伪云端部署Paddle模型实现单目标垃圾分类

1. 伪云端实现：

使用花生壳5，对本地IP实现内网穿透（IP地址及端口的映射），实现公网与内网的数据通信交互，与之后介绍的基于Flask的部署方式有关。



图 1 花生壳5内网穿透示例

1. 服务器端模型部署：

根据这篇“手把手教你部署深度学习模型” <https://www.jianshu.com/p/43da2553a2fb>提供的思路，实现Paddle模型的部署。

目前，我们已经将在AI Studio上训练好的模型保存至本地，那么根据test.py中的代码，我们可以实现Paddle模型的**“一次性推断”**。其中，在Line 4导入的model1是由PaddlePaddle提供的残差神经网络ResNet模型库；Line 46中定义的***read\_image()***方法中，需要手动添加一个等待预测的图片的相对路径或绝对路径。经过模型的计算，会将预测的类别存入变量lab中，并且输出。

|  |
| --- |
| **import** paddle.fluid **as** fluid  **import** numpy **as** np **from** PIL **import** Image, ImageDraw  **import** model1  **def** resize\_img(img, target\_size):  img = img.resize(target\_size[1:], Image.ANTIALIAS)  **return** img **def** read\_image(img\_path):  target\_size = [3, 224, 224]  origin=Image.open(img\_path)  *#origin = img\_path.copy()* img = resize\_img(origin, target\_size)  **if** img.mode != **'RGB'**:  img = img.convert(**'RGB'**)  img = np.array(img).astype(**'float32'**).transpose((2, 0, 1)) *# HWC to CHW  #img = normalize\_image(img)* img -= 127.5  img \*= 0.007843  img = img[np.newaxis, :]  **return** origin,img  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  target\_size = [3, 224, 224]  *# 定义输入层* image = fluid.layers.data(name=**'image'**, shape=[3, 224, 224], dtype=**'float32'**)  mod = model1.ResNet50()  model = mod.net(image, 4)  *# 模型位置* path = **"best\_model"** *# 定义执行器* place = fluid.CUDAPlace(0)  exe = fluid.Executor(place)  *# 初始化参数* exe.run(fluid.default\_startup\_program())  [inference\_program, feed\_target\_names, fetch\_targets] = \  fluid.io.load\_inference\_model(dirname=path, executor=exe)  *# 获取训练和测试程序* test\_program = fluid.default\_main\_program().clone(for\_test=**True**)  \_,tensor\_img = read\_image(**""**) *#预测图片路径* result = exe.run(test\_program, feed={**'image'**: tensor\_img}, fetch\_list=[model])  lab = np.argsort(result)[0][0][-1]  print(lab) |

接下来，我们将**“一次性推断”**转换为**“在线推理”**。计划是将这些代码包装在Flask应用程序中。Flask是一个非常轻量级的Python Web框架，它允许您用最少的工作来创建一个HTTP API服务器。根据server.py代码，我们可以实现“在线推理”。

|  |
| --- |
| *#coding:utf-8* **from** flask **import** request, Flask **import** base64 **import** cv2 **import** numpy **as** np **from** paddle **import** fluid **import** model1 **import** test app = Flask(\_\_name\_\_) @app.route(**"/"**, methods=[**'POST'**,**'GET'**]) **def** classify():  *#将接受的base64代码转化成图片，保存在本地，命名为01.png* img = base64.b64decode(str(request.form[**'image'**]))  image\_data = np.fromstring(img, np.uint8)  image\_data = cv2.imdecode(image\_data, cv2.IMREAD\_COLOR)  cv2.imwrite(**'01.png'**, image\_data)  \_,tensor\_img=test.read\_image(**"01.png"**)  *# 定义输入层* image = fluid.layers.data(name=**'image'**, shape=[3, 224, 224], dtype=**'float32'**)  mod = model1.ResNet50()  model = mod.net(image, 4)  *# 模型位置* path = **"best\_model"** *# 定义执行器* place = fluid.CUDAPlace(0)  exe = fluid.Executor(place)  *# 初始化参数* exe.run(fluid.default\_startup\_program())  [inference\_program, feed\_target\_names, fetch\_targets] = fluid.io.load\_inference\_model(dirname=path, executor=exe)  *# 获取训练和测试程序* test\_program = fluid.default\_main\_program().clone(for\_test=**True**)  *# 初始化推测列表* result = exe.run(test\_program, feed={**'image'**: tensor\_img}, fetch\_list=[model])  lab = np.argsort(result)[0][0][-1]  print(lab)  lab = str(lab)  **return** lab *#将预测结果返回给客户端* **if** \_\_name\_\_ == **"\_\_main\_\_"**:  app.run(**""**, port=) |

其实从本质上来说，无论是图片还是文字或者字符，都是数据，即最原始的一串0和1组成的二进制数据，API接收或者返回图片，本质上也就是接收或返回一段数据流。

向服务器发送一张图片的方式，我们在下一个部分介绍。从服务器接收图片，我的做法是服务器端将通过base64将字节转化为图片。

将转化的图片保存到本地后，接下来进行推理的过程，与在本地进行“一次性推推断”是一致的。用Flask实现正是我们所需要的——Flask与PaddlePaddle是完全同步的，他们将按照接收到的顺序一次处理一个请求，并且可以将处理的结果返回给客户端。

在最后一行的方法***app.run("", port=)***中，我们需要填写用花生壳5实现内网穿透的**本地IP地址**及**端口号**，完成路由注册。

1. 客户端上传/接收实现：

向服务器发送图片，具体的实现步骤是：编写Python脚本，使用OpenCV读取摄像头，设置延迟3秒后拍一张照片，并将照片保存在本地；接着，以二进制的方式打开，将其转换为base64后添加的List型变量image中，再post给服务器。其中，Line 30中定义的方法***requests.post(url,data=res)***，第一个参数需要填写一个和服务器端相同的IP地址，由于我们利用花生壳5实现内网穿透，可以脱离局域网的限制，可以利用公网IP与服务器实现通信，因此，这个参数url可以填写**花生壳5提供的域名及端口号**。

|  |
| --- |
| **from** time **import** sleep **import** RPi.GPIO **as** GPIO **import** base64 **import** requests **import** cv2 GPIO.setmode(GPIO.BCM) GPIO.setwarnings(**False**) cap = cv2.VideoCapture(0) cap.set(3,640) cap.set(4,480) **def** setServoAngle(servo, angle):  pwm = GPIO.PWM(servo, 50)  pwm.start(0)  *# dutyCycle = angle / 18 + 3*  pwm.ChangeDutyCycle(angle)  sleep(0.3)  pwm.stop() **while**(1):  ret,frame = cap.read()  sleep(3)  cv2.imwrite(**"01.jpg"**,frame)  img = cv2.imread(**"01.jpg"**)  cap.release()  **break with** open(**"01.jpg"**,**"rb"**) **as** f:  img = base64.b64encode(f.read()).decode() image = [] image.append(img) res = {**"image"**:image} url = **""** result = requests.post(url,data=res) label = str(result.content,**"utf-8"**) **if** label == **'0'**:  print(**"plastic"**)  servo = int(27)  GPIO.setup(servo, GPIO.OUT)  setServoAngle(servo, int(8))  sleep(2)  setServoAngle(servo, int(6)) **elif** label == **'1'**:  print(**"metal"**)  servo = int(22)  GPIO.setup(servo, GPIO.OUT)  setServoAngle(servo, int(8))  sleep(2)  setServoAngle(servo, int(6)) **elif** label == **'2'**:  print(**"glass"**)  servo = int(17)  GPIO.setup(servo, GPIO.OUT)  setServoAngle(servo, int(8))  sleep(2)  setServoAngle(servo, int(6)) **elif** label == **'3'**:  print(**"paper"**)  servo = int(4)  GPIO.setup(servo, GPIO.OUT)  setServoAngle(servo, int(8))  sleep(2)  setServoAngle(servo, int(6)) **else**:  **pass** |

经过服务器的处理，会将推理生成的Label值以byte型变量传回客户端，具体可参考Line31，32代码实现的过程，利用str()强制类型转换成str型的label后，根据判断调用树莓派GPIO的指定端口，实现舵机的运作。