天津大学

《专业课程设计2》结课报告

基于深度学习的图书馆一层区域识别

学　　院　智能与计算学部

专　　业　软件工程

年　　级　2017级

姓　　名　曾洋等五人

学　　号　3017218147

2019 年 9 月 9 日

目　　录

第一章　项目概述

1.1　总述

本项目开发周期为三周左右，旨在通过基于SSD算法的深度学习训练物体检测模型，达成识别郑东图书馆一层各区域的目的。

同时，搭建服务器和安卓客户端以实现照片的上传与接收，并将推理结果返回客户端以实现结果可视化。

1.2　成员分工

大致成员分工如下表所示：

表 1-1　成员分工

姓名	学号	分工
曾洋	3017218147	模型训练
全康连	3017218135	服务器端设计
侯湘琳	3017218154	服务器端设计
夏熙	3017218170	客户端设计
姚金妮	3017218176	客户端设计

1.3　项目过程

本项目为期三周，大致过程如下表所示：

表 1-2　项目过程

时间	项目进程
2019.08.19-2019.08.21	确定项目主题和小组分工
2019.08.22-2019.08.23	拍摄照片、搭建训练环境
2019.08.24-2019.08.27	初步完成服务器搭建、实现客户端
2019.08.28-2019.08.30	准备数据集、训练模型
2019.08.31-2019.09.03	完成模型训练、测试服务器端与客户端建立联系
2019.09.04-2019.09.06	项目完善、完成实验报告

第二章　　需求分析

2.1　项目目标

天津大学郑东图书馆场馆大，馆内相似度高，学生很难通过路标找到相应的位置，因此希望开发一款app来实现天津大学郑东图书馆的室内定位导航技术。经过查阅资料，我们考虑采用图像识别技术来实现图书馆室内定位。用户可以通过在app上拍照，来确定自己当前所在图书馆的位置。

2.2　项目特点

运用图像识别领域准确率相对较高的ssd算法进行模型训练，并将其部署到服务器，供客户端上传照片推理使用。

2.3　项目周期

软件的开发期限约为三周。

2.4　功能说明

用户在android端拍摄图片并将它上传到服务器，服务器接收图片并推理，将图片所代表的图书馆最可能的位置通过字符串传给android端，android通过处理将图书馆位置展示给客户。

2.5　运行环境

1. 客户端平台：Android 8.0以上
2. 服务器平台：tomcat
3. 模型训练平台：Linux + Jupyter Notebook

第三章　项目设计

3.1　客户端设计

3.1.1　实现功能

调用相机拍照和从相册选择照片功能，实现裁剪图片功能，并将裁剪后的图片保存到指定路径；将图片上传至服务器，从服务器端获取字符串并将根据返回的字符串内容将对应文字和图片显示在界面上。

3.1.2　Java文件结构

文件结构见图3-1所示。其中，constant文件存储服务器基地址，SelectPicturePopupWindow用于实现选择框窗口，PictureSelectFragment用于实现拍照和从相册选择照片功能，CropActivity实现裁剪照片功能，ResultActicity实现接收返回字符串并修改UI界面功能，BaseFragment实现询问是否授予权限功能。

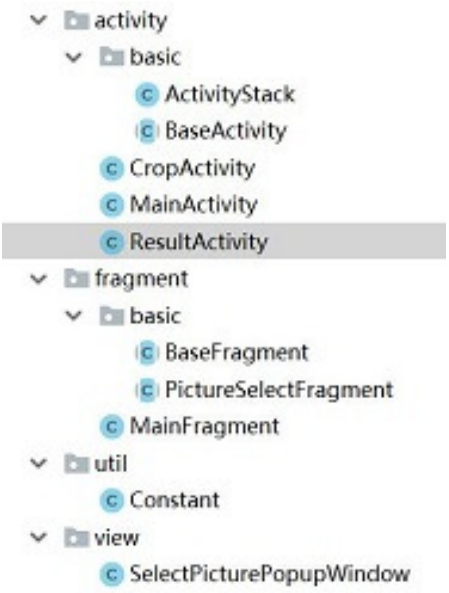

图 3-1　Java文件结构

3.1.3　初始页面介绍

3.1.3.1　点击选择图片后跳出选择方式窗口

选择框效果如图3-2所示。

设置图片点击监听：

图 3-2　　选择框效果

```
mPictureIv.setOnClickListener(new View.OnClickListener()
    {
        @Override
        public void onClick(View v) {
            selectPicture();
        }
    });
```

为选择框设置跳出动画，符合平时的使用习惯：

```
public void showPopupWindow(Activity activity) {
    popupWindow = new PopupWindow(mMenuView,      // 添加
        到popupWindow
            ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.WRAP_CONTENT);
    popupWindow.setBackgroundDrawable(new ColorDrawable(
        Color.TRANSPARENT));
    popupWindow.showAtLocation(activity.getWindow().
        getDecorView(), Gravity.CENTER | Gravity.BOTTOM, 0,
         0);
    popupWindow.setAnimationStyle(android.R.style.
        Animation_InputMethod);     // 设置窗口显示的动画效
        果
    popupWindow.setFocusable(true); // 点击其他地方隐藏键
        盘 popupWindow
    popupWindow.setOutsideTouchable(true);
```

```
9       popupWindow.setTouchInterceptor(new View.
            OnTouchListener() {
10          public boolean onTouch(View v, MotionEvent event)
                {
11              if (event.getAction() == MotionEvent.
                    ACTION_OUTSIDE) {
12                  popupWindow.dismiss();
13                  return true;
14              }
15              return false;
16          }
17      });
18      popupWindow.update();
19  }
```

3.1.3.2 实现选择框中的点击事件

实现点击事件，点击不同的按钮则向onSelectedListener监听接口传入不同参数：

```
1   public void onClick(View v) {
2       switch (v.getId()) {
3           case R.id.picture_selector_take_photo_btn:
4               if(null != mOnSelectedListener) {
5                   mOnSelectedListener.OnSelected(v, 0);
6               }
7               break;
8           case R.id.picture_selector_pick_picture_btn:
9               if(null != mOnSelectedListener) {
10                  mOnSelectedListener.OnSelected(v, 1);
11              }
12              break;
13              case R.id.picture_selector_cancel_btn:
14              if(null != mOnSelectedListener) {
15                  mOnSelectedListener.OnSelected(v, 2);
16              }
17              break;
18          }
```

```
19      }
```

编写选择监听接口，该接口调用了OnSelected函数，该函数定义在PictureSelectFragment类中（类PictureSelectFragment实现了SelectPicturePopupWindow.OnSelectedListener接口），从而实现通过点击不同的按钮实现不同的方法：

```
1  public interface OnSelectedListener {//选择监听接口
2      void OnSelected(View v, int position);
3  }
```

通过对选择框SelectPicturePopupWindow的封装，使用起来就比较简单了，只需要初始化及设置选择的监听就可以：

```
1  mSelectPicturePopupWindow = new SelectPicturePopupWindow(
       mContext);
2  mSelectPicturePopupWindow.setOnSelectedListener(this);
```

3.1.3.3 实现拍照和选择照片功能

在类PictureSelectFragment中实现拍照功能：

```
1  private void takePhoto() {
2      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.
           JELLY_BEAN // Permission was added in API Level 16
3                  && ActivityCompat.checkSelfPermission(
                       mActivity, Manifest.permission.
                       WRITE_EXTERNAL_STORAGE)
4                  != PackageManager.PERMISSION_GRANTED) {
5              requestPermission(Manifest.permission.
                   WRITE_EXTERNAL_STORAGE,
6                          getString(R.string.
                               permission_write_storage_rationale),
                           EQUEST_STORAGE_WRITE_ACCESS_PERMISSION)
                           ;
7      } else {
8          mSelectPicturePopupWindow.dismissPopupWindow();
9          Intent takeIntent = new Intent(MediaStore.
               ACTION_IMAGE_CAPTURE);
10         //下面这句指定调用相机拍照后的照片存储的路径
```

```
            takeIntent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.
                fromFile(new File(mTempPhotoPath)));
            startActivityForResult(takeIntent,
                CAMERA_REQUEST_CODE);
        }
    }
```

从相册选取照片的功能：

```
private void pickFromGallery() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.
        JELLY_BEAN // Permission was added in API Level 16
            && ActivityCompat.checkSelfPermission(
                mActivity, Manifest.permission.
                READ_EXTERNAL_STORAGE)
            != PackageManager.PERMISSION_GRANTED) {
        requestPermission(Manifest.permission.
            READ_EXTERNAL_STORAGE,
                getString(R.string.
                    permission_read_storage_rationale),
                REQUEST_STORAGE_READ_ACCESS_PERMISSION);
    } else {
        mSelectPicturePopupWindow.dismissPopupWindow();
        Intent pickIntent = new Intent(Intent.ACTION_PICK
            , null);
        pickIntent.setDataAndType(MediaStore.Images.Media
            .EXTERNAL_CONTENT_URI, "image/*");
        startActivityForResult(pickIntent,
            GALLERY_REQUEST_CODE);
    }
}
```

因为上面两个功能涉及读写SD卡，所以需要在AndroidManifest中声明读写SD卡的权限：

```
<uses-permission android:name="android.permission.
    READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.
```

```
    WRITE_EXTERNAL_STORAGE" />
```

3.1.4 裁剪页面介绍

3.1.4.1 隐式Intent回调

拍照和从相册获取照片完成后会通过隐式Intent回调到onActivityResult()方法中，通过调用startCropActivity()方法对图片进行进一步处理，startCropActivity()通过CropActiivty对图片进行裁剪，裁剪完成时依旧会回调到onActivityResult，在这里处理选择的返回结果:

```
public void startCropActivity(Uri uri) {
    UCrop.of(uri, mDestinationUri)
            .withAspectRatio(1, 1)//比例为：11
            .withMaxResultSize(512, 512)//像素像素512*512
            .withTargetActivity(CropActivity.class)
            .start(mActivity, this);
}
```

3.1.4.2 实现裁剪图片

这里使用了一个适用于Android的图像裁剪库uCrop，该布局中主要的CropView就是uCrop框架提供的。在CropActivity中配置uCropView：

```
protected void initViews() {
    initToolBar();
    mGestureCropImageView = mUCropView.getCropImageView()
        ;
    mOverlayView = mUCropView.getOverlayView();
    mGestureCropImageView.setScaleEnabled(true);// 设置允许
        缩放
    mGestureCropImageView.setRotateEnabled(false);// 设置
        禁止旋转
    mOverlayView.setDimmedColor(Color.parseColor("#
        AA000000"));// 设置外部阴影颜
        色
    mOverlayView.setOvalDimmedLayer(false);// 设置周围阴影为
        矩形
    mOverlayView.setShowCropFrame(true); // 设置显示裁剪边框
    mOverlayView.setShowCropGrid(false);// 设置不显示裁剪网
        格
    final Intent intent = getIntent();
```

```
    setImageData(intent);
}
```

实现裁剪并保存图片：

```
try {
    final Bitmap croppedBitmap = mGestureCropImageView.
        cropImage();
    if (croppedBitmap != null) {
        outputStream = getContentResolver().
            openOutputStream(mOutputUri);
        croppedBitmap.compress(Bitmap.CompressFormat.JPEG
            , 85, outputStream);
        croppedBitmap.recycle();
        setResultUri(mOutputUri, mGestureCropImageView.
            getTargetAspectRatio());
        finish();
    } else {
        setResultException(new NullPointerException("
            CropImageView.cropImage() returned null."));
    }
}
```

在PictureSelectFragment中处理裁剪成功或不成功的返回值，若不成功，则通过Toast告知用户：

```
private void handleCropError(Intent result) {
    deleteTempPhotoFile();//删除临时文件
    final Throwable cropError = UCrop.getError(result);
    if (cropError != null) {
        Toast.makeText(mContext, cropError.getMessage(),
            Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(mContext, "无法剪切选择图
            片", Toast.LENGTH_SHORT).show();
    }
}
```

设置选择的回调接口，便于封装抽取：

```
public interface OnPictureSelectedListener {
    void onPictureSelected(Uri fileUri, Bitmap bitmap);
}
```

3.1.5 上传页面介绍

3.1.5.1 实现向服务器上传图片

完善MainFragment中裁剪图片结果的监听回调，实现裁剪完成后上传图片到服务器：

```
// 设置裁剪图片结果监听
setOnPictureSelectedListener(new
    OnPictureSelectedListener() {
    @Override
    public void onPictureSelected(Uri fileUri, Bitmap
        bitmap) {
        String filePath = fileUri.getEncodedPath();
        final String imagePath = Uri.decode(filePath);
        uploadImage(imagePath);
    }
});
```

使用OkHttp完成图片上传，OkHttp提供了MultipartBody来进行文件的上传，由于OkHttp虽然有异步网络访问，但是回调还是处在子线程不能修改界面，所以编写一个NetworkTask来进行子线程到主线程的链接：

```
OkHttpClient mOkHttpClient = new OkHttpClient();
String result = "error";
MultipartBody.Builder builder = new MultipartBody.Builder
    ();
builder.addFormDataPart("image", imagePath,
        RequestBody.create(MediaType.parse("image/jpeg"),
            new File(imagePath)));
RequestBody requestBody = builder.build();
Request.Builder reqBuilder = new Request.Builder();
Request request = reqBuilder
        .url(Constant.BASE_URL + "/uploadimage")
        .post(requestBody)
```

```
        .build();
```

3.1.5.2 更改上传方法的过程

向服务器上传文件的步骤是碰壁次数最多的。最开始没有考虑线程，导致APP向服务器上传时经常卡死，显示APP无响应。在查找资料后也意识到了不可以使用GET方法，GET方法只能用于上传较小的数据。再改成利用多线程和http的post请求，到后来考虑用asynctask使用异步线程以及http连接，再到最后查了许多资料后，安卓端和服务器都进行了多次改动，最终能够使用okhttp最终能够实现用户端向服务器稳定地上传文件。

3.1.6 结果页面介绍

3.1.6.1 接收服务器返回的字符串

因为接收的是字符串，体积较小，所以使用HTTP的GET方法即可。因连接网络是耗时操作，故使用子线程。并且将收到的字符串写入到response中：

```
URL url = new URL("http://172.20.10.7:8080/
    ImageUploadServer/");
connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod("GET");
connection.setConnectTimeout(8000);
connection.setReadTimeout(8000);
InputStream in = connection.getInputStream();
reader = new BufferedReader(new InputStreamReader(in));
 StringBuilder response = new StringBuilder();
String line;
 while ((line = reader.readLine()) != null) {
    response.append(line);
}
showResponse(response.toString(), resultImage);
```

3.1.6.2 标识位置

再判断字符串的内容，即通过模型对图片进行推理得出的最大概率的标签，来判断处于图书馆的哪一区域，更改 ResultActivity的UI界面，再进行展示：

```
if (firstW.contains("Borrow")) {
resultText.setText("自助借还区");
firstImg.setImageDrawable(ContextCompat.getDrawable(
    ResultActivity.this, R.drawable.borrow));
```

3.2 服务器设计

项目初构想如下:

Android上传图片至服务器，发送图像识别http请求，服务器java调用tensorlfow，推理结果回送安卓。根据该设想，服务器搭建过程主要分为原理讲解，实际操作指导，代码编写三部分

3.2.1 原理讲解

- 我们使用javaweb+tomcat来实现我们的服务器。其中，javaweb是运行在tomcat服务器上的处理请求的应答者程序；tomcat则是javaweb程序的服务器容器
- Android客户端发送http请求和接收到应答的流程
  - 客户端调用指定函数向指定ip地址的服务器发出http请求
  - 服务器容器tomcat收到http请求，将请求交给javaweb处理
  - javaweb应用程序使用servlet处理http请求
  - servlet根据请求执行相应操作，发送应答到Android客户端

3.2.2 实际操作指导

1. Tomcat服务器安装

   Tomcat是由Apache组织提供的一种Web服务器，提供对jsp和Servlet的支持，它是一种轻量级的javaweb容器，完全免费，主要用于中小型web项目。在安装tomcat服务器的前提下，本机pc需要配置jdk的环境，在此不详细讲解，具体教程可参考链接https://www.runoob.com/java/java-environment-setup.html。以下开始Tomcat服务器的安装指导

   (a) 到http://tomcat.apache.org下载tomcat

   tomcat首页，点击左侧download处tomcat9.0，如图3-3

   tomcat下载，选择32或64的window压缩包，如图3-4

   下载后的包解压并安装（不要装在c盘），如图3-5,3-6为下载压缩包和解压后文件

   tomcat的安装目录介绍

   bin:可执行文件

   conf:tomcat服务器的配置文件

   lib:tomcat启动后需要依赖的jar包

   logs:tomcat工作后的日志文件

   webapps:tomcat部署javaweb工程的目录

   work:jsp文件被翻译之后和session对象被序列化之后保存的位置

图 3-3　　tomcat官网首页

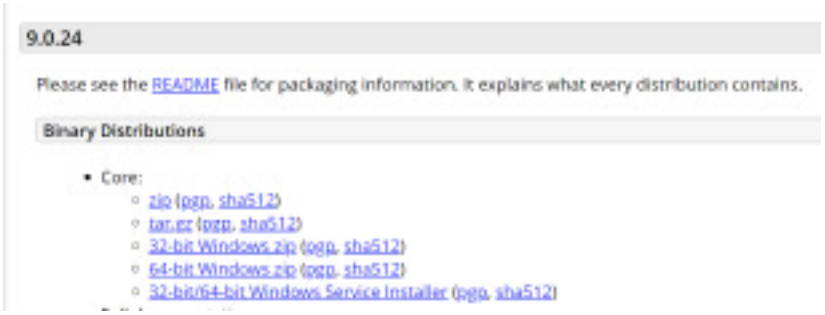

图 3-4　　tomcat安装选项

图 3-5　　下载压缩包

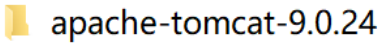

图 3-6　　解压后文件夹

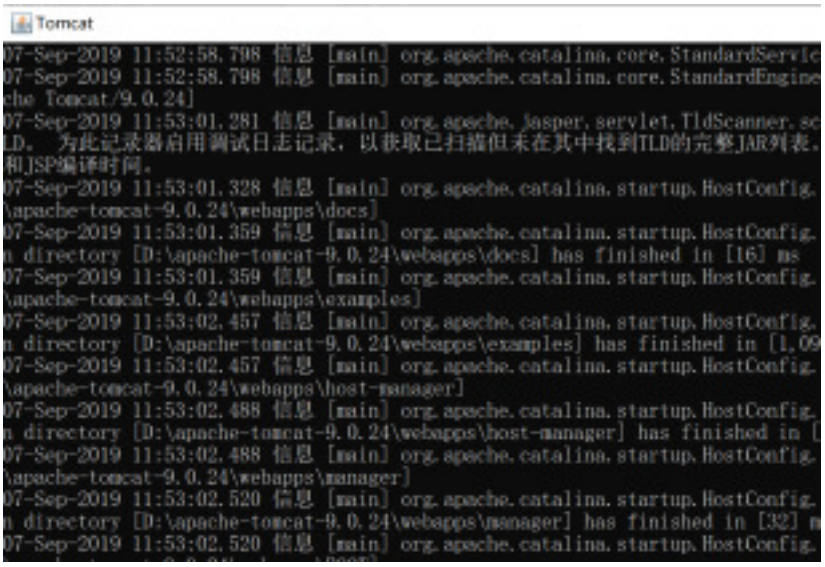

图 3-7　　tomcat运行弹窗

(b) tomcat环境变量配置及启动验证

jdk环境变量配置（参考给出的jdk链接教程）

打开控制台（cmd命令打开窗口，以管理员身份运行），输入命令startup.bat,会弹出一个黑窗口，不要关闭此窗口，关闭后即把tomcat停止了。如图3-7所示

打开浏览器，输入localhost:8080，若出现如下页面，则说明tomcat启动成功.如图3-8

2. JavaWeb–Servlet项目建立

Servlet使用java编写的服务器端程序，其主要功能在于交互式地浏览和修改数据，生成动态web内容。Servlet的工作模式如下所述：

- 客户端发送请求至服务器
- 服务器启动并调用servlet，servlet根据客户端请求生成相应内容并将其传给服务器
- 服务器将响应返回客户端

(a) 建立javaweb项目

本机使用eclipse完成web项目的编写。在eclipse中依次点击files-¿new-¿project，在该界面中点击web中的Dynamic Web Project。如图3-9所示

接着在该页面中target runtime中选择安装的正确tomcat的正确版本。如图**??**所示

在该页面中不要忘记勾选自动生成项目的配置文件web.xml。如

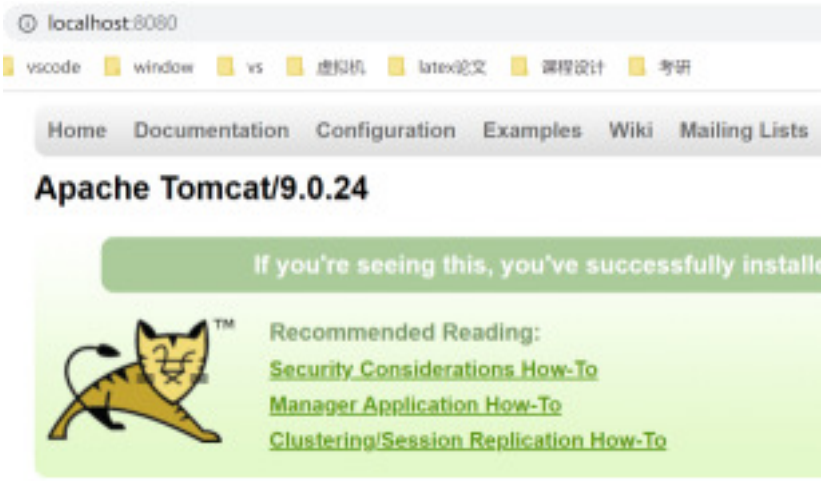

图 3-8　　tomcat首页

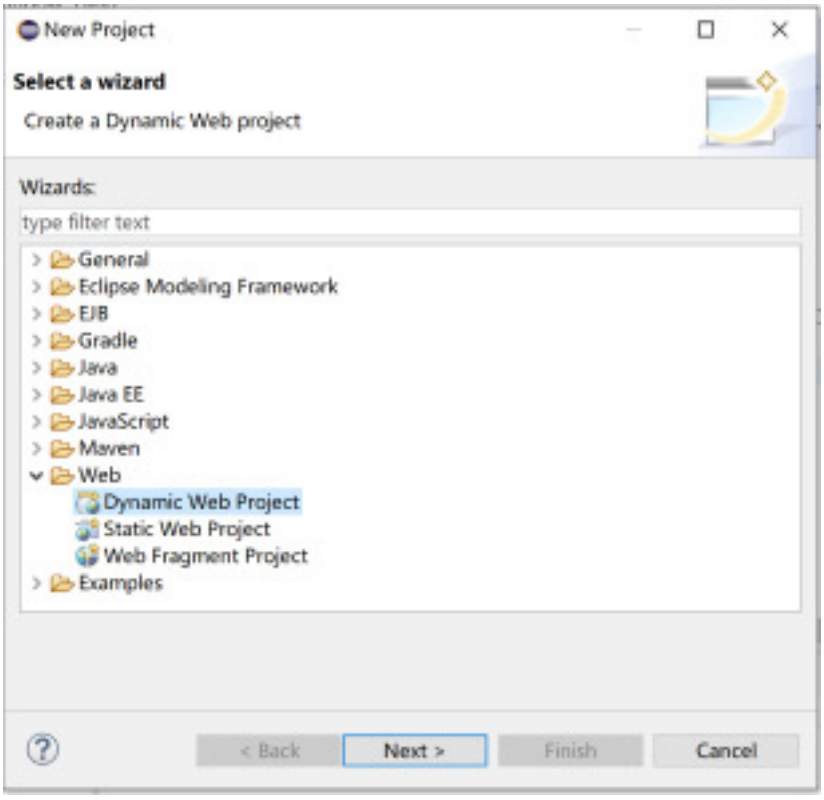

图 3-9　　Dynamic Web Project

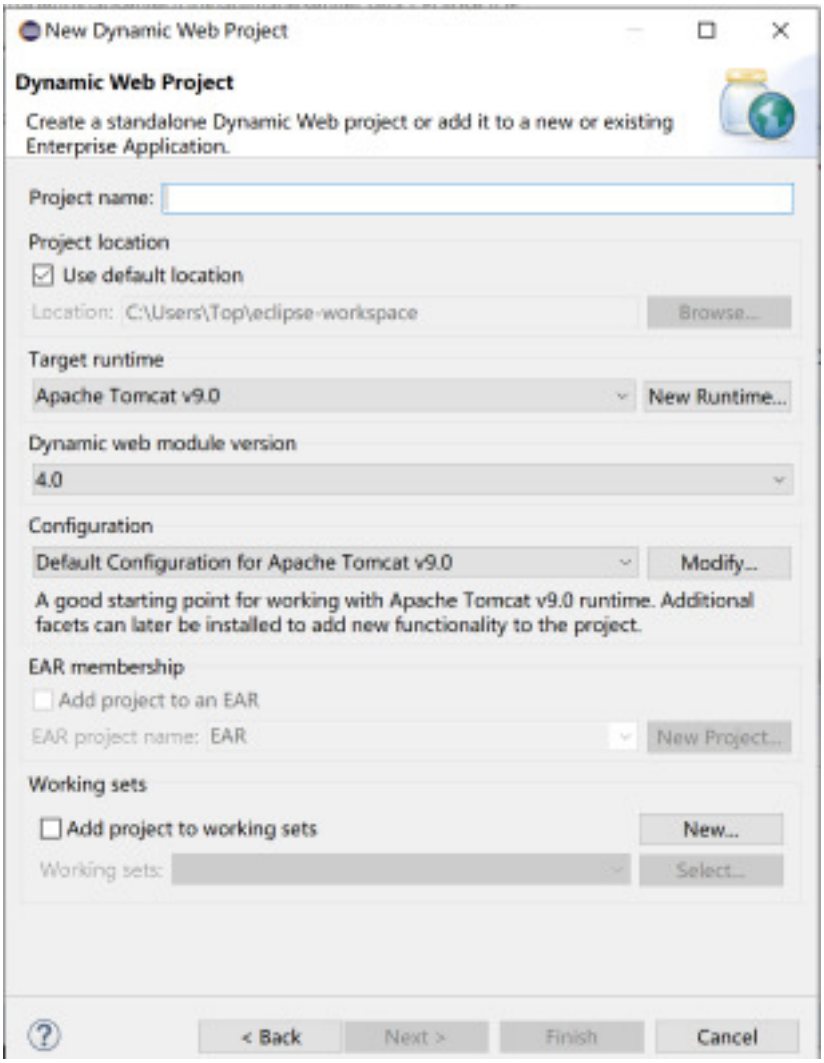

图 3-10　　选择tomcat

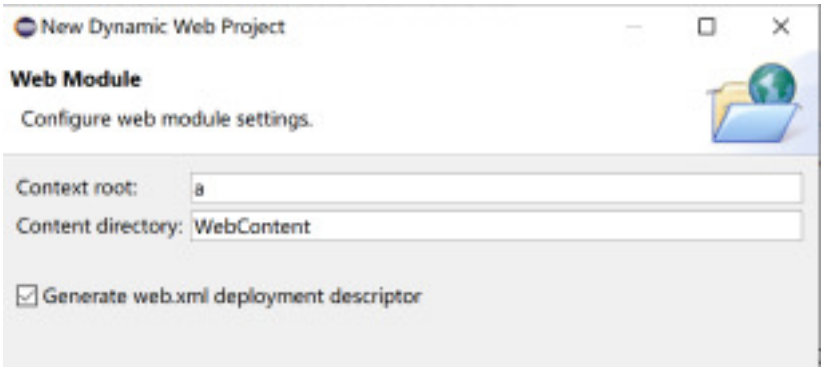

图 3-11　　生成web.xml配置文件

图3-11所示

(b) 新建servlet

在项目src下新建放置servlet的package，然后右键点击package-new-servlet。在该页面中可以选择servlet中的方法，根据自己的服务器功能选择相应方法。此次服务器选择的是post和get。如图3-12所示

(c) 导入所需资源包及配置tensorflow环境

在将web项目部署到tomcat时，需要将项目导出成war文件，只有webcontent文件夹和src导出，所以我们需要将额外导入的jar包放入到webcontent文件夹下的lib文件夹中，项目目录如下。将jar添加后，对project进行build path。注意tensorflow_jni.dll文件放在web项目下即可。如图3-13所示

配置window下的tensorflow环境请参考教程

https://www.jb51.net/article/137458.htm

配置jdk的tensorflow环境请参考教程

https://tensorflow.google.cn/install/lang_java

(d) 放置tensorlfow模型

在webroot/web-inf下将tensorflow训练好的模型文件models放置

(e) 在web.xml中配置新建的servlet

在web.xml中添加图下配置。如图3-14所示

3. 部署web项目到tomcat

在将web项目的代码编写完成后，需要将项目部署到tomcat上，才能成功运行web项目

(a) 导出项目war文件

右键点击项目-export-war file,在该页面中选择导出路径，该路径为tomcat/webapps目录下。如图3-15所示

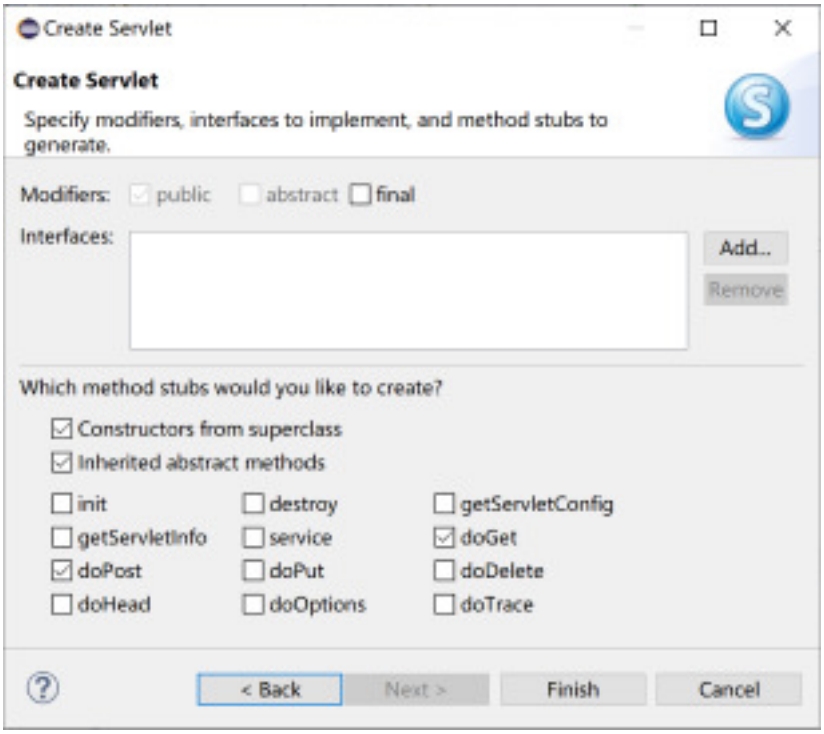

图 3-12　　选择servlet方法

(b) 建立server

eclipse中点击window-show view-other-server,出现如下界面。正确选择tomcat的安装版本，为你的server起名，点击next。如图3-16所示在该界面中，先不要添加web项目，先进行server的配置，点击finish。如图3-20所示

建立server后servers窗口出现server实例，双击该server，进行server的配置。在server location中选择第二个选项，该选项为tomcat的安装目录，便于之后对模型的调取和安卓端图片的保存。配置好后点击保存。如图3-18,3-19所示

之后右键点击server-¿add and remove,将web项目添加到server上。之后双击server，便可运行web项目。如图3-20

3.2.3　代码编写

服务器的设计先后有两个版本，分别为web项目classify_web_server和ImageUploadServer，根据安卓端的更改服务器端也相应发生了变化，下面分别阐述两版服务器编写的思路

1. classify_web_server

在classify_web_server的src中，有servlet-MyServlet和一个处理java调用tensorflow模型的类LabelImage

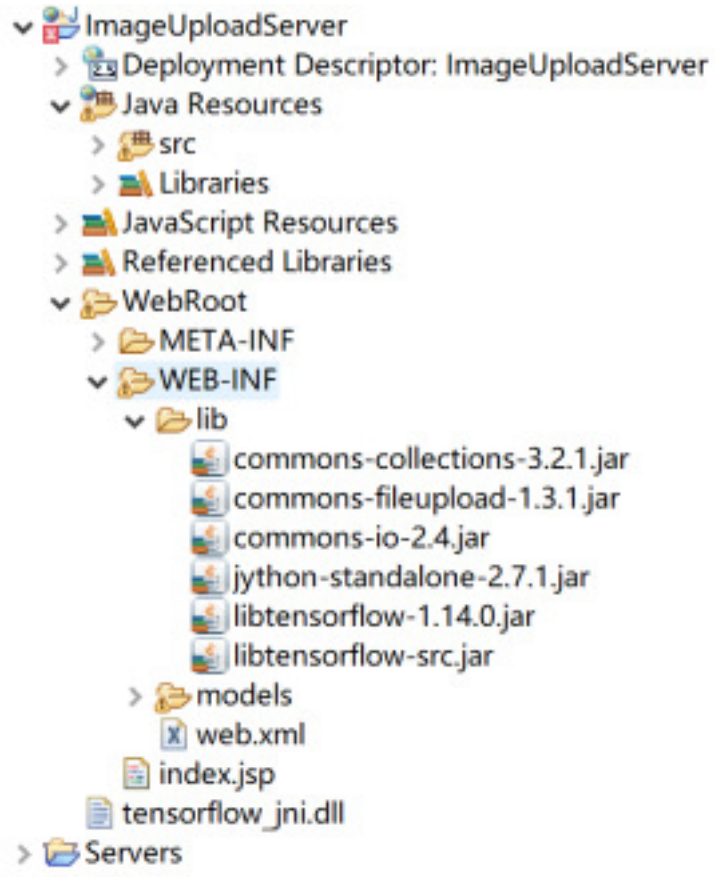

图 3-13　　web工程目录

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <display-name></display-name>

  <servlet>
    <servlet-name>UploadImageServlet</servlet-name>
    <servlet-class>com.kevin.imageuploadserver.UploadImageServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>UploadImageServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
```

图 3-14　　配置web.xml文件

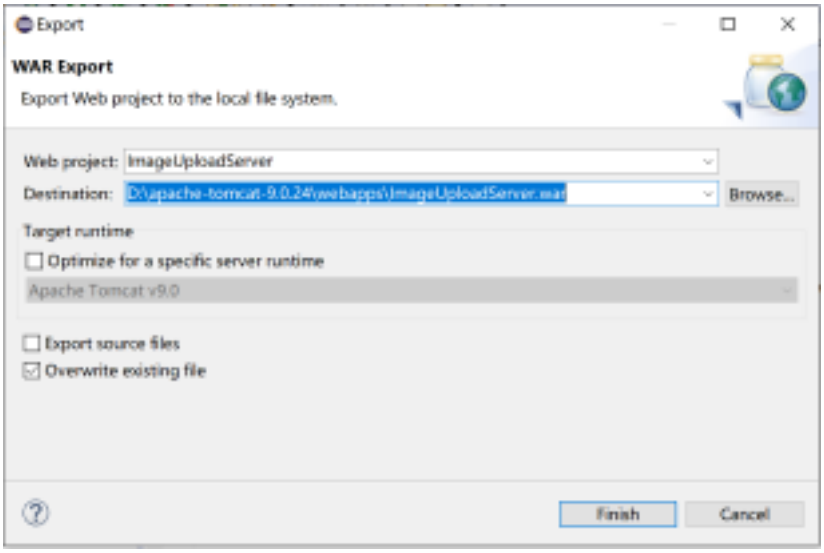

图 3-15　　导出项目war文件

- servlet的编写逻辑 MyServlet中主要实现方法为两个，doPost()处理客户端的post请求；convertStreamToString()用于将客户端的输入流转为string，方便模型的调用

```
/*
* 安卓端上传图片请求方法为HTTPpost
* 方法中接受图片请保存，使用模型进行图像识别doPost
*/
protected void doPost(HttpServletRequest
    request, HttpServletResponse response) throws
     ServletException, IOException {

//设置编码请求和应答的编码
response.setContentType("text/html,charset=UTF
    -8");
request.setCharacterEncoding("utf-8");
response.setCharacterEncoding("utf-8");
//应答输出流
PrintWriter out=response.getWriter();

//得到请求输入流
InputStream in=request.getInputStream();

```

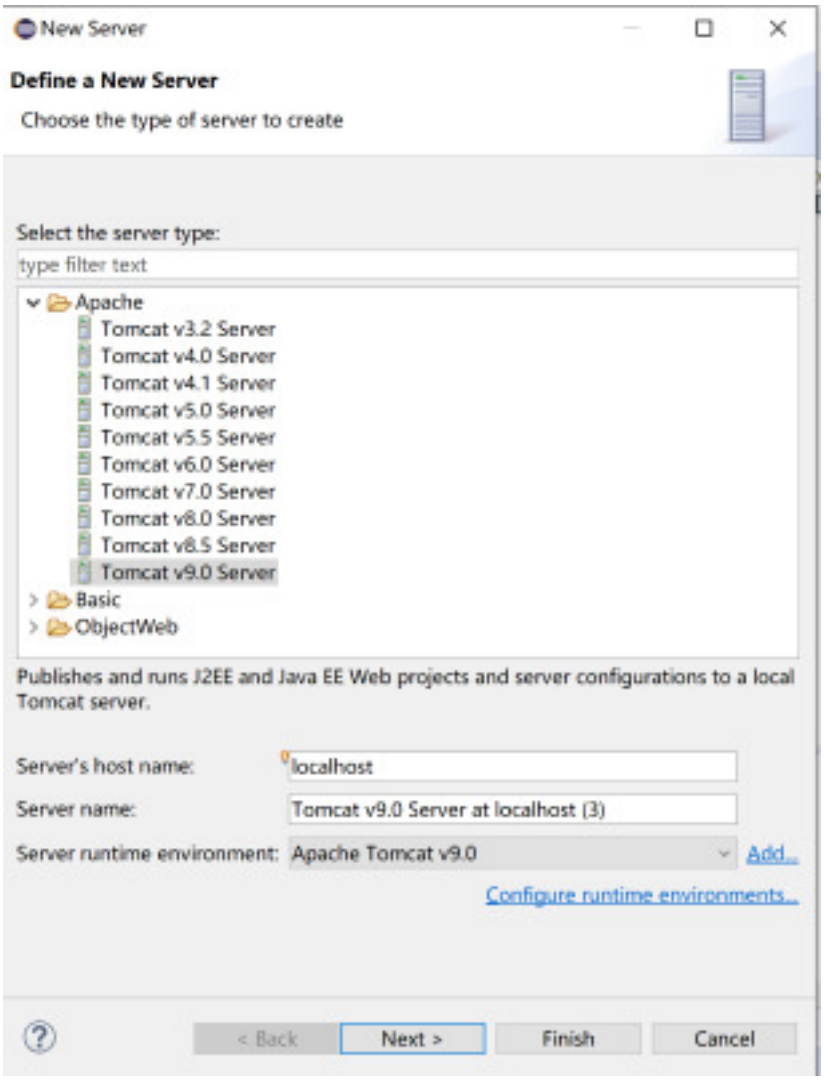

图 3-16　　创建server视图

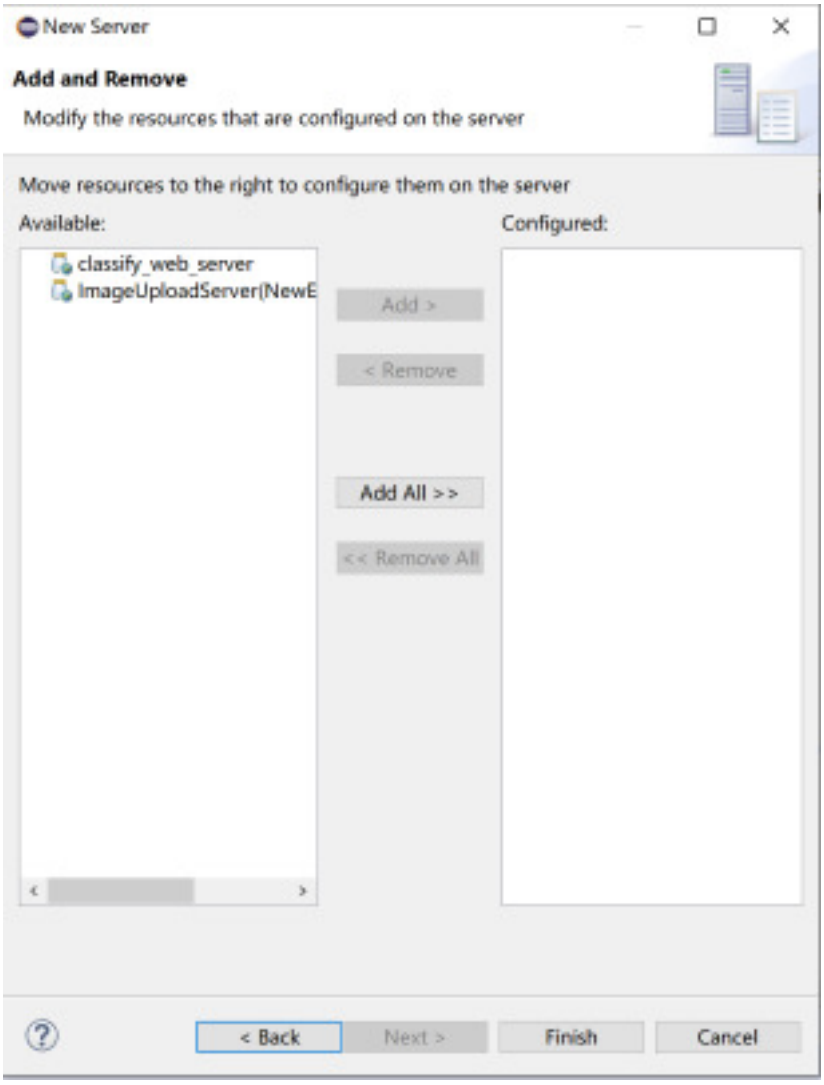
New Server
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
classify_web_server
ImageUploadServer(NewE
Add >
< Remove
Add All >>
<< Remove All
< Back
Next >
Finish
Cancel

图 3-17　　项目部署图

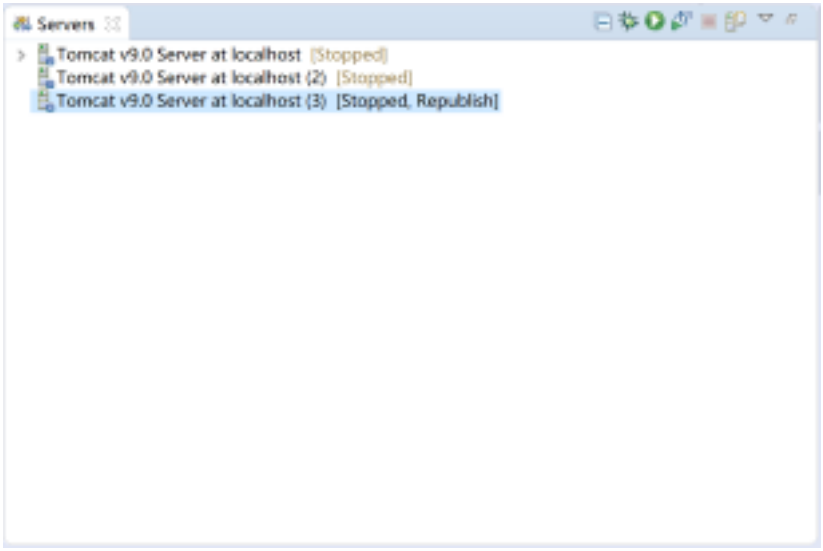

图 3-18　　server窗口

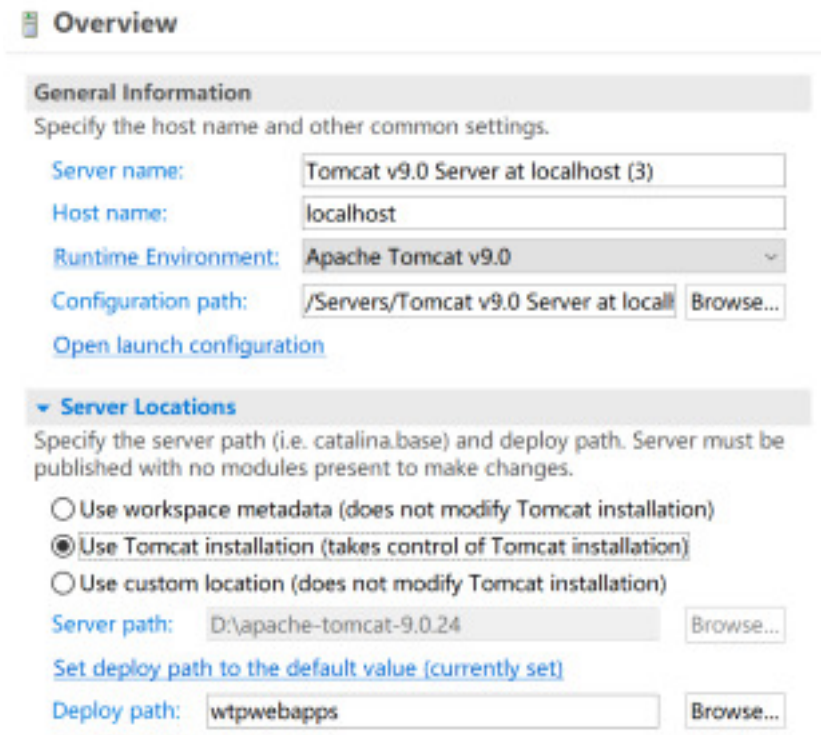

图 3-19　　server配置

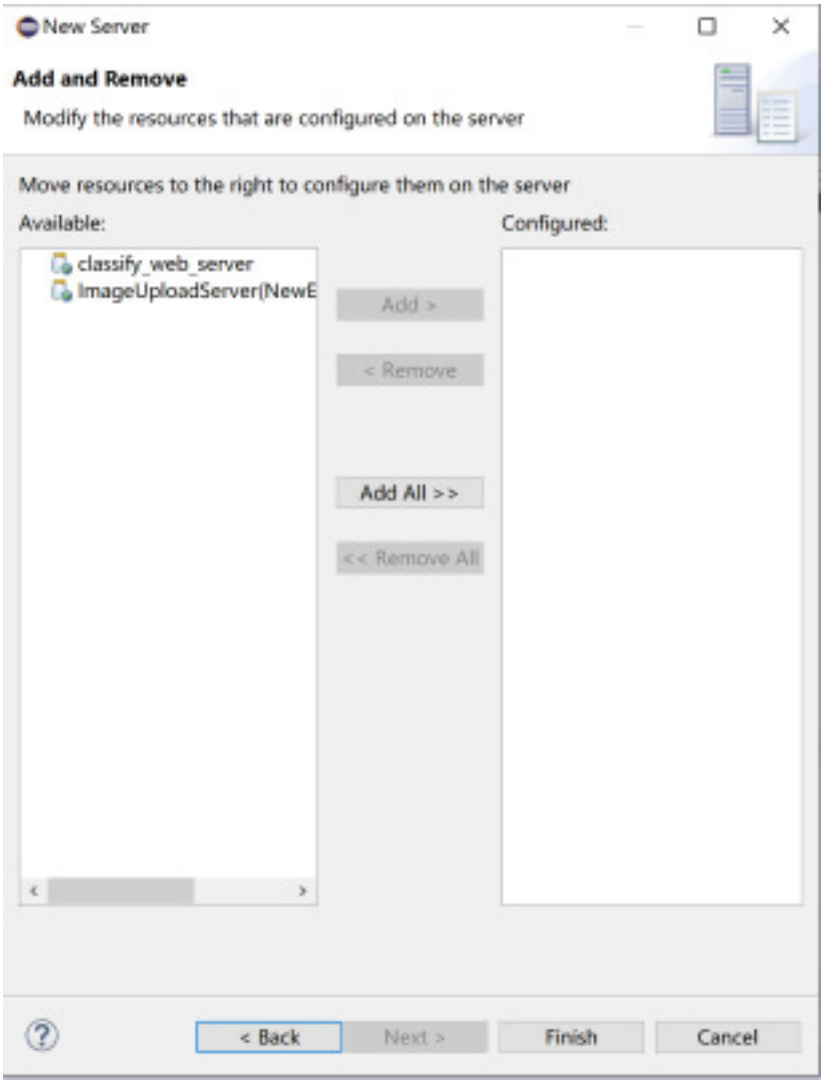
New Server
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
classify_web_server
ImageUploadServer(NewE
Add >
< Remove
Add All >>
<< Remove All
< Back
Next >
Finish
Cancel

图 3-20　　项目部署图

```
17 //请求的内容长度
18 int length=request.getContentLength();
19
20 //得到工程运行时的路径
21 project=request.getSession().getServletContext
       ().getRealPath("/");
22 //模型的放置路径tensorlfow
23 String model=project+"WEB-INF"+File.separator+"
       models";
24
25 System.out.println(model);
26 //调用模型javatensorlfow
27 try {
28
29 String upload;
30 String finalLabel;
31
32 //输入流转字符串结果
33 upload = convertStreamToString(in,"UTF-8",
       length);
34 //调用模型的标签结果
35 finalLabel=new LabelImage().excute(upload,
       model);
36 System.out.println(finalLabel);
37 //将标签结果写入应答
38 out.write(finalLabel);
39 }
40 catch(Exception e) {
41 e.printStackTrace();
42 }
43 }
```

```
1 //将转成字符串inputStream
2 /*
3 * @param 为请求输入流in
4 * @param 为字符编码charSet
```

```
 * @param 为请求内容长度length
 */
private String convertStreamToString(
    InputStream in,String charSet,int length)
    throws Exception, IOException {
ByteArrayOutputStream out=new
    ByteArrayOutputStream();
byte [] data=new byte[length];
int count=-1;

//读取输入流
while((count=in.read(data,0,length))!=-1) {
out.write(data,0,length);

}
data=null;
//返回对象String
return new String(out.toByteArray(),charSet);
}
```

- LabelImage的编写逻辑

  LabelIamge的代码编写参考了tensorlfow的github的程序示例，再根据服务器的功能进行修改，主要方法为excute(),该方法中对模型进行调用并返回概率最大的标签字符串

```
//调用模型并返回概率最大的标签label
/*
 * @param 为图像转为字符串的保存形式imageFile
 * @param 为模型的保存路径modeltensorflow
 */
public String excute(String imageFile,String
    model) throws IOException {
//模型所在路径tensorflow

System.out.println(model);

//读取模型数据并转为数组byte
```

```
12  byte [] graphDef=readAllBytesOrExit(Paths.get(
        model, "frozen_inference_graph.pb"));
13
14  //将模型的分类标签文件读入
15  List<String> labels=readAllLinesOrExit(Paths.
        get(model, "test_pb.txt"));
16
17  //图像转为数组byte
18  byte[] imageBytes=imageFile.getBytes();
19
20  //将图像转为模型训练所需的对象tensor
21  try (Tensor<Float> image=
        constructAndExecuteGraphToNormalizeImage(
        imageBytes)) {
22
23  //得到标签对应的浮点数概率数组
24  float[] labelProbablities=executeInceptionGraph
        (graphDef,image);
25
26  //得到最大的概率下标
27  int bestLabelIdx=maxIndex(labelProbablities);
28  //得到最大概率的标签字符串
29  String bestLabel=labels.get(bestLabelIdx);
30
31  //返回标签结果
32  return bestLabel;
33   }
34  }
```

2. ImageUploadServer

   在ImageUploadServer的src中， 有servlet-UploadImageServlet和 处 理java调用tensorflow模型的类LabelImage

   - servlet的编写逻辑 UploadImageServlet中主要实现方法为doGet()实现处理客户端的http请求, uploadImage()得到上传图片， 并将图片保存至指定路径文件夹下, usePython()通过runtime方法在java中执行python脚本，使用模型对上传图片进行推理, genericPath()产生上

传图片的随机路径，保证用户资料的安全性。

```
// 上传图片文件
private void uploadImage(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
String message = "";
try{
DiskFileItemFactory dff = new DiskFileItemFactory();
ServletFileUpload sfu = new ServletFileUpload(dff);
List<FileItem> items = sfu.parseRequest(request);
// 获取上传字段
FileItem fileItem = items.get(0);
// 更改文件名为唯一的
String filename = fileItem.getName();
if (filename != null) {
filename = IdGenertor.generateGUID() + "." + FilenameUtils.getExtension(filename);
}
// 生成存储路径
String storeDirectory = getServletContext().getRealPath("/files/images");
System.out.println(storeDirectory);

File file = new File(storeDirectory);
if (!file.exists()) {
file.mkdir();
}
String path = genericPath(filename, storeDirectory);
// 处理文件的上传
try {
fileItem.write(new File(storeDirectory + path,
```

```
    filename));
String filePath = "/files/images" + path + "/"
    + filename;
message = filePath;
} catch (Exception e) {
message = "上传图片失败";
}
} catch (Exception e) {
message = "上传图片失败";
} finally {
        response.getWriter().write(message);
                }
    }
```

```


//计算文件的存放目录
private String genericPath(String filename,
    String storeDirectory) {
int hashCode = filename.hashCode();
int dir1 = hashCode&0xf;
int dir2 = (hashCode&0xf0)>>4;

String dir = "/"+dir1+"/"+dir2;

File file = new File(storeDirectory,dir);
if(!file.exists()){
        file.mkdirs();
        }
return dir;
 }
```

```

//调用程序Javapython
private String usePython(String dir,String path
    ){
```

```
//脚本所在路径python
String pythonDir=dir+"get_test_result.py";
//图片文件所在路径
String image=path+"\\";

System.out.println(pythonDir);
System.out.println(image);

//另起进程，执行脚本python
Process pro;
String result="";

//执行脚本命令行
String [] arg=new String[] {"python",pythonDir,
    image};

try {
//执行文件python
pro=Runtime.getRuntime().exec(arg);

//用输入流输出流截取结果
BufferedReader in=new BufferedReader(new
    InputStreamReader(pro.getInputStream()));
String line=null;

//读取输出结果
while((line=in.readLine())!=null) {
System.out.println(line);
result+=line;
}
//关闭输入流和进程
in.close();
pro.waitFor();

}
catch(IOException e) {
```

```
38            e.printStackTrace();
39        }
40        catch(InterruptedException e) {
41            e.printStackTrace();
42        }
43        return result;
44            }
45    }
```

- LabelImage的编写逻辑
  这一版的excute方法中，传入的iamgeFile参数为图像文件的存储路径而不再是文件的字符串形式

```
1   public class LabelImage {
2
3   //调用模型并返回概率最大的标签label
4   public static String excute(String imageFile,
        String model) throws IOException {
5   //模型所在路径tensorflow
6   System.out.println(model);
7
8   byte [] graphDef=readAllBytesOrExit(Paths.get(
        model, "frozen_inference_graph.pb"));
9   List<String> labels=readAllLinesOrExit(Paths.
        get(model, "test_pb.txt"));
10
11  byte[] imageBytes=imageFile.getBytes();
12  try (Tensor<Float> image=
        constructAndExecuteGraphToNormalizeImage(
        imageBytes)) {
13  float[] labelProbablities=executeInceptionGraph
        (graphDef,image);
14  int bestLabelIdx=maxIndex(labelProbablities);
15  String bestLabel=labels.get(bestLabelIdx);
16  return bestLabel;
17
18  }
```

```
19 }
```

3.3 模型训练过程

3.3.1 数据集制作

3.3.1.1 确定数据集范围

本小组选择郑东图书馆一层作为图像识别范围，具体原因如下：

1. 专业课程设计时间有限，不足以将整个郑东图书馆的图像识别做完。
2. 郑东图书馆的一层相较于郑东图书馆的第二、三、四层来说，标志物较明显，比较容易通过图像识别来确定位置。

选择好大致识别范围后，我们又划分了几个区域作为有标志物的识别区域。同时利用这些标志物作为识别主体来进行深度学习训练模型。

这几个区域分别是自助借还书区、新技术体验区（以两个标志物同时作为识别主体，只要识别出其中一个即可认定为新技术体验区）、检索区、文化展区（因上传及下载文件过程中数据损坏导致并未完成此区域的训练）以及大厅。至此我们确定了数据集范围。

3.3.1.2 拍摄照片

小组内五人分组拍照，对每个要进行训练的区域都拍摄了900余张照片，作为基础数据集。

3.3.1.3 裁剪图片

由于找到的教程对于图片大小有要求，即高度不超过600像素。并且由于图片数目较多，所以使用python脚本来对图片大小进行批量修改。 python脚本核心代码如下。

```
1     parser = argparse.ArgumentParser()
2     parser.add_argument('--image_dir', help='Directory of
          images to resize')
3     args = parser.parse_args()
4     image_dir = os.getcwd() + "/" + args.image_dir
5     for f in os.listdir(image_dir):
6         filename = os.fsdecode(f)
7         image = Image.open(image_dir + '/' + filename)
8         print(image_dir + '/' + filename)
9         height, width = image.size
10        if width > 600:
11            resize_amt = 600 / width
```

```
            new_height = int(round(height * resize_amt))
            image = image.resize((new_height, 600))
            image.save(os.getcwd() + "/" + filename)
```

首先取出图片，判断图片高度，若大于600像素则将高度改为600像素，反之则不做处理。

3.3.1.4 扩大数据集

扩大数据集，利用python脚本对拍到的照片进行镜像翻转、旋转一定角度、添加椒盐噪声（一种随机出现的白点或者黑点）、添加高斯噪声（指它的概率密度函数服从高斯分布的一类噪声）、调整亮度的操作，使得数据集扩大为原来的六倍，共近3万张图片。部分核心代码如下。

```
    def SaltAndPepper(src, percetage)://添加椒盐噪声
        SP_NoiseImg = src.copy()
        SP_NoiseNum = int(percetage * src.shape[0] * src.
            shape[1])
        for i in range(SP_NoiseNum):
            randR=np.random.randint(0, src.shape[0] - 1)
            randG=np.random.randint(0, src.shape[1] - 1)
            randB=np.random.randint(0, 3)
            if np.random.randint(0, 1) == 0:
                SP_NoiseImg[randR, randG, randB] = 0
            else:
                SP_NoiseImg[randR, randG, randB] = 255
        return SP_NoiseImg
```

```
    def addGaussianNoise(image, percetage)://添加高斯噪声
        G_Noiseimg = image.copy()
        w = image.shape[1]
        h = image.shape[0]
        G_NoiseNum = int(percetage * image.shape[0] *
            image.shape[1])
        for i in range(G_NoiseNum):
            temp_x = np.random.randint(0, h)
            temp_y = np.random.randint(0, w)
            G_Noiseimg[temp_x][temp_y][np.random.randint
                (3)] = np.random.randn(1)[0]
```

```
        return G_Noiseimg
```

```
    def darker(image, percetage=0.9)://更改亮度
        image_copy = image.copy()
        w = image.shape[1]
        h = image.shape[0]
        # get darker
        for xi in range(0, w):
            for xj in range(0, h):
                image_copy[xj, xi, 0]=int(image[xj, xi,
                    0] * percetage)
                image_copy[xj, xi, 1]=int(image[xj, xi,
                    1] * percetage)
                image_copy[xj, xi, 2]=int(image[xj, xi,
                    2] * percetage)
        return image_copy
```

3.3.1.5 标识特征物

利用labelImg软件对想要识别的物体进行标注。选择CreateRectbox，框住标志物，输入特定的标签，如图3-21所示。

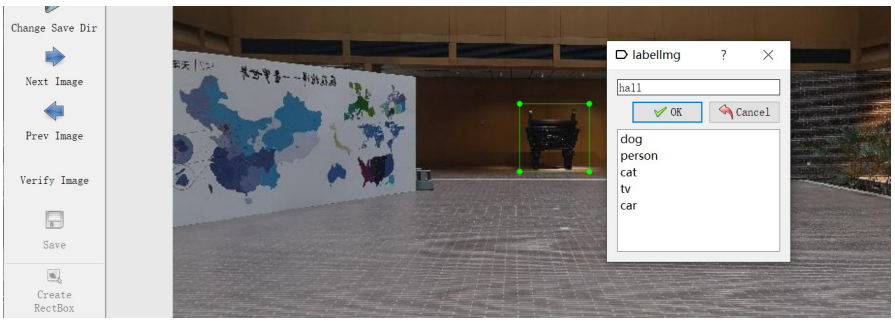

图 3-21　　labelImg打框

labelImg软件会在指定的目录下生成对应的xml文件，xml文件格式如图3-22所示。其中path为原图片所在路径， name为输入的特定标签。

3.3.1.6 数据集划分

将生成的xml文件划分为训练集、验证集和测试集三部分，分别存储于三个文件夹中。

```
    import os
    import random
```

```
<?xml version="1.0"?>
- <annotation>
    <folder>picture</folder>
    <filename>IMG_4727.JPG</filename>
    <path>C:\Users\82305\Desktop\lily\picture\IMG_4727.JPG</path>
  - <source>
      <database>Unknown</database>
    </source>
  - <size>
      <width>800</width>
      <height>600</height>
      <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
      <name>Borrow machine</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
```

图 3-22　　xml文件格式

```
import time
import shutil

xmlfilepath=r'merged_xml'
saveBasePath=r"./annotations"

trainval_percent=0.9
train_percent=0.85
total_xml = os.listdir(xmlfilepath)
num=len(total_xml)
list=range(num)
tv=int(num*trainval_percent)
tr=int(tv*train_percent)
trainval= random.sample(list ,tv)
train=random.sample(trainval ,tr)
print("train and val size",tv)
print("train size",tr)
start = time.time()
test_num=0
val_num=0
train_num=0
for i  in list:
    name=total_xml[i]
```

```
26          if i in trainval:  #train and val set
27              if i in train:
28                  directory="train"
29                  train_num+=1
30                  xml_path = os.path.join(os.getcwd(), '
                        annotations/{}'.format(directory))
31                  if(not os.path.exists(xml_path)):
32                      os.mkdir(xml_path)
33                  filePath=os.path.join(xmlfilepath,name)
34                  newfile=os.path.join(saveBasePath,os.path
                        .join(directory,name))
35                  shutil.copyfile(filePath, newfile)
36
37              else:
38                  directory="validation"
39                  xml_path = os.path.join(os.getcwd(), '
                        annotations/{}'.format(directory))
40                  if(not os.path.exists(xml_path)):
41                      os.mkdir(xml_path)
42                  val_num+=1
43                  filePath=os.path.join(xmlfilepath,name)
44                  newfile=os.path.join(saveBasePath,os.path
                        .join(directory,name))
45                  shutil.copyfile(filePath, newfile)
46          else:  #test set
47              directory="test"
48              xml_path = os.path.join(os.getcwd(), '
                    annotations/{}'.format(directory))
49              if(not os.path.exists(xml_path)):
50                  os.mkdir(xml_path)
51              test_num+=1
52              filePath=os.path.join(xmlfilepath,name)
53              newfile=os.path.join(saveBasePath,os.path.
                    join(directory,name))
54              shutil.copyfile(filePath, newfile)
55
```

```
    # End time
    end = time.time()
    seconds=end-start
    print("train total : "+str(train_num))
    print("validation total : "+str(val_num))
    print("test total : "+str(test_num))
    total_num=train_num+val_num+test_num
    print("total number : "+str(total_num))
    print( "Time taken : {0} seconds".format(seconds))
```

3.3.1.7 转化为record文件

由于xml文件并不能用于训练模型，因此要将划分出的三类xml文件分别转换为record文件，核心代码如下。

```
def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', '
       class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=
       column_name)
    return xml_df
```

3.3.1.8 注意事项

准备数据集是训练模型的基础，在准备数据集的过程中，我们由于经验不足出现了一些问题，在此我们将这些问题总结出来，希望如果有同学想从头做深度学习的话能够注意一下这些，可以少走弯路并且节省时间。

1. xml文件并不是独立的文件，它不包含图片中的所有信息。因此不能够删除原图，两者都存在才能有效地转换为record文件。
2. xml文件中有个参数是图片所在的路径，因此在对图片进行标框分工之前，最好对图片的路径进行统一规定。否则就会出现xml文件无法与图片对应，出现这样的情况就只能够通过python脚本对xml中的文件路径进行统一修改，导致投入大量的时间。修改xml文件内容的代码如下：

```
        wait
```

3. 将xml文件以及图片转化为record文件这项工作最好放在个人电脑上做，在jupyter上运行该python脚本速度十分缓慢，过于拖进度。
4. 不要把数据集上传后在下载，在这样的过程中会造成一些不可恢复的丢失。

3.3.2 环境搭建

3.3.2.1 Windows环境搭建

TensorFlow是Google研发的第二代人工智能学习系统，本项目使用 TensorFlow的models模块，其中的Object Detection模块内置了许多关于物体检测的先进模型（见图），均为使用COCO数据集预训练而成，其冻结权重可以被直接加载使用，为基于自制数据集的模型重训练提供了便利。

基于移植移动设备的考虑，使用了其中的ssd-MobileNets模型作为初始模型，MobileNets是为移动和嵌入式设备提出的高效模型，基于流线型架构，使用深度可分离卷积来构建轻量级深度神经网络。

1. TensorFlow的安装

   TensorFlow既能支持CPU，也能支持GPU。前者的环境需求较为简单，需要具备的安装条件如下：

   (a) VisualC++ Redistributable for Visual Studio 2015 （或更高版本）

   (b) python3.5（或更高版本）

   打开Windows的命令行窗口，输入以下命令：

```
pip3 install --upgrade tensorflow
```

   安装完成后，进入python环境验证是否安装成功。

```
>>> import tensorflow as tf
```

```
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
```

如果能正常输出hello字符串，则说明安装成功。

```
Hello, TensorFlow!
```

2. Object Detection API安装

(a) 克隆models模块

```
git clone https://github.com/tensorflow/models
```

(b) 安装库文件

```
sudo apt-get install protobuf-compiler python-pil python-lxml python-tk
pip install --user Cython
pip install --user contextlib2
pip install --user matplotlib
```

(c) protobuf配置与测试需要将模块中/proto下所有proto文件编译成python文件，首先cd到/models/research目录下执行如下命令：

```
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
protoc object_detection/protos/*.proto --python_out=.
```

由于第二条命令无法成功执行，思考解决方案后编写python脚本单条重复执行编译文件成功。

```
import os

path_url = os.path.join(os.getcwd(),r"object_detection\protos")
print("proto path:",path_url)

for file in os.listdir(path_url):
    cmd = "protoc object_detection\protos\{} --python_out=."
    if file.endswith(".proto"):
        command = cmd.format(file)
```

```
            print("excuting command:",command)
            os.popen(command)
```

最后，运行测试脚本测试是否配置成功。

```
python object_detection/builders/
    model_builder_test.py
```

在屏幕打印出“OK”即为成功。

3.3.2.2 云端部署

出于训练速度的考虑，本项目使用了Intel提供的云计算资源进行模型训练。使用Jupyter Notebook与之完成连接，并如以上步骤配置TensorFlow环境及解压需要的zip库，将图片数据、相应的record文件、models文件夹上传至Jupyter中。

需注意的是，Jupyter不支持上传文件夹及解压过大的压缩文件，且上传过程时常不够稳定而导致中断，所以需要将文件分卷压缩为约100M大小再进行上传。上传成功后，在Jupyter终端运行如下命令进行解压缩：

```
cat x.zip* > x.zip
unzip x.zip
```

至此，模型训练的准备工作就已经全部完成。

3.3.3 模型训练

3.3.3.1 修改设置文件

Object Detection API是依赖一种特殊的设置文件进行训练的，参考其中已给出的示例进行config文件的修改，即可使其与自制数据集适应。

在模块中找到ssd_inception_v2_pets.config文件打开，并作如下修改：

1. 将num_classes修改为本项目的类别数。

```
ssd {
    num_classes: 5 分为类#5
    box_coder {
    faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
        }
    }
```

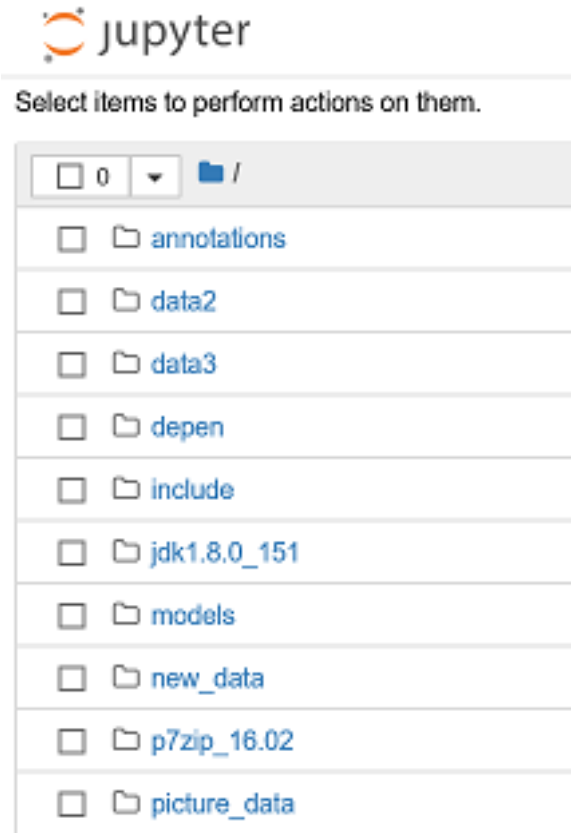

图 3-23　　将数据上传Jupyter Notebook

2. 将所有标为PATH_TO_BE_CONFIGURED的路径修改为相应的正确路径。

```
train_input_reader: {
  tf_record_input_reader {
    input_path: "new_data/whsyxt_train.tfrecord"
  }
  label_map_path: "new_data/label_map.pbtxt"
}
eval_input_reader: {
  tf_record_input_reader {
    input_path: "new_data/whsyxt_validation.tfrecord
      "
  }
  label_map_path: "new_data/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

3. 调整训练参数。其中的部分关键参数解读如下：
   (a) batch_size: 批尺寸，主要用于批梯度下降算法中，批梯度下降算法是每次迭代都遍历批中的所有样本，由批中的样本共同决定最优的方向，Batch_Size 正是批中的样本数量。
   (b) learning_rate: 学习率，决定了参数移动到最优值的速度快慢，如果学习率过大，很可能会越过最优值导致函数无法收敛，甚至发散；反之，如果学习率过小，优化的效率可能过低，算法长时间无法收敛，也易使算法陷入局部最优（非凸函数不能保证达到全局最优）。合适的学习率应该是在保证收敛的前提下，能尽快收敛。
   (c) decay_steps: 改变学习率时的迭代次数，每迭代某一特定次数，对学习率进行一次改变。
   (d) decay_factor: 改变学习率时的百分率，每次对学习率进行更改时，都与该参数相乘来决定。
   (e) num_steps: 总共进行的迭代次数

```
train_config: {
  batch_size: 48 每次训练采用张图片#48
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
```

```
      exponential_decay_learning_rate {
        initial_learning_rate: 0.001    #0.004
        decay_steps: 8000
        decay_factor: 0.95
      }
    }
    momentum_optimizer_value: 0.9
    decay: 0.9
    epsilon: 1.0
  }
}
#fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED/
    model.ckpt"
from_detection_checkpoint: false
load_all_detection_checkpoint_vars: true
num_steps: 20000
data_augmentation_options {
  random_horizontal_flip {
  }
}
data_augmentation_options {
  ssd_random_crop {
  }
}
max_number_of_boxes: 50
}
```

3.3.3.2 执行训练

相关参数设置完成后，就可以开始执行模型的训练。

调用train.py并传入路径参数如下：

```
python object_detection/train.py \
--logtostderr \
--pipeline_config_path=new_data/ssd_inception_v2_pets
    .config \
--train_dir=new_data/
```

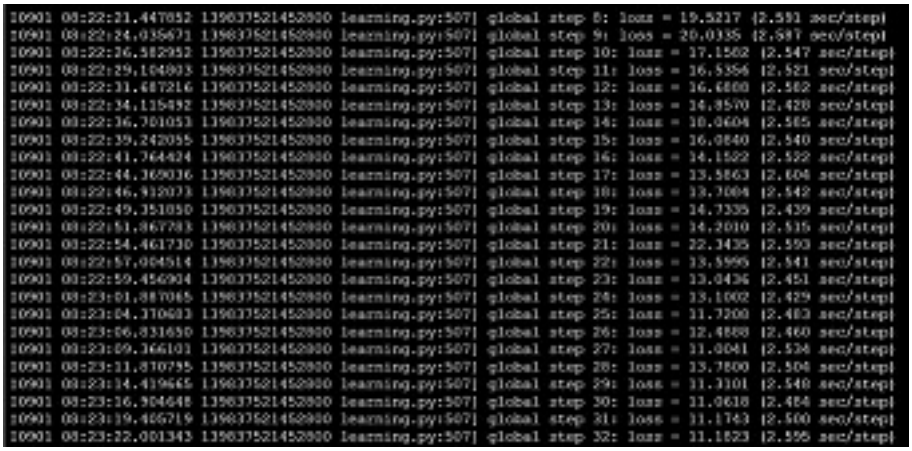

图 3-24　　训练过程

即可看到训练开始进行，随着迭代次数的增加，损失率不断减小，说明学习过程相对顺利。

3.3.4　模型固化及导出

完成训练后，会在数据文件夹下生成ckpt模型文件，将计算图的结构和图上参数取值分成了不同的文件存储，以便后续的再训练工作。

```
model.ckpt-${CHECKPOINT_NUMBER}.data-00000-of-00001,
model.ckpt-${CHECKPOINT_NUMBER}.index
model.ckpt-${CHECKPOINT_NUMBER}.meta
```

这些文件无法直接使用，由于要交付给服务器端，需将已有的ckpt文件固化为pb模型导出。

调用export_inference_graph.py并传入路径参数如下：

```
python object_detection/export_inference_graph.py \
--input_type=image_tensor \
--pipeline_config_path=new_data/ssd_inception_v2_pets
    .config \
--trained_checkpoint_prefix=new_data/model.ckpt-16521
     \
--output_directory=new_data/result
```

即可在new_data/result文件夹下生成固化后的pb模型。

3.3.5　结果验证

采用单张图片进行验证，返回一张图片作为结果，若识别成功，将在图片

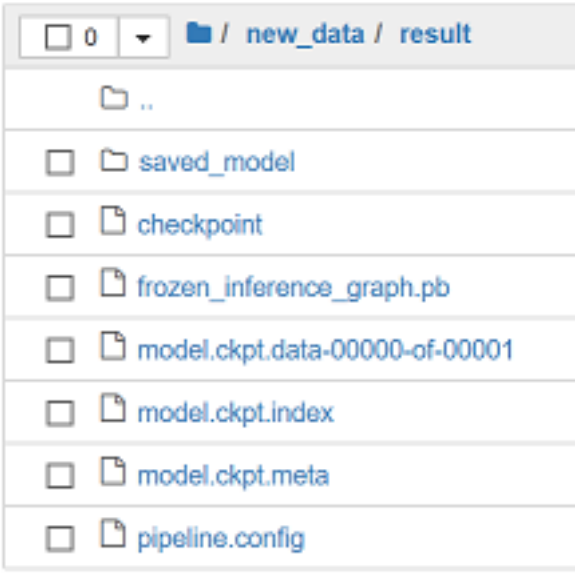

图 3-25　　模型固化

上框注出推理类别。

```
import cv2
import numpy as np
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import
   visualization_utils as vis_util


class TOD(object):
    def __init__(self):
        self.PATH_TO_CKPT = '/new_data/result/
           frozen_inference_graph.pb'
        self.PATH_TO_LABELS = '/new_data/label_map.
           pbtxt'
        self.NUM_CLASSES = 5
        self.detection_graph = self._load_model()
        self.category_index = self._load_label_map()

    def _load_model(self):
        detection_graph = tf.Graph()
```

```
18              with detection_graph.as_default():
19                  od_graph_def = tf.GraphDef()
20                  with tf.gfile.GFile(self.PATH_TO_CKPT, '
                        rb') as fid:
21                      serialized_graph = fid.read()
22                      od_graph_def.ParseFromString(
                            serialized_graph)
23                      tf.import_graph_def(od_graph_def,
                            name='')
24              return detection_graph
25
26          def _load_label_map(self):
27              label_map = label_map_util.load_labelmap(self
                    .PATH_TO_LABELS)
28              categories = label_map_util.
                    convert_label_map_to_categories(label_map,
                    max_num_classes=self.NUM_CLASSES,
                    use_display_name=True)
29              category_index = label_map_util.
                    create_category_index(categories)
30              return category_index
31
32          def detect(self, image):
33              with self.detection_graph.as_default():
34                  with tf.Session(graph=self.
                        detection_graph) as sess:
35                      # Expand dimensions since the model
                            expects images to have shape: [1,
                            None, None, 3]
36                      image_np_expanded = np.expand_dims(
                            image, axis=0)
37                      image_tensor = self.detection_graph.
                            get_tensor_by_name('image_tensor:0'
                            )
38                      boxes = self.detection_graph.
                            get_tensor_by_name('detection_boxes
```

```
                :0')
            scores = self.detection_graph.
                get_tensor_by_name('
                detection_scores:0')
            classes = self.detection_graph.
                get_tensor_by_name('
                detection_classes:0')
            num_detections = self.detection_graph
                .get_tensor_by_name('num_detections
                :0')
            # Actual detection.
            (boxes, scores, classes,
                num_detections) = sess.run(
                 [boxes, scores, classes,
                     num_detections],
                 feed_dict={image_tensor:
                     image_np_expanded})
            # Visualization of the results of a
                detection.
            vis_util.
                visualize_boxes_and_labels_on_image_array
                (
                 image,
                 np.squeeze(boxes),
                 np.squeeze(classes).astype(np.
                     int32),
                 np.squeeze(scores),
                 self.category_index,
                 use_normalized_coordinates=True,
                 line_thickness=8)

        cv2.namedWindow("detection", cv2.
            WINDOW_NORMAL)
        cv2.imshow("detection", image)
        cv2.waitKey(0)
```

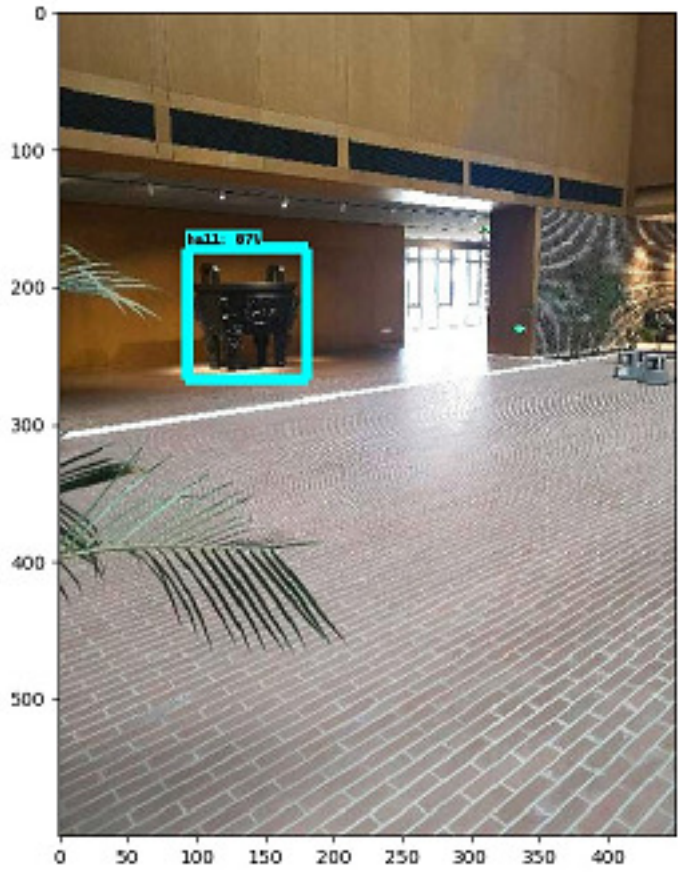

图 3-26　　模型验证

```
60  if __name__ == '__main__':
61      image = cv2.imread('/picture_data/
            IMG_20190820_161903.jpg4.jpg')
62      detecotr = TOD()
63      detecotr.detect(image)
```

返回图片如下：

3.3.6　问题及反思

在模型的训练过程中发生了一些问题，究其根本大多出于数据集处理的不严谨，以及数据在平台间传送不当导致的数据丢失和一系列不便。以此为教训，今后应更加认真处理数据问题。

第四章　项目测试

4.1　概述

针对目前已导出的迭代16521次的模型进行测试，具体为对此前划分出的6000余张测试照片进行推理，将结果存储于csv文件中返回，并进行统计分析。

4.2　测试过程

4.2.1　数据处理

首先需要从原有的照片集中筛选出用于测试的部分并单独存储于一个文件夹，同时在文件名中加上原有的类别名以起到对比推理结果正误的作用，使用到的脚本代码如下：

```
from PIL import Image
import os.path
import glob
from xml.etree.ElementTree import parse, Element
#import xml.dom.minidom as xmldom

annotations_test_dir = "annotations/test/"
Images_dir = "picture_data"
test_images_dir = "test_images_new"
i = 0
for xmlfile in os.listdir(annotations_test_dir):
    if os.path.isdir(xmlfile):
        print("donot worry")
        continue
    (filepath, tempfilename) = os.path.split(xmlfile)
    (shotname, extension) = os.path.splitext(
        tempfilename)
    xmlname = shotname
#     print(xmlname)
    newStr = os.path.join(annotations_test_dir,
        xmlfile)
    dom = parse(newStr)
```

```
#       dom_obj = xmldom.parse(filepath)
        root = dom.getroot()
        label = ""
        for obj in root.iter('object'):
            label = obj.find('name').text

        for jpgfile in os.listdir(Images_dir):
            (filepath, tempfilename) = os.path.split(
                jpgfile)
            (jpgname, extension) = os.path.splitext(
                tempfilename)
            if jpgname == xmlname:
                if os.path.exists(Images_dir+"/" +
                    jpgname + ".jpg"):
                    img = Image.open(Images_dir+"/" +
                        jpgname + ".jpg")
                    dst = os.path.join(
                        test_images_dir, jpgname + '_'
                        + label +'.jpg')
                    img.save(os.path.join(
                        test_images_dir, os.path.
                        basename(jpgfile)))
                    os.rename(os.path.join(
                        test_images_dir, os.path.
                        basename(jpgfile)),dst)
                    i += 1
                    print(i)
print(i)
```

4.2.2 执行测试

完成以上步骤后，就可以调用以下脚本开始测试：

```
# -*- coding: utf-8 -*-
import os
from PIL import Image
import time
```

```
import tensorflow as tf
from PIL import Image
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import zipfile
import time
import pandas as pd

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from object_detection.utils import label_map_util
from object_detection.utils import
    visualization_utils as vis_util

flags = tf.app.flags
flags.DEFINE_string('test_images_path', '',
                    'Path to a test image.')
flags.DEFINE_string('model_path', '',
                    'Path to a model file.')
flags.DEFINE_string('label_path', '',
                    'Path to a label file.')

FLAGS = flags.FLAGS

PATH_TO_TEST_IMAGES = FLAGS.test_images_path # "
    test_images/"
#MODEL_NAME = 'new_data_copy'
PATH_TO_CKPT = FLAGS.model_path # MODEL_NAME + '/
    frozen_inference_graph.pb'
PATH_TO_LABELS = FLAGS.label_path # MODEL_NAME+'/
    label_map.pbtxt'
NUM_CLASSES = 5
```

```
37
38      def load_image_into_numpy_array(image):
39          (im_width, im_height) = image.size
40          return np.array(image.getdata()).reshape((
                im_height, im_width, 3)).astype(np.uint8)
41
42      def save_object_detection_result():
43          data_list = []
44          column_name = ['picName', 'label']
45          IMAGE_SIZE = (12, 8)
46          detect_result = ""
47          j = 0
48          # 读取模型
49          detection_graph = tf.Graph()
50          with detection_graph.as_default():
51              od_graph_def = tf.GraphDef()
52              # loading ckpt file to graph
53              with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as
                    fid:
54                  serialized_graph = fid.read()
55                  od_graph_def.ParseFromString(
                        serialized_graph)
56                  tf.import_graph_def(od_graph_def, name=''
                        )
57          # Loading label map
58          label_map = label_map_util.load_labelmap(
                PATH_TO_LABELS)
59          categories = label_map_util.
                convert_label_map_to_categories(label_map,
                max_num_classes=NUM_CLASSES, use_display_name=
                True)
60          category_index = label_map_util.
                create_category_index(categories)
61          # Helper code
62          with detection_graph.as_default():
63              with tf.Session(graph=detection_graph) as
```

```
        sess:
64        start = time.time()
65        for test_image in os.listdir(
              PATH_TO_TEST_IMAGES):
66          image = Image.open(
                PATH_TO_TEST_IMAGES + test_image)
67          # the array based representation of
                the image will be used later in
                order to prepare the
68          # result image with boxes and labels
                on it.
69          image_np =
                load_image_into_numpy_array(image)
70          # Expand dimensions since the model
                expects images to have shape: [1,
                None, None, 3]
71          image_np_expanded = np.expand_dims(
                image_np, axis=0)
72          image_tensor = detection_graph.
                get_tensor_by_name('image_tensor:0'
                )
73          # Each box represents a part of the
                image where a particular object was
                 detected.
74          boxes = detection_graph.
                get_tensor_by_name('detection_boxes
                :0')
75          # Each score represent how level of
                confidence for each of the objects.
76          # Score is shown on the result image,
                 together with the class label.
77          scores = detection_graph.
                get_tensor_by_name('
                detection_scores:0')
78          classes = detection_graph.
                get_tensor_by_name('
```

```
        detection_classes:0')
79      num_detections = detection_graph.
        get_tensor_by_name('num_detections
        :0')
80      # Actual detection.
81      (boxes, scores, classes,
        num_detections) = sess.run(
82       [boxes, scores, classes,
            num_detections],
83       feed_dict={image_tensor:
            image_np_expanded})
84      # Visualization of the results of a
        detection.
85      vis_util.
        visualize_boxes_and_labels_on_image_array
        (
86       image_np,
87       np.squeeze(boxes),
88       np.squeeze(classes).astype(np.
            int32),
89       np.squeeze(scores),
90       category_index,
91       use_normalized_coordinates=True,
92       line_thickness=8)
93
94      final_score = np.squeeze(scores)
95      count = 0
96      for i in range(100):
97          if scores is None or final_score[
            i] > 0.5:
98              count = count + 1
99      print()
100     print("the count of objects is: ",
        count)
101     (im_width, im_height) = image.size
102     for i in range(count):
```

```
                    detect_result = category_index[
                        classes[0][i]]['name']
                picName = test_image.split('/')[-1]
                if count == 0:
                    value = (picName,'null')
                else:
                    value = (picName,detect_result)
                data_list.append(value)
                data_df = pd.DataFrame(data_list,
                    columns=column_name)
                j += 1
                print(value)
                print(j)
        data_df.to_csv('./new_detect_result', index=None)
        print('Successfully create csv.')

    save_object_detection_result()
```

执行以下调用命令：

```
~/models/research$ python get_all_test_result.py --
    test_images_path=test_images/ --model_path=
    new_data_copy/frozen_inference_graph.pb --label_path=
    new_data_copy/label_map.pbtxt
```

4.2.3 测试结果

对测试生成的csv表格进行统计分析，得到结果如下表所示：

表 4-1 各类别推理准确率

类别	测试总数	正确识别数	未识别数	准确率
借还书区	1182	888	257	0.751269036
检索区	1244	641	568	0.515273312
新技术体验区1	1135	571	335	0.5030837
大厅	1430	1186	201	0.829370629
新技术体验区2	1475	1420	55	0.962711864

表 4-2　　类别间交叉识别错误结果

类别	借还书区	检索区	新技术体验区1	大厅	新技术体验区2
借还书区	/	0	0	0	37
检索区	29	/	0	0	6
新技术体验区1	0	0	/	0	229
大厅	15	3	0	/	0
新技术体验区2	0	0	0	0	/

4.3　测试分析

1. 本次测试结果准确率从0.50至0.96不等，识别效果最好的区域与最差的区域均为新技术体验区，这也是唯一选取了两个特征物的区域。
2. 其次，检索区的识别效果也不甚理想，与新技术体验区对比，考虑到以下几种影响因素：照片画质；特征物框选；拍摄角度过多等。
3. 由于借还书区的数据存在约一千张的原片丢失，测试结果理论上应该更好。
4. 改进方案设想：考虑更具代表性的特征物；直接选取场景全貌作为特征；划分更多种类兼顾各个拍摄角度；对学习率及其他参数进行进一步调整。

第五章　使用手册

5.1　从安卓APP上传图片

操作步骤如下表所示：

1. 点击应用进入主界面，点击选择图片，选择拍照或从相册选取(见图5-1)。
2. 选好图片后裁剪图片，突出识别主体(见图5-2)。
3. 点击确认后回到主界面，点击上传(见图5-3)。
4. 跳转到结果界面，点击获取位置信息，得到所处位置(见图5-4)。

图 5-1　选择图片

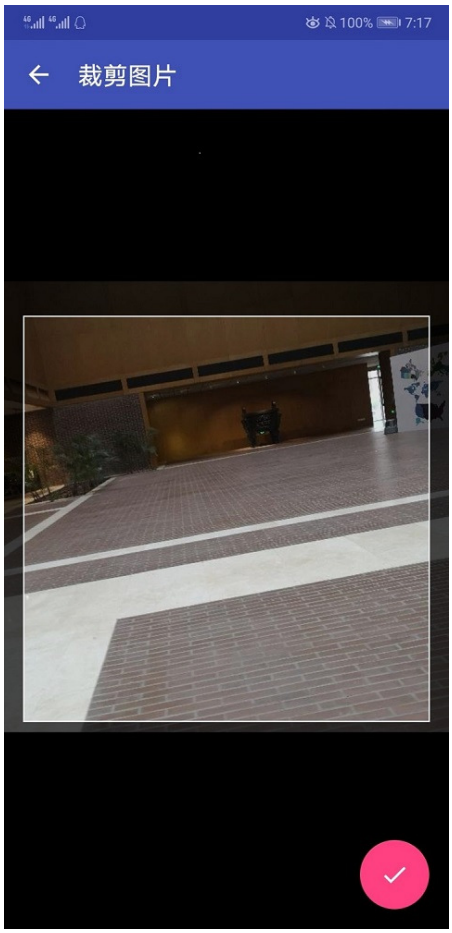

图 5-2　裁剪图片

5.2　模型的调用

5.2.1　大批量测试

1. 在models/research文件下新建文件夹model和test_images。

图 5-3　　点击上传

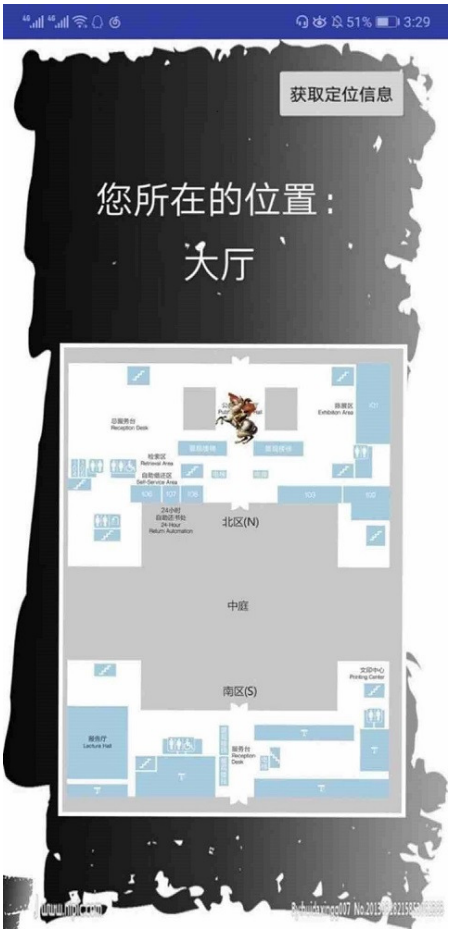

图 5-4　　获取位置信息

2. 将pb模型和label.pbtxt置于model文件夹下，测试图片集置于test_images文件夹下。
3. 在models/research执行以下命令：

```
python get_all_test_result.py --test_images_path=test_images/ --model_path=model/model.pb --label_path=model/label.pbtxt
```

5.2.2 小范围测试

1. 所需文件结构与大批量测试相同。
2. 在models/research执行以下命令：

```
python get_test_result.py --test_images_path=./test_images/
```