for i in set(set_out_classes):
            set_predicted_class = self.labels_name[i]
            self.band_dict[str(set_predicted_class)] += 1

        for i, c in list(enumerate(count_labels)):
            pred_labels = self.labels_name[c]
            boundingboxs = count_boundingbox[i]
            values = count_values[i]

            tops_value, lefts_value, bottoms_value, rights_value = boun
dingboxs
            tops_value = tops_value - 5
            lefts_value = lefts_value - 5
            bottoms_value = bottoms_value + 5
            rights_value = rights_value + 5

            tops_value = max(0, np.floor(tops_value + 0.5).astype('int3
2'))
            lefts_value = max(0, np.floor(lefts_value + 0.5).astype('in
t32'))
            bottoms_value = min(img.size[1], np.floor(bottoms_value + 0
.5).astype('int32'))
            rights_value = min(img.size[0], np.floor(rights_value + 0.5
).astype('int32'))


            out_labels = '{} {:.2f}'.format(pred_labels, values)
            To_img = ImageDraw.Draw(img)
            labels_shape = To_img.textsize(out_labels, font)
            out_labels = out_labels.encode('utf-8')

            if tops_value - labels_shape[1] >= 0:
                txt_draw = np.array([lefts_value, tops_value - labels_s
hape[1]])
            else:
                txt_draw = np.array([lefts_value, tops_value + 1])

            for i in range(t_value):
                To_img.rectangle(
                    [lefts_value + i, tops_value + i, rights_value - i,
 bottoms_value - i],
```

```
                    outline=self.convert_img[c])
            To_img.rectangle(
                [tuple(txt_draw), tuple(txt_draw + labels_shape)],
                fill=self.convert_img[c])
            To_img.text(txt_draw, str(out_labels,'UTF-
8'), fill=(0, 0, 0), font=font)
            del To_img

        r_image = np.asarray(img,dtype=np.uint8)
        target_path = './target_img'
        try:
            co_img = mathc_img(r_image, target_path, 0.7)
        except:
            co_img = r_image
        co_img = Image.fromarray(co_img)
        im = BytesIO()
        co_img.save(im, "jpeg")
        return im, img

    def close_session(self):
        self.sess.close()
```

train.py

(The code which I adapted from other sources)

```
import numpy as np
import os
import tensorflow as tf
from keras.callbacks import TensorBoard, ModelCheckpoint, ReduceLROnPla
teau, EarlyStopping
from nets.yolo3 import networks_yolo
from nets.loss import yolos_total_losss
import keras.backend as K
from keras.layers import Lambda
from keras.optimizers import Adam
from utils.utils import give_rand_dataset
```

```
from keras.models import load_model,save_model,model_from_json,clone_mo
del
from keras.layers import Input,core,average
from PIL import Image, ImageFont, ImageDraw, FontFile, SgiImagePlugin
from keras.models import Model
from keras.backend.tensorflow_backend import set_session


def give_total_labels(label_dir):

    with open(label_dir) as file:
        labels_numbers = file.readlines()
    labels_numbers = [x.strip() for x in labels_numbers]
    return labels_numbers

def give_total_boundingbox(ancor_dir):

    with open(ancor_dir) as file:
        ancor_num = file.readline()
    ancor_num = [float(x) for x in ancor_num.split(',')]
    return np.asarray(ancor_num).reshape(-1, 2)

def give_dataset_create(nums, bs, in_shapes, ancors, number_classes):

    num = len(nums)
    x = 0
    while True:
        img_datasets = []
        boxs_dataset = []
        for b in range(bs):
            if x == 0:
                np.random.shuffle(nums)
            imgs, boxs = give_rand_dataset(nums[x], in_shapes, random=T
rue)
            img_datasets.append(imgs)
            boxs_dataset.append(boxs)
            x = (x+1) % num
        img_datasets = np.array(img_datasets)
        boxs_dataset = np.array(boxs_dataset)
        labels = preprocess_true_boxes(boxs_dataset, in_shapes, ancors,
 number_classes)
        yield [img_datasets, *labels], np.zeros(bs)
```

```
def preprocess_true_boxes(ground_boundingbox, in_shapes, ancors, number
_labels):

    assert (ground_boundingbox[..., 4]<number_labels).all(), '.'

    number_net = len(ancors)//3
    if number_net ==3 :
        boundingbox_mask = [[6,7,8], [3,4,5], [0,1,2]]
    else:
        boundingbox_mask = [[3,4,5], [1,2,3]]

    ground_boundingboxs = np.array(ground_boundingbox, dtype='float32')
    in_shapes = np.array(in_shapes, dtype='int32') # 416,416

    boundingbox_xx = (ground_boundingboxs[..., 0:2] + ground_boundingbo
xs[..., 2:4]) // 2
    boundingbox_ww = ground_boundingboxs[..., 2:4] - ground_boundingbox
s[..., 0:2]

    ground_boundingboxs[..., 0:2] = boundingbox_xx/in_shapes[::-1]
    ground_boundingboxs[..., 2:4] = boundingbox_ww/in_shapes[::-1]


    m_temp = ground_boundingboxs.shape[0]

    grids_shapes = [in_shapes//{0:32, 1:16, 2:8}[l] for l in range(numb
er_net)]

    out_data = [np.zeros((m_temp,grids_shapes[l][0],grids_shapes[l][1],
len(boundingbox_mask[l]),5+number_labels),
        dtype='float32') for l in range(number_net)]

    ancors = np.expand_dims(ancors, 0)
    ancors_max_num = ancors / 2.
    ancors_min_num = -ancors_max_num

    test_mask = boundingbox_ww[..., 0]>0

    for x in range(m_temp):
```

```
        w_h = boundingbox_ww[x, test_mask[x]]
        if len(w_h)==0: continue

        w_h = np.expand_dims(w_h, -2)
        boxs_max_num = w_h / 2.
        boxs_min_num = -boxs_max_num


        insect_min = np.maximum(boxs_min_num, ancors_min_num)
        insect_max = np.minimum(boxs_max_num, ancors_max_num)
        insect_w_h = np.maximum(insect_max - insect_min, 0.)
        insect_area = insect_w_h[..., 0] * insect_w_h[..., 1]
        boxs_area = w_h[..., 0] * w_h[..., 1]
        ancor_area = ancors[..., 0] * ancors[..., 1]
        ious_bos = insect_area / (boxs_area + ancor_area - insect_area)

        best_ancor = np.argmax(ious_bos, axis=-1)

        for q, w in enumerate(best_ancor):
            for e in range(number_net):
                if w in boundingbox_mask[e]:

                    i = np.floor(ground_boundingboxs[x,q,0]*grids_shape
s[e][1]).astype('int32')
                    j = np.floor(ground_boundingboxs[x,q,1]*grids_shape
s[e][0]).astype('int32')

                    k = boundingbox_mask[e].index(w)
                    c = ground_boundingboxs[x,q, 4].astype('int32')
                    out_data[e][x, j, i, k, 0:4] = ground_boundingboxs[
x,q, 0:4]
                    out_data[e][x, j, i, k, 4] = 1
                    out_data[e][x, j, i, k, 5+c] = 1

    return out_data


configs = tf.ConfigProto()
configs.gpu_options.allocator_type = 'BFC'
configs.gpu_options.per_process_gpu_memory_fraction = 0.7
configs.gpu_options.allow_growth = True
```

```
set_session(tf.Session(config=configs))


if __name__ == "__main__":

    xml_dir = '2007_train.txt'

    class_dir = 'model_data/voc_classes.txt'
    ancor_dir = 'model_data/yolo_anchors.txt'

    model_stat_dir = 'model_data/ep021-loss15.804-val_loss15.938.h5'

    class_names = give_total_labels(class_dir)
    anchors_box = give_total_boundingbox(ancor_dir)

    number_labels = len(class_names)
    number_box = len(anchors_box)

    loger_data_path = 'logs/'

    inputs_shapes = (416,416)


    K.clear_session()


    imgs_inputs = Input((None, None, 3))
    heigh, width = inputs_shapes


    print(' {}  {}'.format(number_box, number_labels))
    model_train = networks_yolo(imgs_inputs, number_box//3, number_labe
ls)


    print(' {}.'.format(model_stat_dir))
    model_train.load_weights(model_stat_dir, by_name=True, skip_mismatc
h=True)


    labels = [Input(shape=(heigh//{0:32, 1:16, 2:8}[l], width//{0:32, 1
:16, 2:8}[l], \
        number_box//3, number_labels+5)) for l in range(3)]
```

```
    loss_input = [*model_train.output, *labels]
    total_loss = Lambda(yolos_total_losss, output_shape=(1,), name='yol
o_loss',
        arguments={'anchors': anchors_box, 'num_classes': number_labels
, 'ignore_thresh': 0.5})(loss_input)

    net = Model([model_train.input, *labels], total_loss)


    frzen_number_layer = 184
    for i in range(frzen_number_layer): model_train.layers[i].trainable
 = False
    print(' {} {} '.format(frzen_number_layer, len(model_train.layers))
)


    logging_temp = TensorBoard(log_dir=loger_data_path)
    checkpoint_temp = ModelCheckpoint(loger_data_path + 'ep{epoch:03d}-
loss{loss:.3f}-val_loss{val_loss:.3f}.h5',
        monitor='val_loss', save_weights_only=True, save_best_only=Fals
e, period=1)
    learning_rate = ReduceLROnPlateau(monitor='val_loss', factor=0.5, p
atience=3, verbose=1)
    early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, pat
ience=10, verbose=1)


    test_dataset_number = 0.1
    with open(xml_dir) as f:
        ff = f.readlines()
    np.random.seed(10101)
    np.random.shuffle(ff)
    np.random.seed(None)
    number_test = int(len(ff)*test_dataset_number)
    number_training = len(ff) - number_test


    if True:
        net.compile(optimizer=Adam(lr=1e-3), loss={
            'yolo_loss': lambda y_true, y_pred: y_pred})
```

```
        batch_size = 3
        print('{}  {}  {}.'.format(number_training, number_test, batch_
size))
        net.fit_generator(give_dataset_create(ff[:number_training], bat
ch_size, imgs_inputs, anchors_box, number_labels),
                steps_per_epoch=max(1, number_training//batch_size),
                validation_data=give_dataset_create(ff[number_training:
], batch_size, imgs_inputs, anchors_box, number_labels),
                validation_steps=max(1, number_test//batch_size),
                epochs=50,
                initial_epoch=0,
                callbacks=[logging_temp, checkpoint_temp, learning_rate
, early_stopping])
        net.save_weights(loger_data_path + 'trained_weights_stage_1.h5'
)

    for i in range(frzen_number_layer): model_train.layers[i].trainable
 = True
    if True:
        net.compile(optimizer=Adam(lr=1e-4), loss={
            'yolo_loss': lambda y_true, y_pred: y_pred})

        batch_size = 2
        print('{} {} {}.'.format(number_training, number_test, batch_si
ze))
        net.fit_generator(give_dataset_create(ff[:number_training], bat
ch_size, imgs_inputs, anchors_box, number_labels),
                steps_per_epoch=max(1, number_training//batch_size),
                validation_data=give_dataset_create(ff[number_training:
], batch_size, imgs_inputs, anchors_box, number_labels),
                validation_steps=max(1, number_test//batch_size),
                epochs=100,
                initial_epoch=50,
                callbacks=[logging_temp, checkpoint_temp, learning_rate
, early_stopping])
        net.save_weights(loger_data_path + 'last1.h5')
```

serve.py

(The code which I wrote)

```
import tornado
from tornado.options import define, options
import tornado.ioloop
import tornado.options
import tornado.httpserver
import tornado.web
import os, json
from yolo import YOLO
from PIL import Image
from io import BytesIO
import base64

yolo_net = YOLO()
define("port", default=8000, help="run on the given port", type=int)



class Index_number(tornado.web.RequestHandler):
    def get(self):
        # self.write(json.dumps(TARGET_DATAS["list"]))
        self.render("index.html")
    def post(self):
        images = Image.open(BytesIO(base64.b64decode(self.get_argument(
"img").split("base64,")[1])))
        im, img = yolo_net.detecter_images(images)
        images = base64.b64encode(im.getvalue()).decode()
        self.write(images)


def main():
    tornado.options.parse_command_line()
    html_servers = tornado.httpserver.HTTPServer(
        tornado.web.Application(
            handlers=((r"/", Index_number),),
            template_path=os.path.join(os.path.dirname(__file__), "temp
lates"),
```

```
            static_path=os.path.join(os.path.dirname(__file__), "static
s"),
            debug=True,
        ))
    print("Click to enter http://localhost:%s" % options.port)
    print("sign out Ctrl + C")
    html_servers.listen(options.port)
    tornado.ioloop.IOLoop.current().start()


if __name__ == "__main__":
    main()
```

video.py

(The code which I wrote)

```
from keras.layers import Input
from yolo import YOLO
from PIL import Image
import numpy as np
import cv2
import time
import tkinter as tk
from tkinter import Label

yolo_net = YOLO()

capture_frame = cv2.VideoCapture("img/ai1.mp4")
fourcc_format = cv2.VideoWriter_fourcc(*'XVID')
output_frame = cv2.VideoWriter('output.avi',fourcc_format, 24.0, (600,6
00))
frames_per_second = 0.0
while (True):
    time1 = time.time()

    ref, frame = capture_frame.read()
    if ref == True:
```

```
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        frame = Image.fromarray(np.uint8(frame))
        im, img = yolo_net.detecter_images(frame)
        frame = np.array(img)

        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        frame = cv2.resize(frame,(600,600))
        frames_per_second = (frames_per_second + (1. / (time.time() - t
ime1))) / 2
        print("frames_per_second= %.2f" % (frames_per_second))
        frame = cv2.putText(frame, "frames_per_second= %.2f" % (frames_
per_second), (0, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 244), 1)

        output_frame.write(frame)
        cv2.imshow("aaaa", frame)
        keys = cv2.waitKey(1) & 0xff
        if keys == 27:
            capture_frame.release()
            break
    else:
        break
capture_frame.release()
yolo_net.close_session()
cv2.destroyAllWindows()
brands_list = (yolo_net.band_dict[k]//24.0 for k in yolo_net.band_dict.
keys())
brands_list = list(brands_list)
window = tk.Tk()
window.title('Times')
window.geometry('400x600')
Label(text='361:{}\n'
           'adidas:{}\n'
           'anta:{}\n'
           'erke:{}\n'
           'kappa:{}\n'
           'lining:{}\n'
           'nb:{}\n'
           'nike:{}\n'
           'puma:{}\n'
```

```
            'xtep:{}\n'.format(brands_list[0],brands_list[1],brands_list
[2],brands_list[3],brands_list[4],brands_list[5],brands_list[6],brands_
list[7],brands_list[8],brands_list[9])
      ,font=('宋体',30)).pack()
tk.mainloop()
```

camera.py

(The code which I wrote and adapted from other sources)

```
from keras.layers import Input
from yolo import YOLO
from PIL import Image
import numpy as np
import cv2
import time
yolo_net = YOLO()

capture_frame=cv2.VideoCapture(0)

frames_per_second = 0.0
while(True):
    time1 = time.time()
    ref, frame = capture_frame.read()
    frame = cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
    frame = Image.fromarray(np.uint8(frame))

    im, img = yolo_net.detecter_images(frame)
    frame = np.asarray(img)
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

    frames_per_second = (frames_per_second + (1./(time.time()-
time1)) ) / 2
    print("frames_per_second= %.2f"%(frames_per_second))
    frame = cv2.putText(frame, "frames_per_second= %.2f"%(frames_per_se
cond), (0, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 1)

    cv2.imshow("aaaa",frame)
```

```
    keys = cv2.waitKey(0) & 0xff
    if keys == 12:
        capture_frame.release()
        break

yolo_net.close_session()
```

serve-live_camera.py

https://www.cnblogs.com/arkenstone/p/7159615.html

(The code which I wrote and adapted from other sources)

```
import tornado
from tornado.options import define, options
import tornado.ioloop
import tornado.options
import tornado.httpserver
import tornado.web
import os, json
from yolo import YOLO
from PIL import Image
from io import BytesIO
import base64
import cv2
import numpy as np
import time

yolo = YOLO()

from flask import Flask, render_template, Response
app = Flask(__name__)



app = Flask(__name__)
camera = cv2.VideoCapture(0)
```

```
def gen_frames():
    fps = 0.0
    while True:
        t1 = time.time()
        success, frame = camera.read()
        if not success:
            break
        else:
            frame = Image.fromarray(np.uint8(frame))
            img1, image = yolo.detecter_images(frame)
            frame = np.array(image)
            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-
Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(gen_frames(), mimetype='multipart/x-mixed-
replace; boundary=frame')


@app.route('/')
def index():
    """Video streaming home page."""
    return render_template('index11.html')


if __name__ == '__main__':
    app.run(debug=True)
```

index.html

(The code which I wrote and adapted from other sources)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <title></title>
    <link rel="stylesheet" href="/static/css/bootstrap.min.css">
    <script type="application/javascript" src="/static/js/jquery-
3.3.1.min.js"></script>
    <script type="application/javascript" src="/static/js/bootstrap.min
.js"></script>
</head>
<body>
<h1 align="center">Logo detection </h1>
<nav aria-label="breadcrumb ">
    <ol class="breadcrumb container">
        <li class="breadcrumb-item active" aria-current="page"></li>
    </ol>
</nav>
<div class="container">
    <form action="javascript:void(0);">
        <div class="form-group">
            <input type="button" id="loadFileXml" value="Please choose
an image" onclick="document.getElementById('exampleFormControlFile1').c
lick();" />
            <input type="file"  class="form-control-
file" id="exampleFormControlFile1" name="img"   style="display:none;"
/>

        </div>
        <div class="form-group">
            <fieldset class="h6">Original image</fieldset>
            <div style="width: 360px;height: 360px;border-
style: solid; border-width:1px;">
                <img class="pull-right" src="" alt="" id="show"
                     style="width: 100%;height: 100%">
            </div>

        </div>
        <div class="form-group">
            <button type="submit" class="btn btn-
primary">Submit</button>
        </div>
    </form>

    <div class="breadcrumb">
        <h4>Detection result: <span id="resp_text"></span></h4>
```

```
        <div class="content">
            <img src="" id="resp_img" alt="">
        </div>
    </div>


</div>
<script>
    var img = "";
    $('#exampleFormControlFile1').bind('change', function (e) {
        var files = this.files;
        if (files.length) {
            checkFile(this.files);
        }
    });


    function checkFile(files) {
        var file = files[0];
        var reader = new FileReader();
        // show refers to <div id='show'></div>, use to show the preview
        if (!/image\/\w+/.test(file.type)) {
            show.innerHTML = "Make sure the file is an image";
            return false;
        }
        // onload is Asynchronous Operation
        reader.onload = function (e) {
            $("#show").attr("src", e.target.result);
            img = e.target.result
        };
        reader.readAsDataURL(file);
    }

    $("button").click(function () {
        $("#resp_imgs>img").remove();
        $("#resp_detail>div").remove();

        $.ajax({
            url: "/",
            type: "post",
            dataType: "text",
            data: {
```

```
                "img": img
            }, success: function (resp) {
                // console.log(resp);
                $("#resp_img").attr("src",""+res
p)


            }
        })


    });

</script>


</body>

</html>
```

Index11.thml

(The code which I wrote and adapted from other sources)

```
<!doctype html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.1.3/css/bootstrap.min.css"
          integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" cross
origin="anonymous">

    <title>Live Streaming Demonstration</title>
</head>
```

```
<body>
<div class="container">
    <div class="row">
        <div class="col-lg-8  offset-lg-2">
            <h3 class="mt-5">Live Streaming</h3>
            <img src="{{ url_for('video_feed') }}" width="100%">
        </div>
    </div>
</div>
</body>

</html>
```

Reference

[1] GitHub. 2021. *mochenge/yolo3-serving-keras*. [online] Available at: <https://github.com/mochenge/yolo3-serving-keras> [Accessed 6 May 2021].

[2] GitHub. 2021. *qqwweee/keras-yolo3*. [online] Available at: <https://github.com/qqwweee/keras-yolo3> [Accessed 6 May 2021].

[3] Cnblogs.com. 2021. *Use flask to output OpenCV real-time video stream to the browser - Arkenstone*. [online] Available at: <https://www.cnblogs.com/arkenstone/p/7159615.html> [Accessed 6 May 2021].