Nibiru Studio
VR Object Recognition
Developer Guide

V1.0.0

Nanjing Ruiyue (Nibiru) Technology Co., Ltd.

May 2018 - Nanjing

Legal Notices

Nanjing Ruiyue (Nibiru) Technology Co., Ltd.
Tel: +86 (025) 89635828
Address: Room 405, Building 4, Chuqiaocheng, No.57, Andemen Street, Yuhuatai District, Nanjing, Jiangsu Province, PRC
E-mail： support@inibiru.com
Web: http://dev.inibiru.com
Technical Support QQ Group: 128275865
Any suggestions are welcomed.

Content

1 Introduction to Nibiru Studio

Nibiru Studio is a VR app development framework based on OpenGL ES graphics interface. With the advanced Nibiru VR OS, it has a clear code structure and perfect functions and work flow, providing users with convenient VR app development APIs and tools. It features as follows:

(1) Professional graphic rendering service optimizes 3D graphic rendering process;

(2) Development mode similar with Android makes it easy for android developers to use;

(3) Provide UI controls for VR and optimize their visual experience in the space dimension;

(4) Seamless connection to Nibiru minimizes the rendering latency;

(5) Provide basic supports for development of various apps.

2 Development and Runtime Environment

Nibiru Studio runs in Nibiru VR system. Therefore, a headset mounted with Nibiru OS is needed for debugging. If you need to debug in a smart phone, please install the launcher (NibiruVRMobileLauncher.apk) in the development kit.

Nibiru Studio uses Android Studio as the development environment. For all the environment configurations, including gradle configurations, please refer to documents on Android Studio/gradle/Android. Here, we only introduce how to use Nibiru Studio kits when Android Studio + Android environment runs normal. Demo APK is developed and packaged based on Android Studio 3.0.1.

We mainly build and train models based on Python. For generation and training of specific models, Python is definitely not enough for large-scale data training. Hence, generally, enterprises use Python to build prototype first, then engineer with C++ or Java, finally encapsulate interfaces with Python.

Python development tools used here include:

(1) PyCharm (the Python IDE);

(2) Matplotlib (the 2D drawing library of and Python);

(3) Numpy (the Scientific Computing Library of Python).

Tensorflow is a complete deep-learning framework. This guide mainly helps you pre-process image data and train models using the Tensorflow framework.

If you have any questions on this guide, please refer to Nibiru Studio demo or ask questions in our Developer Forum (http://dev.bbs.inibiru.com/forum.php) or join in our official NibiruSDK QQ group (128275865).

3 Development Guide

3.1 Sample Collection

1. You need thousands of sample images for the training. In theory, we recommend you take pictures by yourself, but you can also use web crawlers to collect a great amount of required samples in a short time.
2. We aim to train models so you don't need to stage photos. As long as target object exists in the picture, it is enough.
3. Please make sure that different objects exist in different backgrounds. Do not take photos with similar surroundings. The commonly used VR environments, including shopping malls, and family life, and other outdoor environment, should be taken into account. Because the principle of image recognition is to recognize common features of objects. Once the scene is similar, the scene is saved as a common feature.
4. Please use models of the same type but different postures and state of use, to prevent over-fitting or failing to recognize objects in different situations.
5. Objects can be blocked in part or only take a small part of the scene to ensure the scene is as natural as possible and the object is close to its actual posture in use.

3.2 Pre-processing

Image is a form of information that people love to see. "Seeing is believing". Sometimes, one picture can excel thousands of words. Image processing achieves better effects through (linear or nonlinear) transformations on numerical images using computers. Photoshop and beauty cameras are applications of image processing technology. The most important application field of deep learning is Computer Vision (CV). In history, digital recognition of MNIST handwritten characters and large-scale image recognition of ImageNet both benefited from deep-learning models, whose accuracy was higher than traditional methods. Starting from the AlexNet model in 2012, subsequent models including VGG, GoogLeNet, and ResNet have refreshed the accuracy record in the ImageNet image recognition, even exceeded the human level. To obtain good recognition effects, pre-processing data sets is also very important compared with better models. The most commonly used methods include scaling, random slicing, random flip, and color transformation.

A. Adapt to neural network structure: data format / type that the network structure can receive is fixed. So, before training, you need to pre-process the samples to a format / type that can be read by the neural network.
B. Purify training samples: there may be "bad data" in the training samples. Through pre-processing, you can remove this part of data, or eliminate its effect.
C. Improve data: pre-processing training samples can increase the diversity of data. For example, rotation, mirroring, cutting and other methods can help enhance the spatial diversity of pictures, and make the models trained more robust.
D. Data normalization: pre-processing can transform data of different specifications into the same specifications. The most typical example is the normalization of image size.
E. Compressed data size: pre-processing can also reduce the data size. For example, the original data is FHD. If compressed to QQVGA, its size will be greatly reduced.

(1) Image pre-processing

Use TensorFlow to pre-process image data and how to use tensorboard tool to visualize image data.

Read code of image data with TensorFlow：

```
reader = tf.WholeFileReader()
key, value = reader.read(tf.train.string_input_producer(['cat.jpg']))
image0 = tf.image.decode_jpeg(value)
```

Image scaling

Code:

```
resized_image = tf.image.resize_images(image0, [256, 256], \
method=tf.image.ResizeMethod.AREA)
```

Four choices for the method:

`ResizeMethod.BILINEAR`: bilinear interpolation

`ResizeMethod.NEAREST_NEIGHBOR`: nearest interpolation

`ResizeMethod.BICUBIC`: bi-cubic interpolation

`ResizeMethod.AREA`: area interpolation

Try these methods above to see scaling effects.

Image cutting

Code:

```
cropped_image = tf.image.crop_to_bounding_box(image0, 20, 20, 256, 256)
```

(2) File name pre-processing

Change file names into those which are easy to package and manage, and it will be easier to train and understand code. get_files can handle all pictures in the folder.

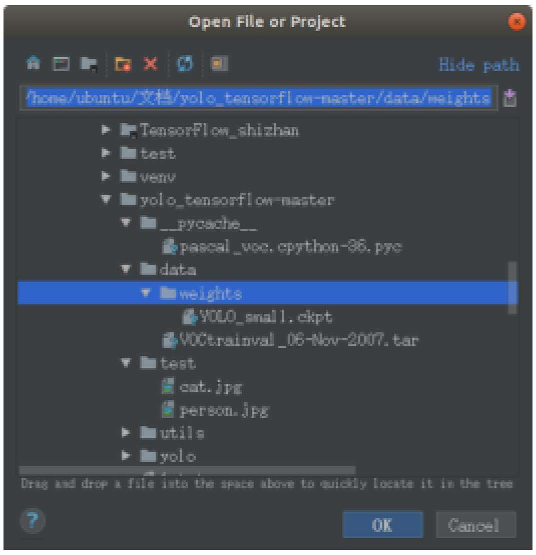
```
# Get all files in this directory and store them in the list
f = os.listdir(path)
n = 0
for i in f:
    # Set old name for file (path + file name)
    oldname = path + f[n]

    # Set new name for file
    newname = path + 'a' + str(n + 1) + '.JPG'

    # Use rename method in OS module to change file name
    os.rename(oldname, newname)
    print(oldname, '======>', newname)

    n += 1
```

3.3 Global Application Class - Config

1. File and path

```
DATA_PATH = 'data'

PASCAL_PATH = os.path.join(DATA_PATH, 'pascal_voc')

CACHE_PATH = os.path.join(PASCAL_PATH, 'cache')

OUTPUT_DIR = os.path.join(PASCAL_PATH, 'output')

WEIGHTS_DIR = os.path.join(PASCAL_PATH, 'weight')
```

2. Models for training are stored in the weight file of the package. Once trained, they are also stored in this file.



```
/home/ubuntu/文档/yolo_tensorflow-master/data/weights

    ▶ TensorFlow_shizhan
    ▶ test
    ▶ venv
    ▼ yolo_tensorflow-master
        ▼ __pycache__
            pascal_voc.cpython-36.pyc
        ▼ data
            ▼ weights
                YOLO_small.ckpt
            VOCtrainval_06-Nov-2007.tar
        ▼ test
            cat.jpg
            person.jpg
        ▶ utils
        ▶ yolo
Drag and drop a file into the space above to quickly locate it in the tree
```

```
WEIGHTS_FILE = None

# WEIGHTS_FILE = os.path.join(DATA_PATH, 'weights', 'YOLO_small.ckpt')
```

3. Model classification

```
CLASSES = ['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car', 'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse', 'motorbike', 'person', 'pottedplant', 'sheep', 'sofa', 'train', 'tvmonitor']
```

4. Model parameters

Image pixel size

```
IMAGE_SIZE = 448
```

Number of sliding window 7*7

```
CELL_SIZE = 7
```

Each sliding window is judged for 2 times

```
BOXES_PER_CELL = 2
```

Related settings for activation function

```
ALPHA = 0.1
DISP_CONSOLE = False


OBJECT_SCALE = 1.0
NOOBJECT_SCALE = 1.0
CLASS_SCALE = 2.0
COORD_SCALE = 5.0# solver parameter
```

Training parameters:

Learning rate

```
LEARNING_RATE = 0.0001
```

Total training times and attenuation rate

```
DECAY_STEPS = 30000
DECAY_RATE = 0.1
```

Scale of one training

```
    BATCH_SIZE = 45
```

Iteration

```
MAX_ITER = 15000
SUMMARY_ITER = 10
SAVE_ITER = 1000
```

Parameters for test model:

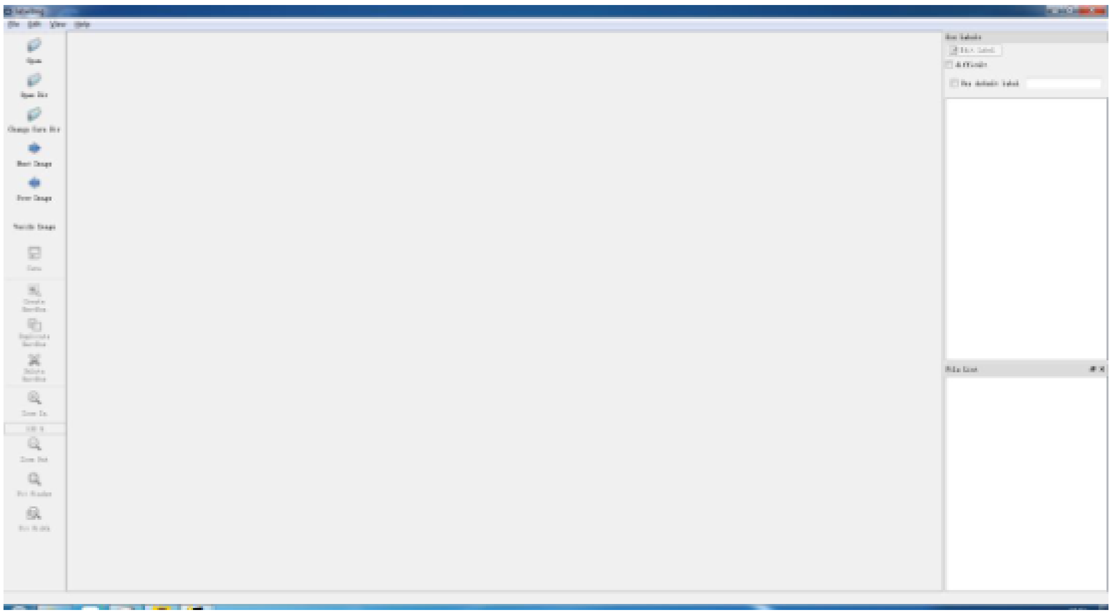
```
THRESHOLD = 0.2
IOU_THRESHOLD = 0.5
```

3.4 Image Annotation

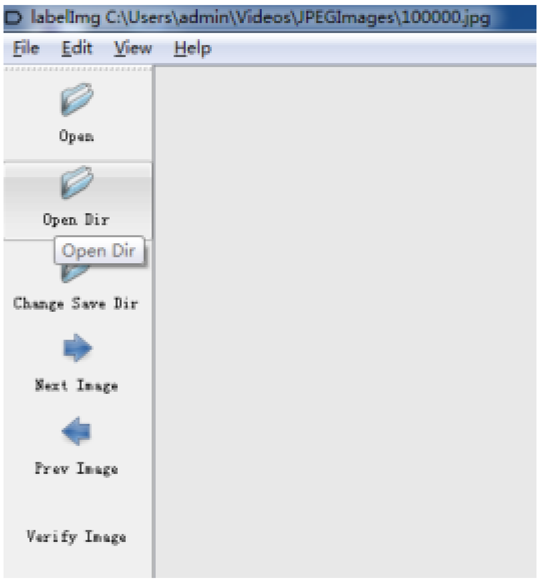
1. Software name: LabelImg

2. Steps:

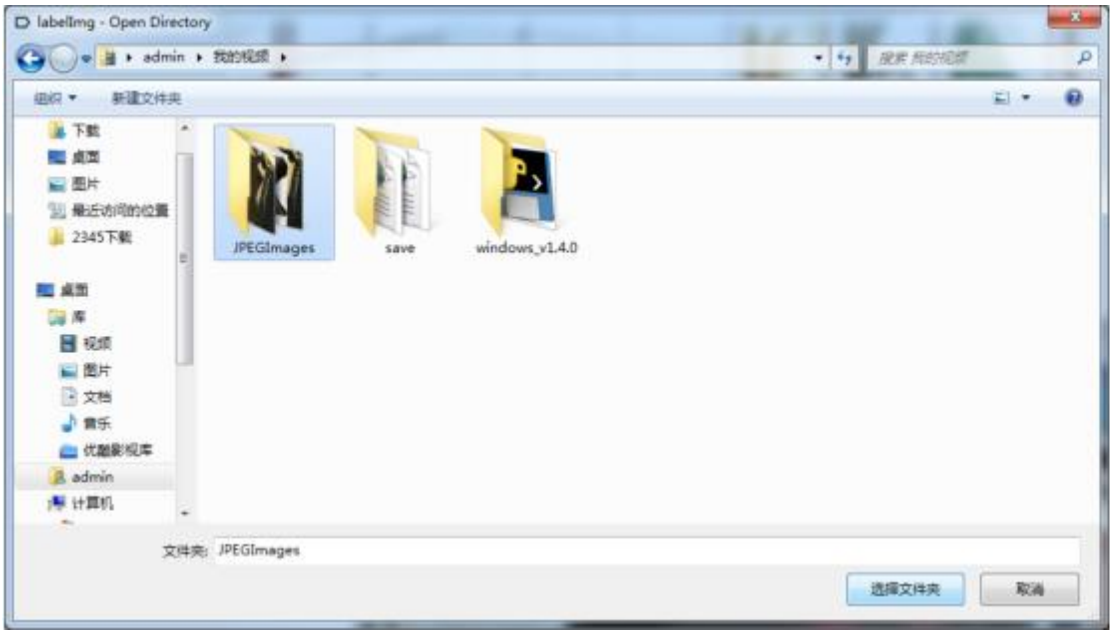
(1) Open the software



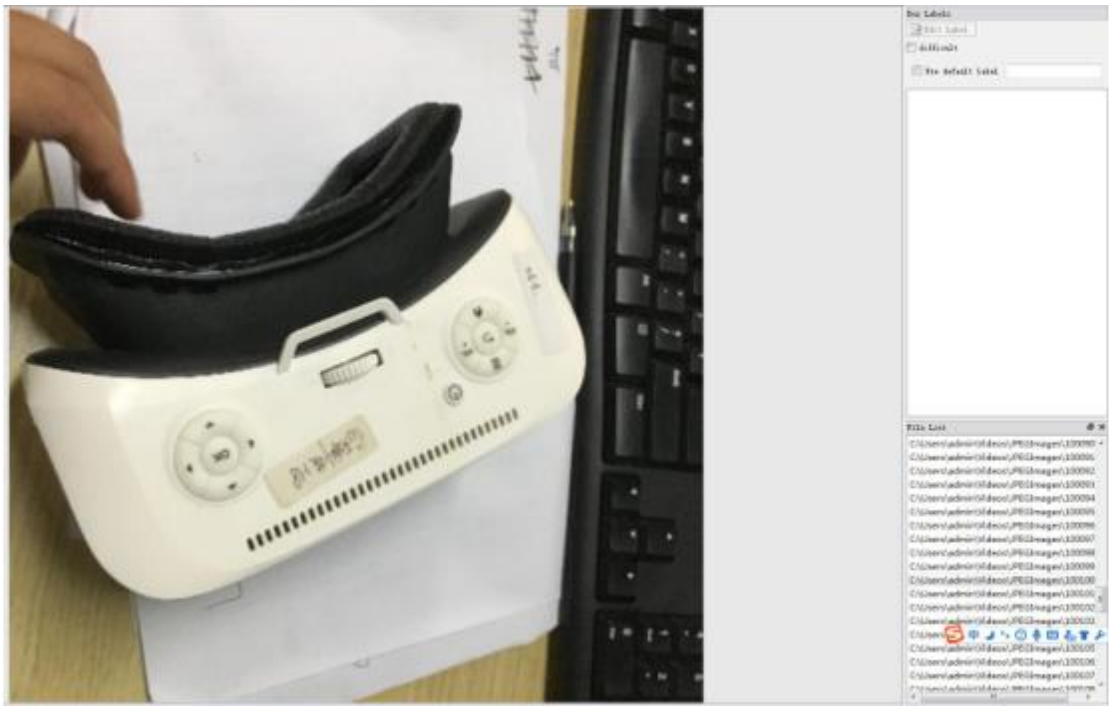
(2) The Open Dir on the left means opening all images in the file



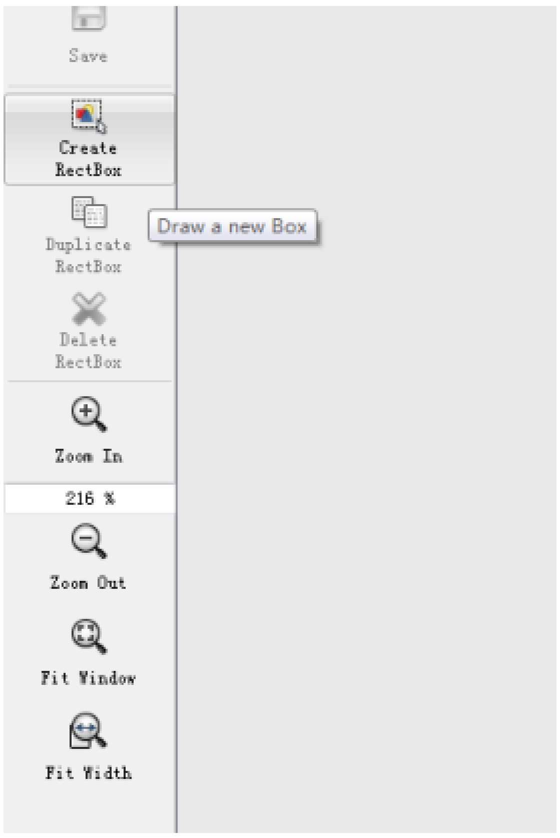
Choose the folder you need



The file is opened as shown



(3) Click Create RectBox to draw a new box



Try to draw a good box but don’t need to pursue perfection.

Draw multiple boxes if there are multiple objects that need recognizing in the picture.

E.g., draw three boxes for computer, mouse, and keyboard in one picture.

3.5 Runtime file - test file

Detector class

includes a series of actions to frame images, all defined from config.

```
self.net = net
self.weights_file = weight_file
self.classes = cfg.CLASSES
self.num_class = len(self.classes)
self.image_size = cfg.IMAGE_SIZE
self.cell_size = cfg.CELL_SIZE
self.boxes_per_cell = cfg.BOXES_PER_CELL
self.threshold = cfg.THRESHOLD
self.iou_threshold = cfg.IOU_THRESHOLD
# int 7x7x20
self.boundary1 = self.cell_size * self.cell_size * self.num_class
# int 7x7x20 + # int 7x7x2
self.boundary2 = self.boundary1 + self.cell_size * self.cell_size *
self.boxes_per_cell

self.sess = tf.Session()
self.sess.run(tf.global_variables_initializer())

print('Restoring weights from: ' + self.weights_file)
self.saver = tf.train.Saver()
self.saver.restore(self.sess, self.weights_file)
```

Statement called at last: `detector = Detector(yolo, weight_file) # 188`

1. draw_result function: draw and frame images
2. detect function: normalization, image input, and enforcement

(1) Image pre-processing

```
img_h, img_w, _ = img.shape
```

```
    inputs = cv2.resize(img, (self.image_size, self.image_size))
    inputs = cv2.cvtColor(inputs, cv2.COLOR_BGR2RGB).astype(np.float32)
```

# The default channel of opencv is BGR. Transform data types to increase accuracy of the following divisions.

```
    inputs = (inputs / 255.0) * 2.0 - 1.0 # normalize, color is at the space center!
    inputs = np.reshape(inputs, (1, self.image_size, self.image_size, 3)) # 4-D
Tensor
```

（2）result: put back the finished results framing

```
    result = self.detect_from_cvmat(inputs)[0]
```

（3）result: normalization

```
    for i in range(len(result)):
        result[i][1] *= (1.0 * img_w / self.image_size)
        result[i][2] *= (1.0 * img_h / self.image_size)
        result[i][3] *= (1.0 * img_w / self.image_size)
        result[i][4] *= (1.0 * img_h / self.image_size)
```

3. detect_from_cvmat: data format conversion

(It is an important matrix transformation function in OpenCV that can be used by calling CvMat* cvCreateMat (int rows, int cols, int type). Here, “type” can be any predefined type, and the structure of predefined typse is: CV_<bit_depth>(S|U|F)C<number_of_channels>.)

(1) Enforce “yolonet” to get result for forward propagation build_network, the result of a series framing

```
net_output = self.sess.run(self.net.logits,
        feed_dict={self.net.images: inputs})
results = []
for i in range(net_output.shape[0]): # .shape[0] number of zero dimension, similar to
len()
```

(2) Make a sequence of framing results

```
results.append(self.interpret_output(net_output[i]))
```

4. interpret_output function: process results of forward propagation and filter iou

（1）class_probs 3-D Tensor 7x7x20: the first half is the classification of class:

```
    class_probs = np.reshape(output[0:self.boundary1], (self.cell_size, self.cell_size,
self.num_class))
```

# scales 3-D Tensor 7x7x2: the second half is the two detection parts

```
    scales = np.reshape(output[self.boundary1:self.boundary2], (self.cell_size,
self.cell_size, self.boxes_per_cell))
```

(2) boxes 4-D Tensor 7x7x2x4: preserves framing info:

```
    boxes = np.reshape(output[self.boundary2:], (self.cell_size, self.cell_size,
self.boxes_per_cell, 4))
```

(3) Generate cell number [0，1，2，3，4，5，6] x 7 x 2. Put numbers in order and then consider their conversion.

```
    offset = np.transpose(np.reshape(np.array([np.arange(self.cell_size)] *
self.cell_size * self.boxes_per_cell), [self.boxes_per_cell, self.cell_size,
self.cell_size]), (1, 2, 0))
```

(4) boxes 4-D Tensor 7x7x2x4：information assignment of 4D framing

```
boxes[:, :, :, 0] += offset
boxes[:, :, :, 1] += np.transpose(offset, (1, 0, 2))
boxes[:, :, :, :2] = 1.0 * boxes[:, :, :, 0:2] / self.cell_size
boxes[:, :, :, 2:] = np.square(boxes[:, :, :, 2:])
boxes *= self.image_size
```

# Turn the normalization value of “boxes” into a meaningful one

# “probs” puts two frames of “cell” and their corresponding categories together

```
for i in range(self.boxes_per_cell):
    for j in range(self.num_class):
            probs[:, :, i, j] = np.multiply(
                class_probs[:, :, j], scales[:, :, i])
```

# “filter_mat_probs” determines “bool function” that is above the threshold, which is the same as the “probs”.

```
filter_mat_probs = np.array(probs >= self.threshold, dtype='bool')
```

# “boxes_filtered” takes out nonzero location and records corresponding “boxes function”

```
filter_mat_boxes = np.nonzero(filter_mat_probs)
boxes_filtered = boxes[filter_mat_boxes[0],
    filter_mat_boxes[1], filter_mat_boxes[2]]
```

# “probs_filtered” selects and turns corresponding part of “True” into an one-dimensional array which matches “boxes label”

```
  probs_filtered = probs[filter_mat_probs]
  classes_num_filtered = np.argmax(filter_mat_probs, axis=3)[filter_mat_boxes[0],
filter_mat_boxes[1], filter_mat_boxes[2]]
```

(5) According to score, all the boxes obtained are placed from high to low, and “argsort function” returns the index value of the array value from small to large.

```
  argsort = np.array(np.argsort(probs_filtered))[::-1]
  boxes_filtered = boxes_filtered[argsort]
  probs_filtered = probs_filtered[argsort]
  classes_num_filtered = classes_num_filtered[argsort]
```

(6) If iou is greater than the threshold value, that is, the same object. The value in the probs_filtered[j] array, the comparison result of the i picture is zero.

```
  for i in range(len(boxes_filtered)):
      if probs_filtered[i] == 0:
          continue
      for j in range(i + 1, len(boxes_filtered)):
          if self.iou(boxes_filtered[i], boxes_filtered[j]) > self.iou_threshold:
```

```
                        probs_filtered[j] = 0.0
```

(7) Remove and not consider the zero value in the "probs_filtered array"

```
    filter_iou = np.array(probs_filtered > 0.0, dtype='bool')
    boxes_filtered = boxes_filtered[filter_iou]
    probs_filtered = probs_filtered[filter_iou]
    classes_num_filtered = classes_num_filtered[filter_iou]
```

(8) Output the qualified value to "result" and use it as function's return value

```
    result = []
    for i in range(len(boxes_filtered)):
            result.append([self.classes[classes_num_filtered[i]], boxes_filtered[i][0],
boxes_filtered[i][1], boxes_filtered[i][2], boxes_filtered[i][3], probs_filtered[i]])
```

5. "iou function": input two frames' information for comparing whether they describe the same object

Functions: select the one with smaller difference. "tb" stands for length, and "lr" for width. "iou" in test is not used. "def iou(self, box1, box2)" has been defined in "yolonet":

```
# select the one with smaller difference. "tb" stands for length, "lr" for width.
        tb = min(box1[0] + 0.5 * box1[2], box2[0] + 0.5 * box2[2]) - \
                max(box1[0] - 0.5 * box1[2], box2[0] - 0.5 * box2[2])
        lr = min(box1[1] + 0.5 * box1[3], box2[1] + 0.5 * box2[3]) - \
                max(box1[1] - 0.5 * box1[3], box2[1] - 0.5 * box2[3])
        if tb < 0 or lr < 0:
                intersection = 0
        else:
                intersection = tb * lr
        return intersection / (box1[2] * box1[3] + box2[2] * box2[3] - intersection)
```

6. camera_detector and image_detector display data from camera and image respectively.

Just read one image during the test

```
imname = 'test/person.jpg'
detector.image_detector(imname)
```

3.6 Training Process - train

1. Parameter settings

(1) Class to import network and training data

```
    self.net = net
    self.data = data
```

(2) Import model parameters from config file

```
    self.weights_file = cfg.WEIGHTS_FILE
    self.max_iter = cfg.MAX_ITER # iteration number
    self.initial_learning_rate = cfg.LEARNING_RATE
    self.decay_steps = cfg.DECAY_STEPS
    self.decay_rate = cfg.DECAY_RATE
    self.staircase = cfg.STAIRCASE
    self.summary_iter = cfg.SUMMARY_ITER
    self.save_iter = cfg.SAVE_ITER
```

(3) Formatting and string initialization

```
    self.output_dir = os.path.join(cfg.OUTPUT_DIR,
  datetime.datetime.now().strftime('%Y_%m_%d_%H_%M'))
    if not os.path.exists(self.output_dir):
        os.makedirs(self.output_dir)
    self.save_cfg()

    self.variable_to_restore = tf.global_variables()
    self.restorer = tf.train.Saver(self.variable_to_restore, max_to_keep=None)
    self.saver = tf.train.Saver(self.variable_to_restore, max_to_keep=None)
    self.ckpt_file = os.path.join(self.output_dir, 'save.ckpt')
```

(4) “Summary” is an operation to monitor how Tensor takes value in the Internet. These

are peripheral operations in image and will nor affect data flow

```
self.summary_op = tf.summary.merge_all()
```

(5) “log” is the directory of event file. Here is the “log” under the project directory. The second parameter is an image that needs to be recorded by event file and also is the default image of tensorflow

```
self.writer = tf.summary.FileWriter(self.output_dir, flush_secs=60)

self.global_step = tf.get_variable(
    'global_step', [], initializer=tf.constant_initializer(0), trainable=False)
self.learning_rate = tf.train.exponential_decay(
    self.initial_learning_rate, self.global_step, self.decay_steps,
    self.decay_rate, self.staircase, name='learning_rate')
self.optimizer = tf.train.GradientDescentOptimizer(
    learning_rate=self.learning_rate).minimize(
    self.net.total_loss, global_step=self.global_step)
```

(6) tf.train.ExponentialMovingAverage is used to update parameters by moving average. The initialization of the function needs to provide decay ratio for controlling updating rate of model. The function maintains a shadow variable (parameter value after updating). The variable’s initial value is the initial value of the shadow variable. You can update the shadow variable as follows:

```
# shadow_variable = decay * shadow_variable + (1-decay) * variable
self.ema = tf.train.ExponentialMovingAverage(decay=0.9999)
# tf.trainable_variables returns all unmarked “trainable=False” variable sets when
getting variables in the current computing graph
self.averages_op = self.ema.apply(tf.trainable_variables())
with tf.control_dependencies([self.optimizer]):
  self.train_op = tf.group(self.averages_op)
```

```
gpu_options = tf.GPUOptions()
config = tf.ConfigProto(gpu_options=gpu_options)
self.sess = tf.Session(config=config)
self.sess.run(tf.global_variables_initializer())
```

(7) initialize weights file to reduce training rate. Call “checkpoint” with “restore”

```
if self.weights_file is not None:
    print('Restoring weights from: ' + self.weights_file)
    self.restorer.restore(self.sess, self.weights_file)

self.writer.add_graph(self.sess.graph)
```

2. train function: The whole training process is defined here, including training method of running definition and other visualization settings

```
train_timer = Timer()
load_timer = Timer()
```

(1) Read data, count time

```
for step in xrange(1, self.max_iter + 1):

  # read file and import it to fed_dict
  load_timer.tic()
  images, labels = self.data.get()
  load_timer.toc()
  feed_dict = {self.net.images: images, self.net.labels: labels}
```

(2) operate summary_op every 10 times, namely, display on tensorboard once

Print once every 100 times

```
if step % self.summary_iter == 0:
    if step % (self.summary_iter * 10) == 0:
    train_timer.tic()
```

```
            summary_str, loss, _ = self.sess.run(
                [self.summary_op, self.net.total_loss, self.train_op],
                feed_dict=feed_dict)
                train_timer.toc()

            # data structure to print
            log_str = ('{} Epoch: {}, Step: {}, Learning rate: {},'
                ' Loss: {:5.3f}\nSpeed: {:.3f}s/iter,'
                ' Load: {:.3f}s/iter, Remain: {}').format(

      datetime.datetime.now().strftime('%m/%d %H:%M:%S'),
                self.data.epoch,
                int(step),
                round(self.learning_rate.eval(session=self.sess), 6),
                loss,
                train_timer.average_time,
                load_timer.average_time,
                train_timer.remain(step, self.max_iter))
            print(log_str)

        else:
            train_timer.tic()
            summary_str, _ = self.sess.run(
                [self.summary_op, self.train_op],
                feed_dict=feed_dict)
            train_timer.toc()

        self.writer.add_summary(summary_str, step)

    else:
```

```
    train_timer.tic()
    self.sess.run(self.train_op, feed_dict=feed_dict)
    train_timer.toc()
```

(3) Save ckpt file after 1000 times, call“saver” for save

```
            if step % self.save_iter == 0:
                print('{} Saving checkpoint file to: {}'.format(
                    datetime.datetime.now().strftime('%m/%d %H:%M:%S'),
                    self.output_dir))
                self.saver.save(self.sess, self.ckpt_file,
                        global_step=self.global_step)
```

3. “save_cfg” defines the method to save temporary data files so as to prevent model lose during training

(1) Introduce “with” statement to call “close()”

```
        with open(os.path.join(self.output_dir, 'config.txt'), 'w') as f:
            cfg_dict = cfg.__dict__
```

(2) for key in sorted(cfg_dict.keys()): # not to change former sequence

```
        # check whether letters in the string are capitalized
        if key[0].isupper():
            cfg_str = '{}: {}\n'.format(key, cfg_dict[key])
            f.write(cfg_str)
```

4. There is no need to change the right function

```
            update_config_paths
        cfg.DATA_PATH = data_dir
        cfg.PASCAL_PATH = os.path.join(data_dir, 'pascal_voc')
        cfg.CACHE_PATH = os.path.join(cfg.PASCAL_PATH, 'cache')
        cfg.OUTPUT_DIR = os.path.join(cfg.PASCAL_PATH, 'output')
```

```
cfg.WEIGHTS_DIR = os.path.join(cfg.PASCAL_PATH, 'weights')
cfg.WEIGHTS_FILE = os.path.join(cfg.WEIGHTS_DIR, weights_file)
```

3.7 Test in Android Terminal

The source code of Android Demo doesn’t include the trained tensorflow model, but it needs model in compiling.

There are three models for object recognition, pedestrian detection, and image style transfer.

When compiling Bazel, you don’t need to download the model by yourself because automatic downloading operation is already set in “//tensorflow/examples/android/BUILD”.

But if you compile with Android Studio or other tools, the automatic downloading operation in “BUILD” will not be enabled.

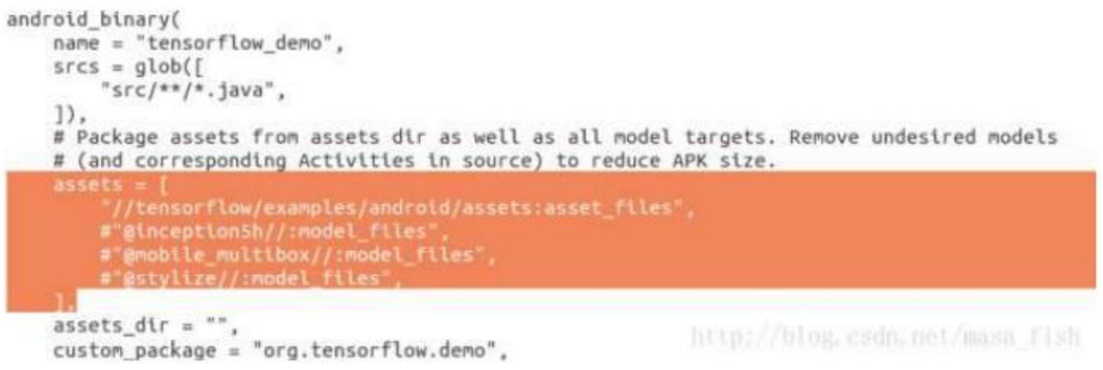
(1) Download zip files of the model manually

```
$ curl -L
https://storage.googleapis.com/download.tensorflow.org/models/inception5h.zip -o
/tmp/inception5h.zip
$ curl -L
https://storage.googleapis.com/download.tensorflow.org/models/mobile_multibox_v1.
zip -o /tmp/mobile_multibox_v1.zip
$ curl -L
https://storage.googleapis.com/download.tensorflow.org/models/mobile_multibox_v1.
zip -o /tmp/stylize_v1.zip
```

Unzip the three files above to folder:

“//tensorflow/examples/android/assets”

Open “//tensoeflow/examples/android/BUILD” to find the code that starts with “android_binary”, use “#” to annotate three codes begin with “@” in “assets = [ ]”


```
android_binary(
    name = "tensorflow_demo",
    srcs = glob([
        "src/**/*.java",
    ]),
    # Package assets from assets dir as well as all model targets. Remove undesired models
    # (and corresponding Activities in source) to reduce APK size.
    assets = [
        "//tensorflow/examples/android/assets:asset_files",
        #"@inception5h//:model_files",
        #"@mobile_multibox//:model_files",
        #"@stylize//:model_files",
    ],
    assets_dir = "",
    custom_package = "org.tensorflow.demo",
```

If not annotated, the model will be downloaded again when compiling with Bazel, even if you put the model in the assets file.

(2) Connect Android cell phone

1. Install “adb$ sudo apt-get install android-tools-adb”

2. Set the phone in “debug model”

Open the phone and find “usb debugging”. Here’s the sample path:

“setting-general-developer options-enable USB debugging”

(Each phone may not be exactly the same)

Connect to PC through USB and it will prompt as follows:

“allow USB debugging?”

Tick “always allow from this computer” and then click “ok”. (It will prompt only once)

(3) Check whether the phone is connected on your PC:

```
$adb devices
```

It will output the connected devices；

```
List of devices
xxxx, device
```

The problem you may meet: List of devices input display: xxx, offline.

Situation like that will lead the compiling to fail. You can resolve this problem as follows:

switch “cd” to “build_tools” folder in the installation path of “sdk”, and run

```
$ adb kill-server$ sudo adb start-server
```

(Some netizens said we should start “adb” with root permission) then run `$ adb devices`.

(4) Compile and install apk

1. Compile and generate apk:

```
$ cd ~/tensorflow$ bazel build //tensorflow/examples/android:tensorflow_demo
```

The following three files will be generated:

```
bazel-bin/tensorflow/examples/android/tensorflow_demo_deploy.jar
bazel-bin/tensorflow/examples/android/tensorflow_demo_unsigned.apk
bazel-bin/tensorflow/examples/android/tensorflow_demo.apk
```

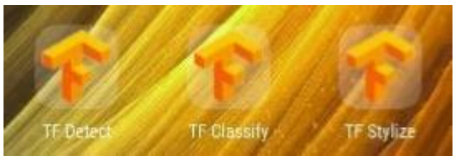
Then find those files in the “bazel-bin/tensorflow/examples/android/”

2. Install apk in Android cell phone

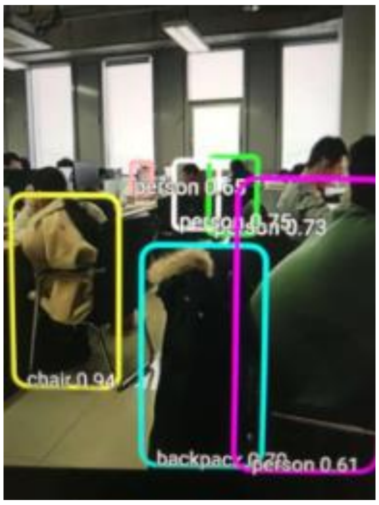
```
$ adb install -r -g bazel-bin/tensorflow/examples/android/tensorflow_demo.apk
```

During the installation, the phone screen will prompt you “whether to install the software, “whether to access to cameras”, and the like.

After installation, you can see three yellow icons on the main screen. Their names are TF Classify, TF Detect, and TF.



Final effect:



Effect completed.