



基于深度学习银行卡号 识别系统

软件开发文档



作 者 蔡益武、朱泓玮、宁鲲鹏

指导教师 梁栋

参赛学校 南京航空航天大学

目录

一、任务概述..... 2

 1.1 目标..... 2

 1.2 假定和约束..... 2

二、需求规定..... 2

 2.1 对功能的规定 2

 2.2 对性能的规定 4

 2.3 输入输出要求 4

 2.4 故障处理要求..... 4

三、系统设计..... 5

 3.1 系统结构模型 5

 3.2 系统行为模型 6

四、系统实现..... 7

 4.1 技术栈介绍 7

 4.2 环境搭建..... 7

 4.3 实现思路..... 10

 4.4 具体实现..... 12

 4.5 实验描述 22

 4.6 结果展示..... 26

 4.7 未来改进方向..... 28

一、任务概述

1.1 目标

随着人工智能、深度学习技术的发展，人工智能技术在视觉领域方面的应用日益突出，得到了广泛的关注和研究。该题要求参赛者使用基于深度学习的视觉识别技术，拓展现有的光学识别技术（OCR）来完成一个识别银行卡号的系统，此系统包括数据集处理、银行卡号定位检测、银行卡号识别三部分。

1.2 假定和约束

- 本系统需要一台高性能的服务器，至少有 CPU 四核、内存 8G、2080Ti GPU 一块、带宽 10M 及以上。
- 假设用户每次只上传一张图片且该图片含有完整的银行卡号。

二、需求规定

2.1 对功能的规定

2.1.1 确定参与者

本系统中，用户作为系统的参与者。用户使用了系统的主要功能，用户需要系统的支持以完成银行卡号的识别，从系统获取所需信息。

2.1.2 确定用例

本例中，通过分析可得到银行卡识别系统的用例图为图 1：

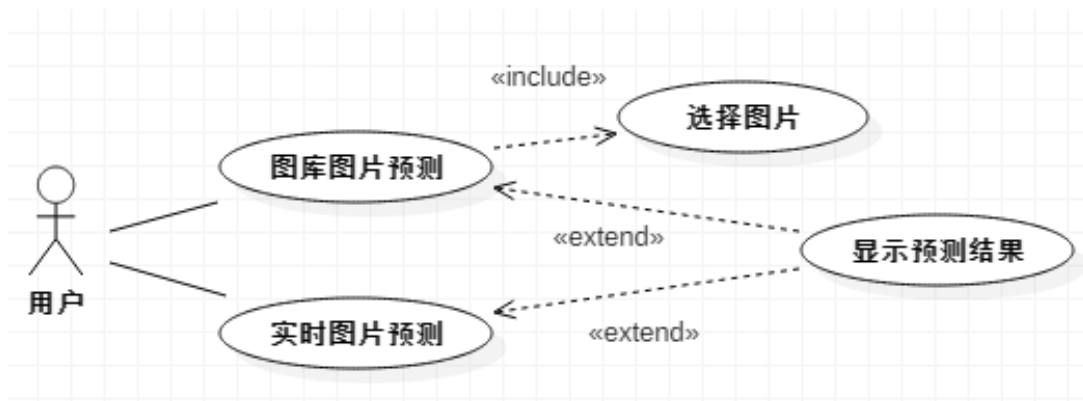


图 1 银行卡识别系统用例图

2.1.3 用例说明

以下是本系统的用例说明：

1. 图库图片预测

用例名：图库图片预测

描述：用户从图库中上传图片到服务器做预测

参与者：用户

前置条件：无

后置条件：系统将用户上传的图片放入模型做预测

2. 实时图片预测

用例名：实时图片预测

描述：用户开启摄像头实时拍摄一张照片上传到服务器做预测

参与者：用户

前置条件：无

后置条件：系统将用户上传的图片放入模型做预测

3. 选择照片

用例名：选择照片

描述：从手机图库中选择一张照片

参与者：用户

前置条件：用户选择图库图片预测功能

后置条件：无

4. 显示预测结果

用例名：显示预测结果

描述：客户端显示模型预测的卡号结果

参与者：用户

前置条件：用户上传图片且模型识别成功

后置条件：无

细节：

2.2 对性能的规定

2.2.1 精度

对银行卡号的识别结果正确率达到 90%以上。

2.2.2 时间特性要求

1. 图片上传时间不应超过 2s
2. 图片识别时间不应超过 2s

2.2.3 灵活性

1. 图片规格上：用户选择的图片分辨率要大于等于 $544 * 544$ ，且可以有轻微的倾斜、明暗变化、对焦不准、背景复杂等情况。
2. 运行环境的变化：允许运行在 Android 或 IOS 下。
3. 精度和有效时限的变化：图片处理、上传对用户体验的影响减小到最小。
4. 计划的变化或改进：较易改进。

2.3 输入输出要求

本系统的输入数据为图片，输出为用户界面的系统信息框。

2.4 故障处理要求

1. 事务内部处理：系统出现添加一些不合法的数据，如上传视频，需要有及时的系统提示。
2. 系统硬件处理：该故障属于不可抗力，如服务器奔溃，网络连接不通等，

需要系统管理员去处理。

三、系统设计

3.1 系统结构模型

3.1.1 对象表

ID	可能的类	英文类名	简单描述
1	银行卡图片	BardCard	包含图片分辨率
2	卡号区	DetectArea	包含卡号区的左上角横纵坐标以及区域的宽高
3	数字区	Number	包含该数字区的左上角横纵坐标、区域的宽高以及该数字区的值
4	标记结点	Node	包含每一个标签以及其左上角、右下角的横纵作标
5	数据增强	DataAugment	包含数据扩展
6	数据集制作	DataMake	包含数据读取以及训练、测试集的生成
7	网络模型	YOLO	包含网络定义的各式操作
8	网络训练	Train	包含网络训练的调用方法以及参数配比
9	卡号定位与预测	Predict	包含银行卡号的定位以及识别

表 1 初始对象表

3.1.2 类图

通过分析，银行卡识别系统的类图如图 2 所示。

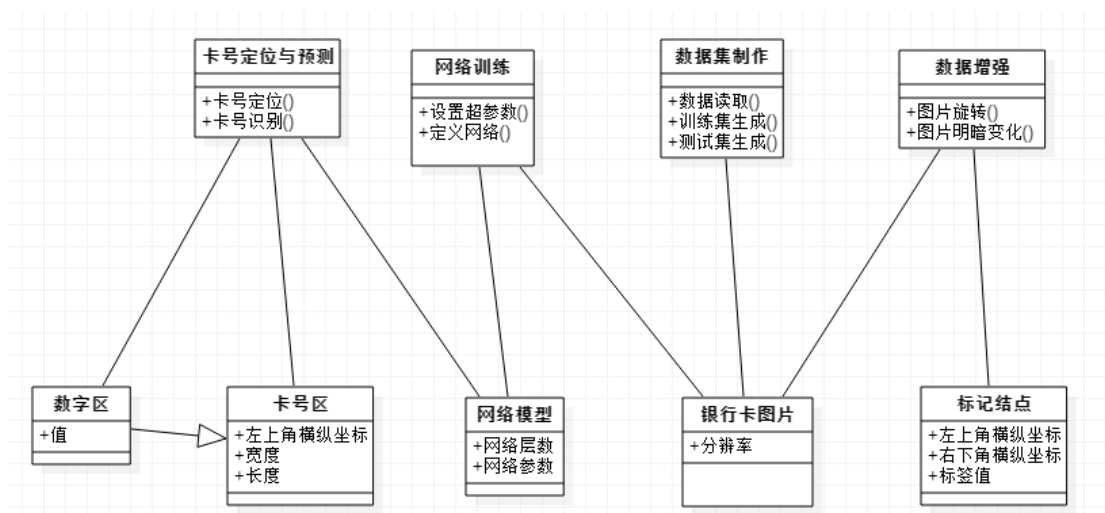


图 2 银行卡识别系统类图

3.2 系统行为模型

3.2.1 交互模型

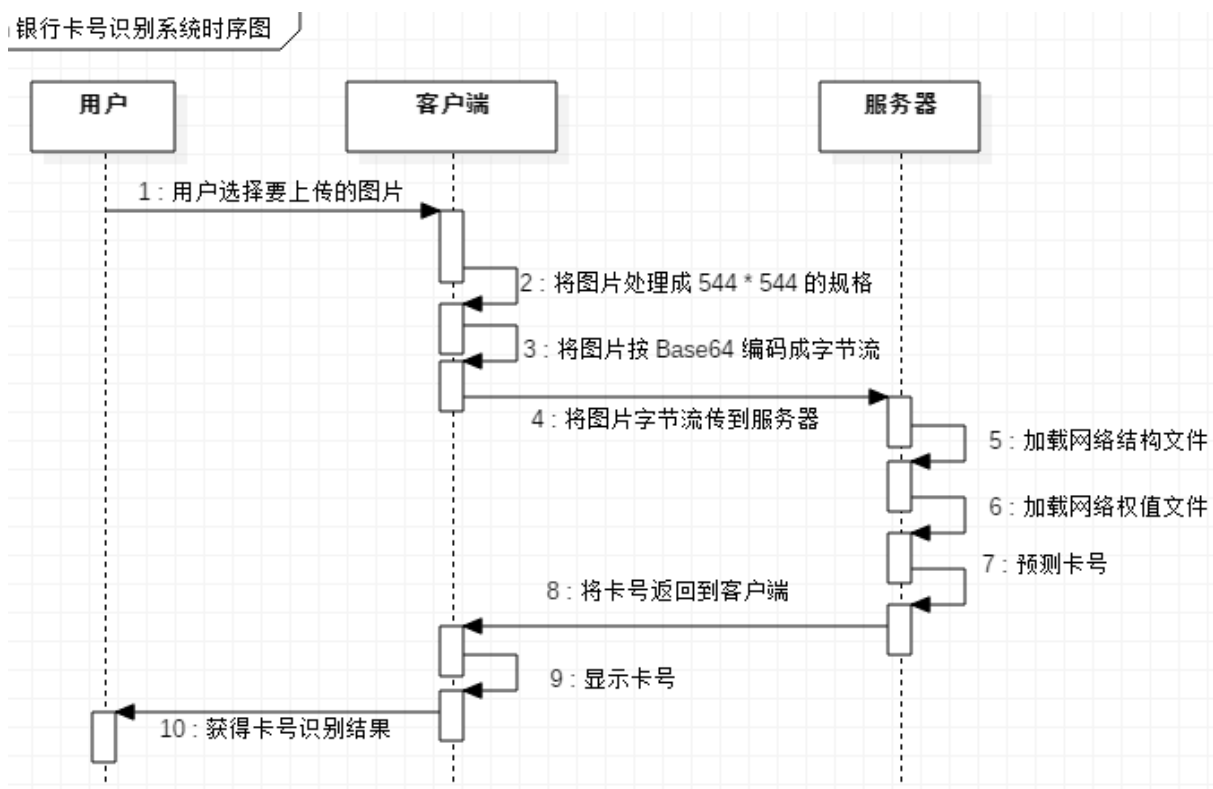


图 3 银行卡识别系统时序图

3.2.2 状态模型



图 4 银行卡识别系统状态图

四、系统实现

4.1 技术栈介绍

1. 使用 LabelImg 标记工具来标记数据。
2. 以 Python 作为主要的编程语言，darknet 作为深度学习框架，通过 Keras 深度学习库，以 Tensorflow 作为后端运行，构建 YOLOv3 网络来完成卡号的定位与识别任务。
3. 通过 Python Flask 来搭建服务器。
4. 通过 HBuilder 来开发跨平台图形界面应用程序。

4.2 环境搭建

4.2.1 操作环境要求

1. 系统要求：Ubuntu Linux x64 v16.04
2. 硬件要求：Nvidia GPU GeForce GTX 960
3. 软件要求：PyCharm 2018.1.4， Anaconda3-5.1.0
4. 编程语言：Python 3.6

4.2.2 安装配置

4.2.2.1 安装 NVIDIA 驱动

装有 Ubuntu 操作系统的电脑点击系统设置，点击软件和更新，然后附加驱动，安装 NVIDIA 驱动，安装之后重启电脑

4.2.2.2 安装 CUDA 驱动

1. 在 nvidia 官网下载 CUDA, 下载之后执行一下命令（这边是 CUDA 8.0）：

```
sudo chmod 777 cuda_8.0.61_375.26_linux.run  
sudo ./cuda_8.0.61_375.26_linux.run
```

2. 环境变量配置

用如下命令打开 ~/.bashrc 文件：

```
sudo gedit ~/.bashrc
```

将以下内容写入到 ~/.bashrc ：

```
export PATH = /usr/local/cuda-8.0/bin${PATH:+:${PATH}}  
export LD_LIBRARY_PATH = /usr/local/cuda-8.0/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

用 sudo 权限打开 /etc/profile 文件在结尾添加如下：

```
PATH=/usr/local/cuda/bin:$PATH  
export PATH
```

保存后，执行下列命令，使环境变量立即生效

```
source /etc/profile
```

添加 lib 库路径

```
sudo gedit /etc/ld.so.conf.d/cuda.conf
```

在文中加入以下内容

```
/usr/local/cuda/lib64
```

执行如下命令使其生效

```
sudo ldconfig
```

3. 试 CUDA 的 samples

```
cd  
/usr/local/cuda-8.0/samples/1_Uutilities/deviceQuery  
make sudo ./deviceQuery
```

如果显示一些 GPU 信息，则证明安装成功。

4.2.2.3 安装 CuDNN

1. 在 nvidia 官网下载 cuDNN，然后解压：

```
tar -zxvf ./cudnn-8.0-linux-x64-v5.0.tgz
```

2. 进入 cuDNN 5.0 解压之后的 include 目录，在命令行进行如下操作：

```
cd cuda/include  
sudo cp cudnn.h /usr/local/cuda/include #复制头文件
```

3. 进入 lib64 目录下的动态文件进行复制和链接：

```
cd ..  
cd lib64  
sudo cp lib* /usr/local/cuda/lib64/ #复制动态链接库  
cd /usr/local/cuda/lib64/  
sudo chmod +r libcudnn.so.5.0.5  
sudo ln -sf libcudnn.so.5.0.5 libcudnn.so.5  
sudo ln -sf libcudnn.so.5 libcudnn.so  
sudo ldconfig
```

4.2.2.4 配置 Python 依赖包

通过 Anaconda 导入 Python 包可以避免一个一个去官网找源的麻烦。

以下列出本项目所依赖的包：

- tensorflow-gpu 1.11.0
- tensorflow 1.11.0
- keras 2.2.4
- numpy 1.14.2
- scikit-image 0.13.1
- pillow 5.0.0
- lxml 4.1.1
- matplotlib 2.1.2
- pydot 1.4.1

4.2.2.5 安装 HBuilder

HBuilder 是数字天堂推出的一款支持 HTML5 的 Web 开发 IDE。我们通过它来将网页相关部署到各种类型的客户端。在 HBuilder 官网下载 9.0.2.windows 的版本，根据引导安装即可，不需要手动配置环境变量等操作。

4.3 实现思路

4.3.1 系统实现流程图

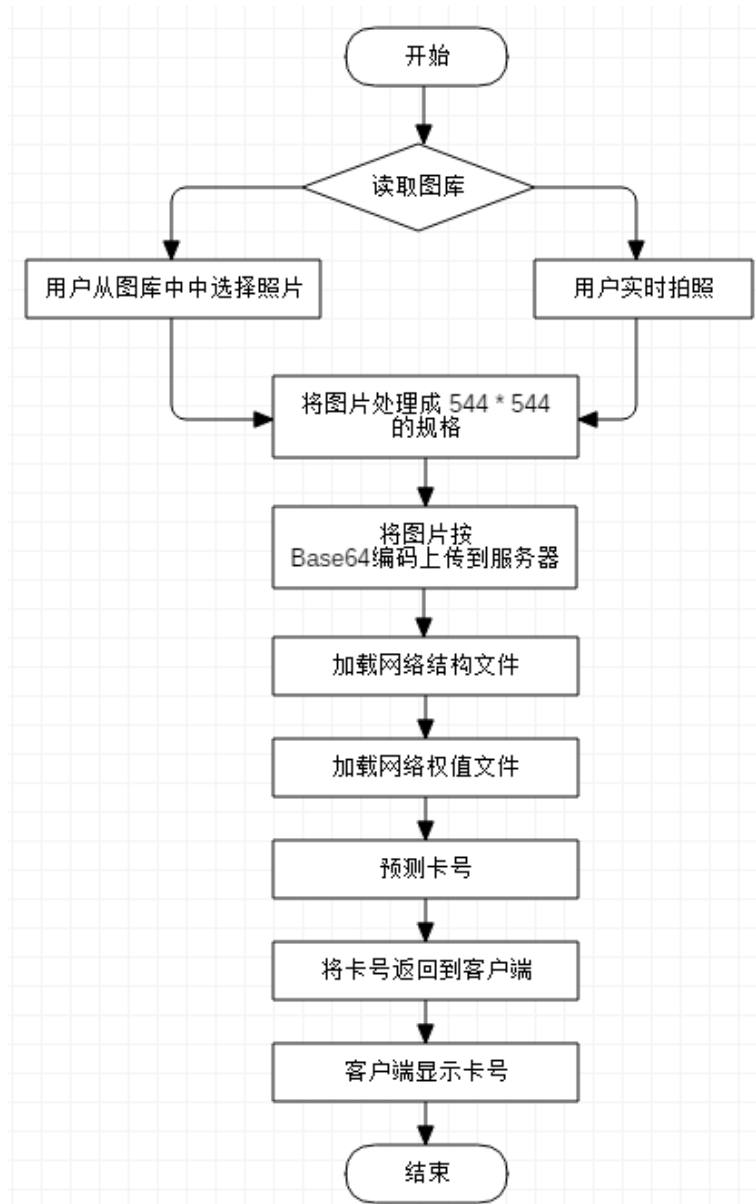


图 5 银行卡识别系统流程图

4.3.2 算法介绍

在本次任务中，我们选用 YOLOv3 作为参考模型。YOLOv3 的网络识别效率远高于其它的目标检测框架，下图是在 TITAN V 上的对比测试结果，可以看出

YOLOv3 兼具速度和精度。

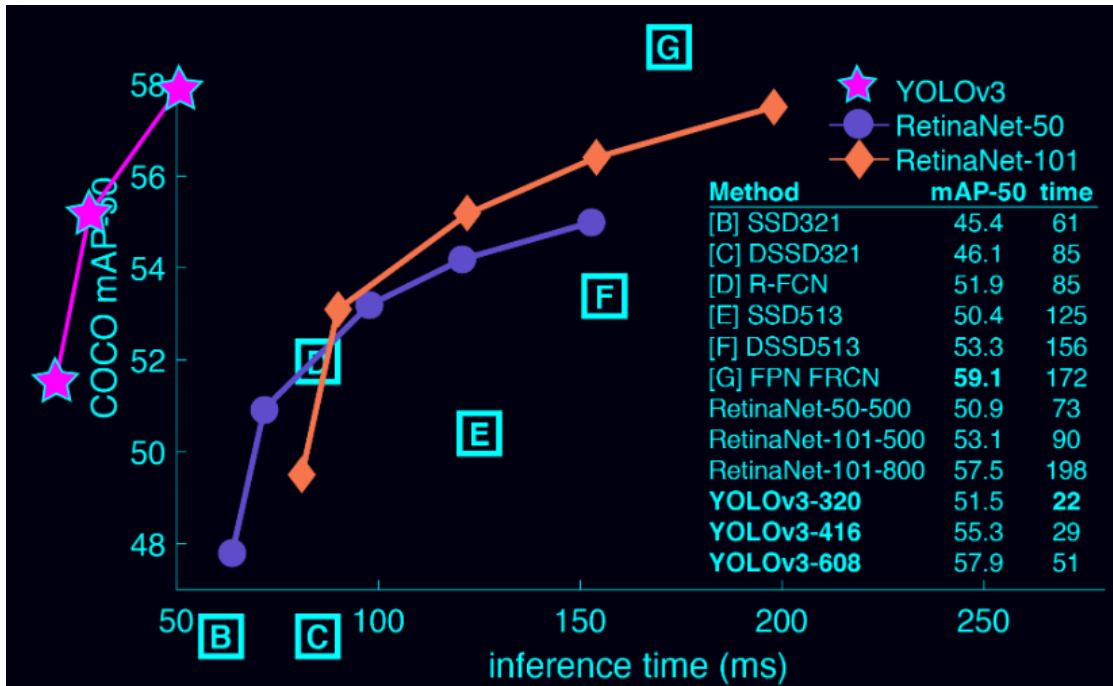


图6 YOLOv3 与其他网络的识别效果对比图（图片源于 YOLO 官网）

YOLOv3 网络识别的准确率与 YOLOv1 ,YOLOv2 相比，都有着较大的提升，尤其在物体定位与识别方面的准确率大大提高。在此基础上，我们将这两个任务放在一起处理。即，在一次预测中，同时给出区域位置与识别的数字串。同时，我们对 YOLOv3 的网络结构进行了如下改进：

1. 考虑到输入图片分辨率在 700×484 到 4032×3024 之间不等，图片过大造成的计算量显著上升，以及避免将图片压缩过小造成的目标物体过小、检测率下降的情况发生，我们设定模型的输入图片的大小为 544×544 。
2. 由于 YOLOv3 中的 NMS（非极大值抑制）只能抑制类内多预测框，即同一种标签有多个预测框，而无法处理类间的多预测框，所以在预测阶段若出现类间多预测框的情况时，通过比较超过指定阈值的 IOU 对应的预测类的置信度来选取最优预测框。

以及在此技术上，我们做了一些创新性工作：

1. 基于先验知识的卡号自动纠错。

对于模型的识别输出结果，将其前几位根据实际生活中的各大银行其固定的银行卡号构建的字典做相似度匹配。例如，对于前六位识别成“620050”的结果，通过查询字典发现“62005”开头的银行卡号，只有“620058”的情况，所以将其纠正成正确结果。

2. 空格识别。

通过 YOLOv3 模型得到的识别结果中，我们可以得到每个数字预测框的几

何位置，计算得到每个边框的间距，将间距结果经过 softmax 处理。在此基础上，设定一个阈值，若两个相邻的数字框之间存在大于此阈值的间距，则认为其中存在空格。之后，通过实验优化，择取最佳阈值。

4.4 具体实现

4.4.1 数据增强

自收集的训练集，其中包括 350 张真实拍摄的银行卡照片，通过标记工具 LabelImg 对其做标记可以得到每个图片中对应的标记数据，如图 7：



图 7 标记工具 labelImg 的使用

得到的 XML 文件部分截图如图 8 所示：

```

<annotation verified="yes">
  <folder>cyw-bank-card-set</folder>
  <filename>cyw0002.jpg</filename>
  <path>C:\Users\cyw35\Desktop\cyw-bank-card-set\cyw0002.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>278</width>
    <height>181</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>6</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>28</xmin>
      <ymin>89</ymin>
      <xmax>39</xmax>
      <ymax>107</ymax>
    </bndbox>
  </object>
  <object>
    <name>2</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>39</xmin>
      <ymin>90</ymin>
      <xmax>49</xmax>
      <ymax>107</ymax>
    </bndbox>
  </object>
</annotation>

```

类别

左上角、右下角的纵横坐标

图 8 标记数据格式

然后通过背景替换、高亮叠加、模糊度调节、轻微旋转、随机调整图片亮暗程度等手段将每一张原始图片拓展为 40 张，同时，为避免重复标记，相应调整了标签的标记值，代码如下：

```

def label_rotate(xmin, ymin, xmax, ymax, angle, image_x, image_y):
    """
    标签标记值的随着图片的转动而改变
    :param xmin: 图片左上角的横坐标
    :param ymin: 图片左上角的纵坐标
    :param xmax: 图片右下角的横坐标
    :param ymax: 图片右下角的纵坐标
    :param angle: 旋转角度
    :param image_x: 图片的宽
    :param image_y: 图片的长
    :return: 旋转后的(xmin, ymin, xmax, ymax) 元组
    """

    # 角度转换成弧度
    angle = -angle * math.pi / 180.0
    center_x = image_x / 2.0
    center_y = image_y / 2.0
    x = [xmin, xmax, xmin, xmax]

```

```

y = [ymin, ymin, ymax, ymax]
# 旋转公式
#  $x = (x1 - x2)\cos \theta - (y1 - y2)\sin \theta + x2$ 
#  $y = (x1 - x2)\sin \theta + (y1 - y2)\cos \theta + y2$ 
# 逆时针旋转的情况
if angle < 0:
    nxmin = (x[0] - center_x) * math.cos(angle) - (y[0] - center_y) *
math.sin(angle) + center_x
    nymin = (x[1] - center_x) * math.sin(angle) + (y[1] - center_y) *
math.cos(angle) + center_y
    nymin = (x[2] - center_x) * math.sin(angle) + (y[2] - center_y) *
math.cos(angle) + center_y
    nxmax = (x[3] - center_x) * math.cos(angle) - (y[3] - center_y) *
math.sin(angle) + center_x
    # 顺时针旋转的情况
else:
    nymin = (x[0] - center_x) * math.sin(angle) + (y[0] - center_y) *
math.cos(angle) + center_y
    nxmax = (x[1] - center_x) * math.cos(angle) - (y[1] - center_y) *
math.sin(angle) + center_x
    nxmin = (x[2] - center_x) * math.cos(angle) - (y[2] - center_y) *
math.sin(angle) + center_x
    nymin = (x[3] - center_x) * math.sin(angle) + (y[3] - center_y) *
math.cos(angle) + center_y

    return (str(int(nxmin)), str(int(nymin)), str(int(nxmax)),
str(int(nymax)))

```

经数据增强生成的图片如下：



图 9 由原始图像生成的拓展图像（中间为原始图像）

至此，我们达到数据增强的目的，为之后的图像识别训练提供充足的数据样本，并且该模块程序能够继续处理新加入的数据样本。

4.4.2 训练过程

4.4.2.1 预处理

1. 图片预处理

将图片预处理成 544*544，之后将[height, weight, 3]张量除于 255，做数据归一化处理。

```
def letterbox_image(image, size):
    '''
    调整 image 的图片大小为 size
    :param image: 待调整的图片
    :param size: (w, h) 元组
    :return:
    '''
    iw, ih = image.size
    w, h = size
    scale = min(w / iw, h / ih)
    nw = int(iw * scale)
    nh = int(ih * scale)
    image = image.resize((nw, nh), Image.BICUBIC)
    new_image = Image.new('RGB', size, (128, 128, 128))
```



```
new_image.paste(image, ((w - nw) // 2, (h - nh) // 2))
padding = ((w - nw) // 2, (h - nh) // 2)
return new_image, scale, padding
```

2. 标签预处理

将标记数据转换成 YOLO 网络需要的数据格式，即[N, grid_shapes[1][0], grid_shapes[1][1], 3, 16]。

- a) N: 标签个数。
- b) Grid_shapes[1][0], Grid_shapes[1][1]: 对应于 YOLOv3 中的 3 个子网络的输入长、宽，值分别为 $\text{input_shape} / 32$, $\text{input_shape} / 16$, $\text{input_shape} / 8$ 。
- c) 3: 表示 9 个由 K-means 算法产生的聚类中心按大小分给 3 个子网络。
- d) 16: 表示 $x_{\min} + y_{\min} + x_{\max} + y_{\max} + \text{num_classes}$ ，因为本次任务中标签为 10 个数字加上 1 个卡号区域，所以最终为 $5 + 11 = 16$ 。

代码如下：

```
def preprocess_true_boxes(true_boxes, input_shape, anchors,
                           num_classes):
    """
    将标记数据转换成 YOLO 网络需要的数据格式
    :param true_boxes: shape 为(m, T, 5)
    :param input_shape: [height, weight]
    :param anchors: 聚类中心集合
    :param num_classes: 标签个数
    :return: list d
    """

    assert (true_boxes[..., 4] < num_classes).all(), 'class id must be less than num_classes'

    num_layers = len(anchors) // 3 # default setting
    anchor_mask = [[6, 7, 8], [3, 4, 5], [0, 1, 2]] if num_layers == 3 else [[3, 4, 5], [1, 2, 3]]

    true_boxes = np.array(true_boxes, dtype='float32')
    input_shape = np.array(input_shape, dtype='int32')
    boxes_xy = (true_boxes[..., 0:2] + true_boxes[..., 2:4]) // 2
    boxes_wh = true_boxes[..., 2:4] - true_boxes[..., 0:2]
    true_boxes[..., 0:2] = boxes_xy / input_shape[:-1]
    true_boxes[..., 2:4] = boxes_wh / input_shape[:-1]
```

```

m = true_boxes.shape[0]
grid_shapes = [input_shape // {0: 32, 1: 16, 2: 8}[l] for l in
range(num_layers)]
y_true = [np.zeros((m, grid_shapes[l][0], grid_shapes[l][1],
len(anchor_mask[l]), 5 + num_classes),
dtype='float32') for l in range(num_layers)]

anchors = np.expand_dims(anchors, 0)
anchor_maxes = anchors / 2.
anchor_mins = -anchor_maxes
valid_mask = boxes_wh[..., 0] > 0
# 每个图片都需要单独处理
for b in range(m):
    wh = boxes_wh[b, valid_mask[b]]
    if len(wh) == 0: continue
    wh = np.expand_dims(wh, -2)
    box_maxes = wh / 2.
    box_mins = -box_maxes

    intersect_mins = np.maximum(box_mins, anchor_mins)
    intersect_maxes = np.minimum(box_maxes, anchor_maxes)
    intersect_wh = np.maximum(intersect_maxes - intersect_mins, 0.)
    intersect_area = intersect_wh[..., 0] * intersect_wh[..., 1]
    box_area = wh[..., 0] * wh[..., 1]
    anchor_area = anchors[..., 0] * anchors[..., 1]
    iou = intersect_area / (box_area + anchor_area - intersect_area)

    # 9 个设定的 ANCHOR 去框定每个输入的 BOX
    best_anchor = np.argmax(iou, axis=-1)

    for t, n in enumerate(best_anchor):
        for l in range(num_layers):
            if n in anchor_mask[l]:
                i = np.floor(true_boxes[b, t, 0] *
grid_shapes[l][1]).astype('int32')
                j = np.floor(true_boxes[b, t, 1] *
grid_shapes[l][0]).astype('int32')
                k = anchor_mask[l].index(n)
                c = true_boxes[b, t, 4].astype('int32')
                # 设定数据 将 T 个 BOX 的标的数据统一放置到 3*B*W*H*3
的维度上
                y_true[l][b, j, i, k, 0:4] = true_boxes[b, t, 0:4]

```

```

y_true[l][b, j, i, k, 4] = 1
y_true[l][b, j, i, k, 5 + c] = 1

return y_true

```

4.4.1.2 训练过程

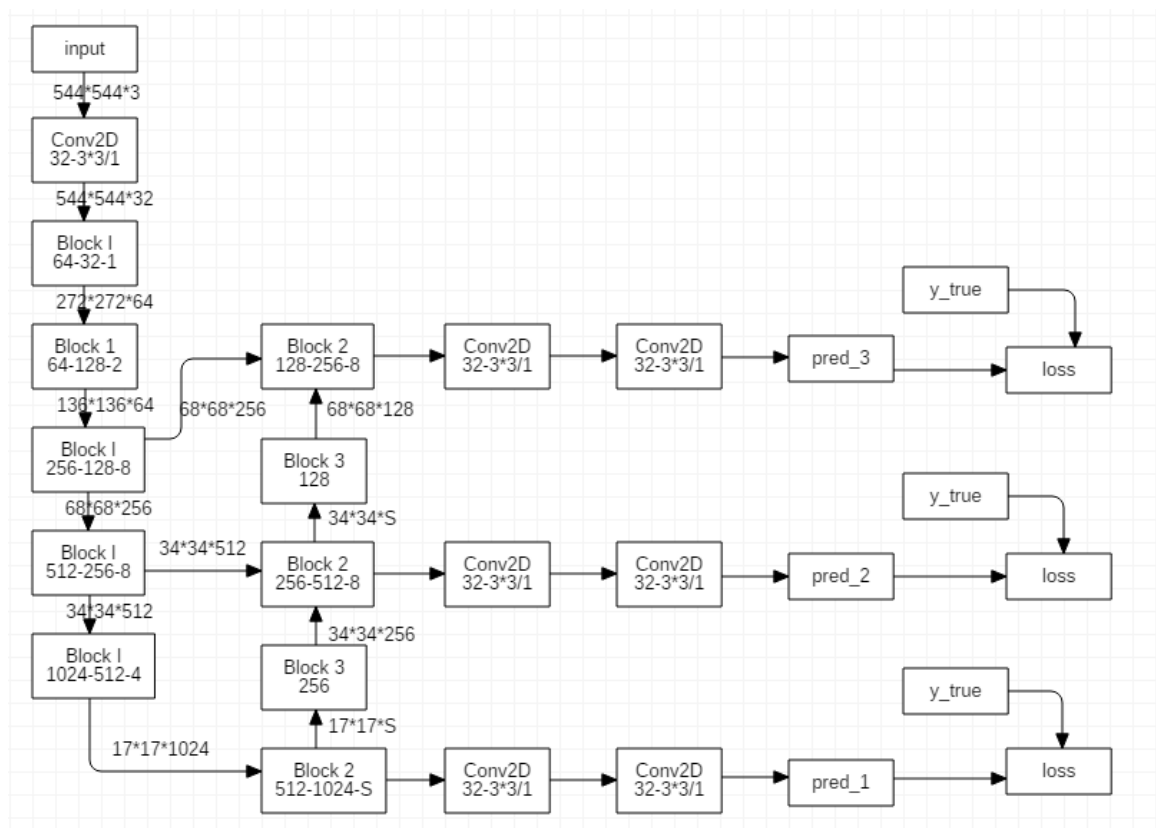


图 10 YOLOv3 网络训练示意图（部分）

完整网络模型我们以长图片形式附在「其他材料中」。

4.4.1.3 Loss 计算

$\text{Loss} = \text{xy_loss} + \text{wh_loss} + \text{confidence_loss} + \text{class_loss}$

- a) xy_loss: 预测框左上角与实际框的误差
- b) wh_loss: 预测框的长宽与实际框长宽的误差
- c) confidence_loss: 预测框与实际框的误差
- d) class_loss: 预测值与实际值的误差

主要代码如下：

```

_, ignore_mask = K.control_flow_ops.while_loop(lambda b, *args: b < m,
loop_body, [0, ignore_mask])

```

```

ignore_mask = ignore_mask.stack()
ignore_mask = K.expand_dims(ignore_mask, -1)

xy_loss = object_mask * box_loss_scale * K.binary_crossentropy(raw_true_xy,
raw_pred[..., 0:2],

from_logits=True)
wh_loss = object_mask * box_loss_scale * 0.5 * K.square(raw_true_wh -
raw_pred[..., 2:4])
confidence_loss = object_mask * K.binary_crossentropy(object_mask,
raw_pred[..., 4:5], from_logits=True) + \
                (1 - object_mask) * K.binary_crossentropy(object_mask,
raw_pred[..., 4:5],

from_logits=True) * ignore_mask
class_loss = object_mask * K.binary_crossentropy(true_class_probs,
raw_pred[..., 5:], from_logits=True)

xy_loss = K.sum(xy_loss) / mf
wh_loss = K.sum(wh_loss) / mf
confidence_loss = K.sum(confidence_loss) / mf
class_loss = K.sum(class_loss) / mf
loss += xy_loss + wh_loss + confidence_loss + class_loss

```

4.4.3 预测部分

4.4.3.1 预处理

与 4.4.2.1 部分一样。

4.4.3.2 预测过程

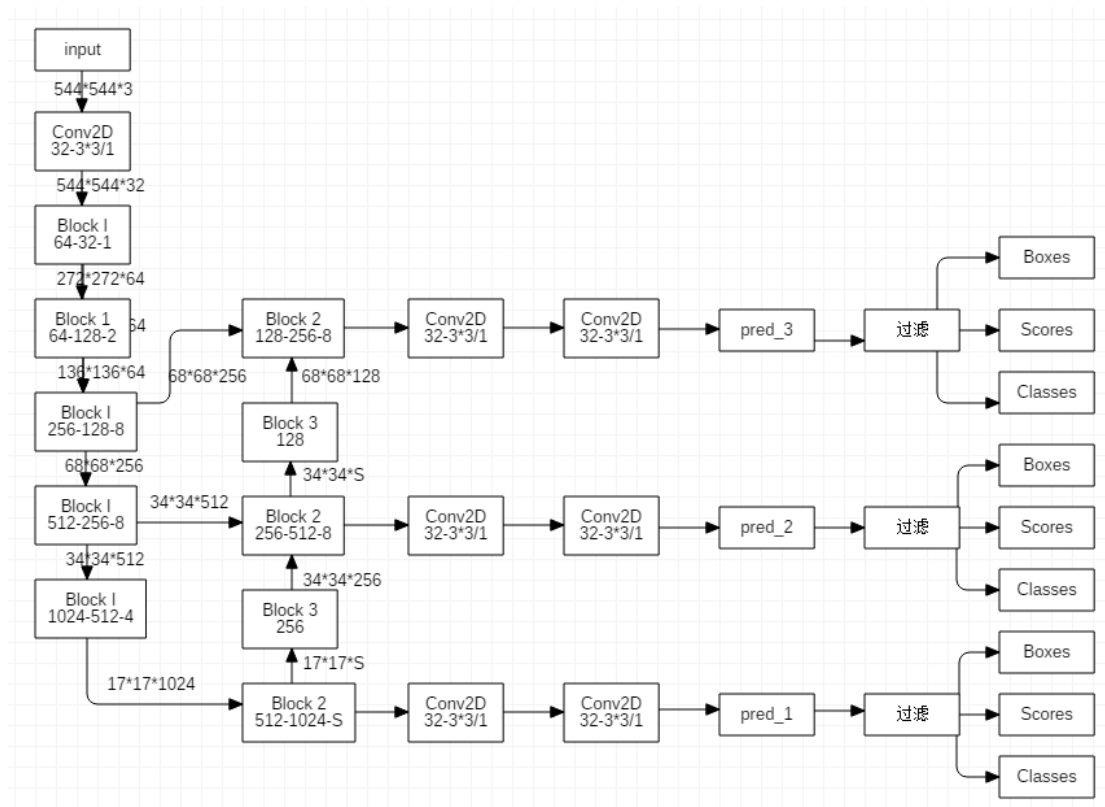


图 11 YOLOv3 网络预测示意图（部分）

完整网络模型我们以长图片形式附在「其他材料中」。

4.4.3.3 过滤

对于任意 `pred_box`:

- 进行以 0.45 为 IOU 阈值, 0.6 为置信度阈值的 NMS, 以去除类内多预测框的情况。代码如下:

```
def yolo_eval(yolo_outputs, anchors, num_classes, image_shape,
              score_threshold, iou_threshold, max_boxes=20):
    '''YOLO 模型的评价机制'''
    num_layers = len(yolo_outputs)
    anchor_mask = [[6, 7, 8], [3, 4, 5], [0, 1, 2]] if num_layers == 3
    else [[3, 4, 5], [1, 2, 3]] # default setting
    input_shape = K.shape(yolo_outputs[0])[1:3] * 32
    boxes = []
    box_scores = []
    for l in range(num_layers):
        _boxes, _box_scores = yolo_boxes_and_scores(yolo_outputs[l],
            anchors[anchor_mask[l]], num_classes, input_shape, image_shape)
```

```

        boxes.append(_boxes)
        box_scores.append(_box_scores)
    boxes = K.concatenate(boxes, axis=0)
    box_scores = K.concatenate(box_scores, axis=0)

    mask = box_scores >= score_threshold
    max_boxes_tensor = K.constant(max_boxes, dtype='int32')
    boxes_ = []
    scores_ = []
    classes_ = []
    for c in range(num_classes):
        class_boxes = tf.boolean_mask(boxes, mask[:, c])
        class_box_scores = tf.boolean_mask(box_scores[:, c], mask[:, c])
        nms_index = tf.image.non_max_suppression(
            class_boxes, class_box_scores, max_boxes_tensor,
iou_threshold=iou_threshold)
        class_boxes = K.gather(class_boxes, nms_index)
        class_box_scores = K.gather(class_box_scores, nms_index)
        classes = K.ones_like(class_box_scores, 'int32') * c
        boxes_.append(class_boxes)
        scores_.append(class_box_scores)
        classes_.append(classes)
    boxes_ = K.concatenate(boxes_, axis=0)
    scores_ = K.concatenate(scores_, axis=0)
    classes_ = K.concatenate(classes_, axis=0)

    return boxes_, scores_, classes_

```

- b) 通过比较超过以 0.45 阈值的 IOU 对应的预测类的置信度来选取最优预测框，以去掉类间多预测框的情况。代码如下：

```

def nms_inter_class(pre_rec, pre_trust, pre_class):
    """
    NMS 只能去掉类内多预测框，本函数处理类间多预测框的问题
    :param pre_rec:
    :param pre_trust:
    :param pre_class:
    :return: ['1', '2', '3']的预测数字串以及每个预测数字对应的区域、可信度
    """
    result_rec = []
    result_trust = []
    result_class = []

```

```

for i in range(len(pre_class)):
    if len(result_rec) == 0:
        result_rec.append(pre_rec[i])
        result_trust.append(pre_trust[i])
        result_class.append(pre_class[i])
    else:
        flag = True
        for j in range(len(result_rec)):
            # 比较 iou 若超过阈值, 则比较置信度
            if compute_iou(result_rec[j], pre_rec[i]) > iou:
                flag = False
                # 将置信度高的替换掉 result
                if pre_trust[i] > result_trust[j]:
                    result_rec[j] = pre_rec[i]
                    result_trust[j] = pre_trust[i]
                    result_class[j] = pre_class[i]
                break
            # 若无重复 则添加到 result
        if flag == True:
            result_rec.append(pre_rec[i])
            result_trust.append(pre_trust[i])
            result_class.append(pre_class[i])
return result_class, result_rec, result_trust

```

4.5 实验描述

4.5.1 精度调参

为了提高模型的泛化性, 在收集的实拍数据中, 不对用户使用习惯进行约束, 让不同的用户以其使用习惯为主, 拍摄其所拥有的各类银行卡。最终收集到 350 张真实图片, 对其进行手工标记, 然后经过数据增强达到 14000 张。

以下为我们的调参过程:

a) 前置条件: 迭代次数为 100 次

我们通过修改不同的 `input_shape` (输入大小), 观察在验证集上的 `loss` 来选出最优的 `input_shape`。需要注意的是, 由于 YOLOv3 网络模型中有 5 层 2×2 的卷积, 所以输入大小设为 32 (2^5) 的倍数比较合适。

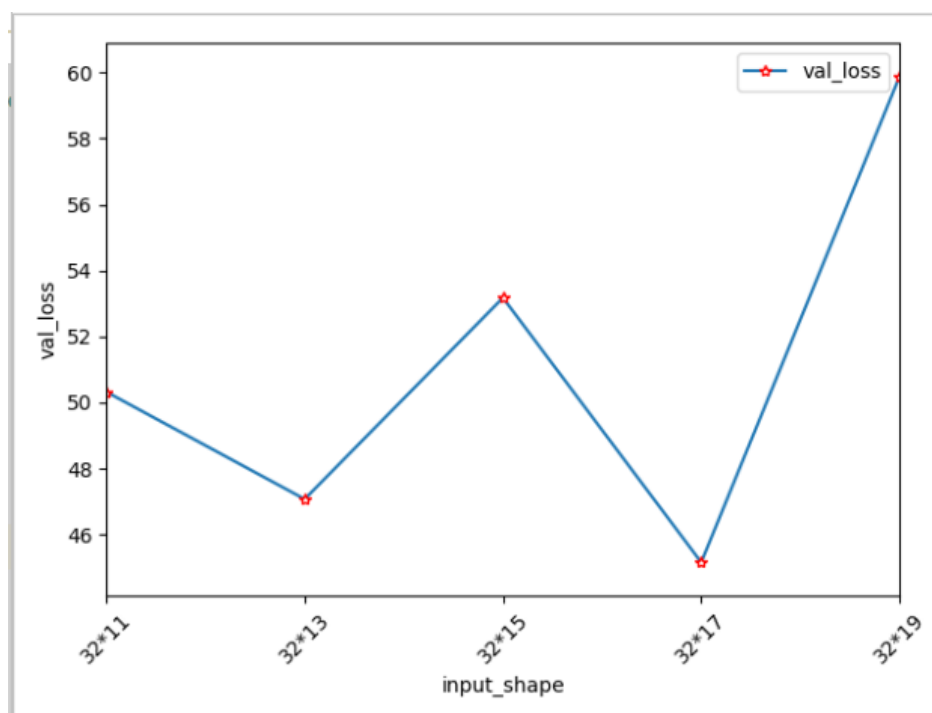


图 12 val_loss 关于 input_shape 的折线图

可以看出，当 input_shape 为【32*17, 32*17】时，val_loss 最低。

b) 前置条件：迭代次数为 100 次，input_shape 为【32*17, 32*17】

通过修改不同的置信度和 IOU 来选出最优的 IOU、置信度。

首先我们选择 IOU 保持默认值 0.45 不变，修改置信度(score_threshold)的值

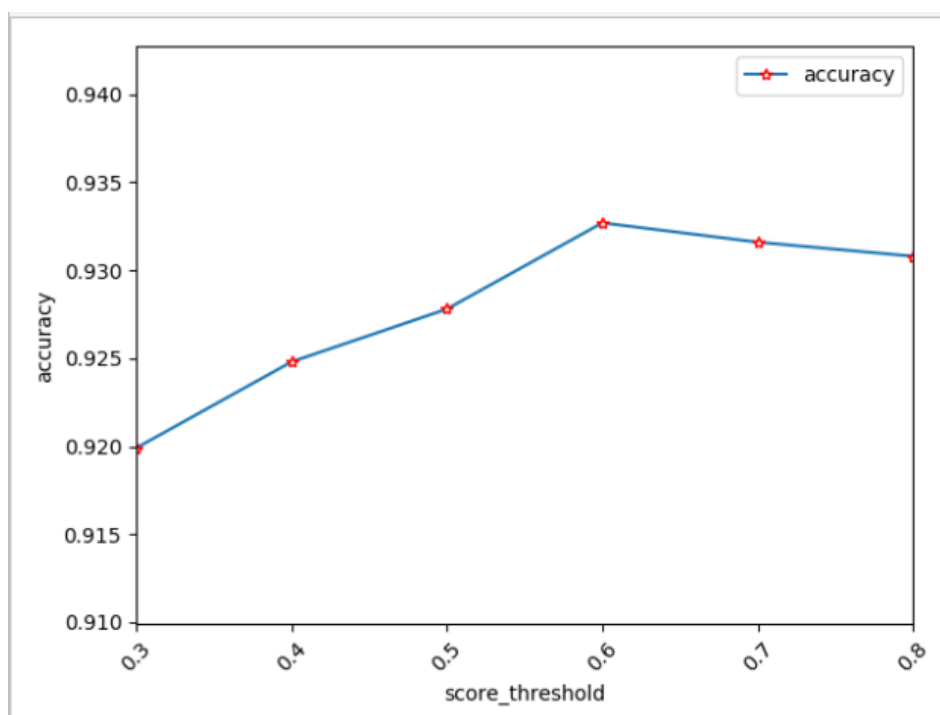


图 13 accuracy 关于 score_threshold 的折线图

可以看出，当 score_threshold 为 0.6 时，accuracy 最大。

然后我们调整 IOU，得到结果如下：

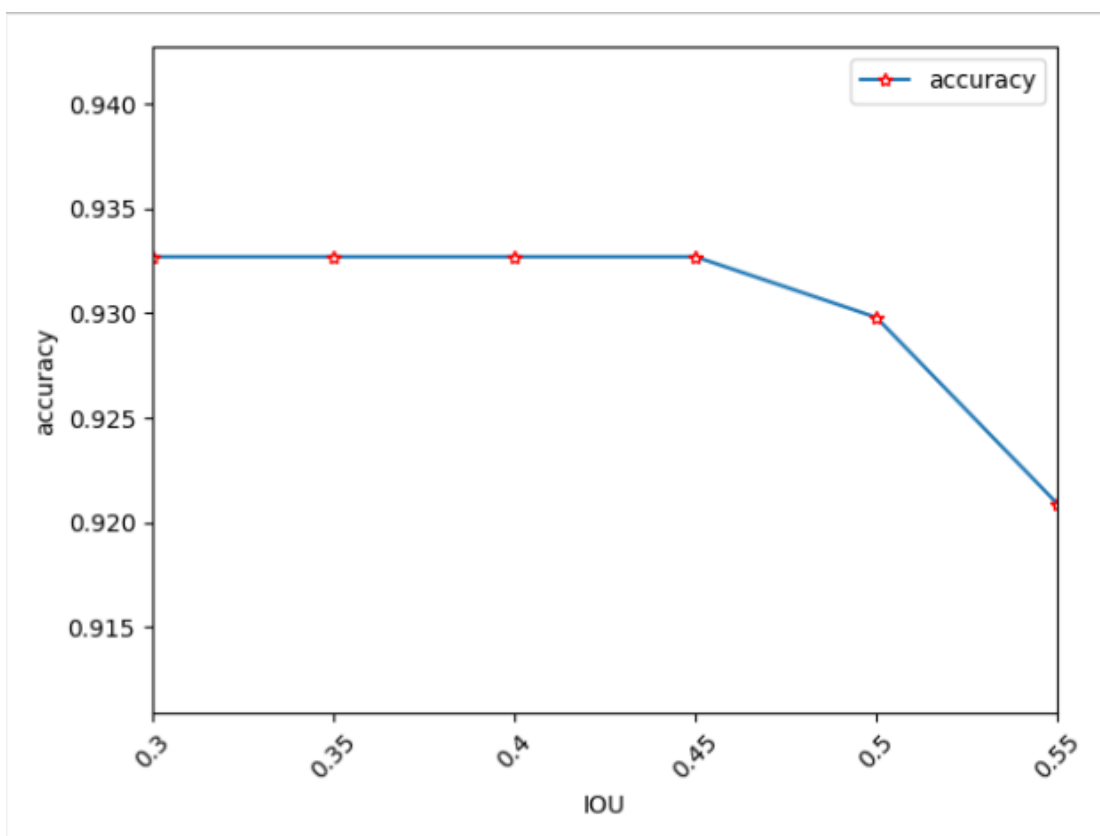


图 14 accuracy 关于 IOU 的折线图

由图可知，0.3-0.45 的结果是一样的，因为 IOU 越大说明预测框与真实框的重叠区域越大，所以在同等效果下我们选择最大的 0.45。

- c) 前置条件：input_shape 为 **【32*17, 32*17】**，IOU 为 0.45，score_threshold 为 0.6，学习率为 Adam 优化器默认的 0.001
通过修改不同迭代次数来选出最优的结果

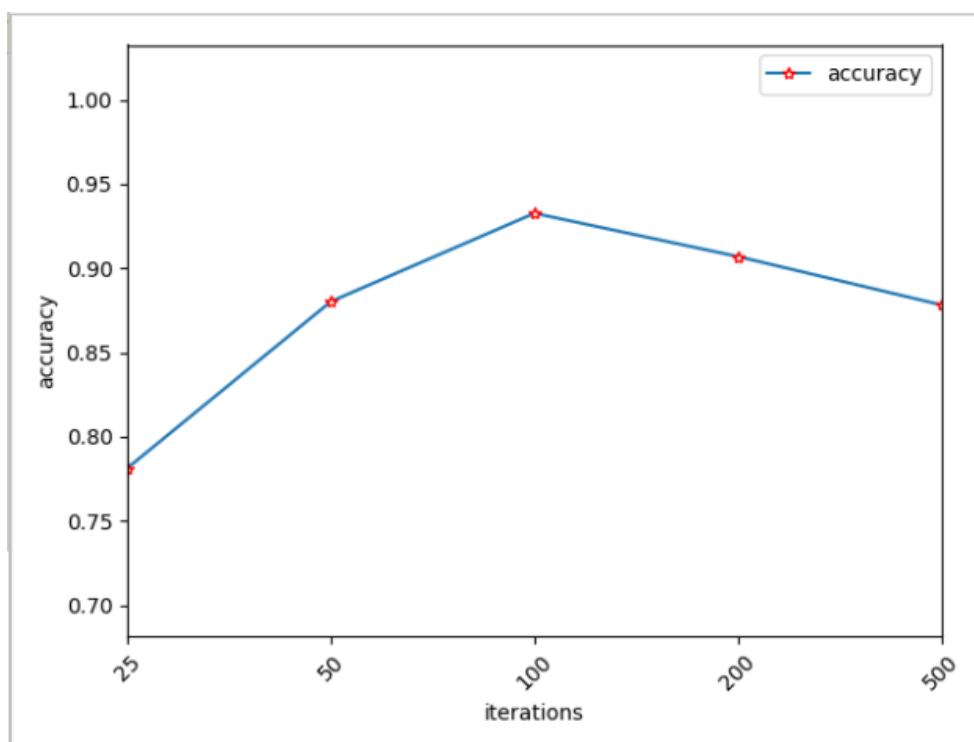


图 15 accuracy 关于迭代次数的折线图

由此可知，迭代次数为 100 时，效果最优。

至此，网络模型的参数配置为：

- Input_shape 为【32*17, 32*17】
- 迭代次数为 100
- 优化器为默认参数的优化器（学习率为 0.001）
- 预测时 IOU 为 0.45、置信度为 0.6

4.5.2 时间调参

为了提高用户体验，考虑到实时识别的重要性，我们使用了如下的几种方法大大缩短模型预测所需时间：

- a) 提前加载模型。我们将预测模型部署在服务器上，当服务器上开启相关服务时，系统将网络模型及其权值文件加载到内存中，对卡号定位预测部分做剥离，使得整体时间大幅度下降。
- b) 客户端图片处理。由于客户端传到服务器端，图像解码时间过长，在精度不丢失的前提下，对图片进行适当压缩。减少解码时间，提高速度。
- c) YOLO 网络剪枝。由于 YOLO 网络层数较多，参数数量更是多达 6 亿个，在移动平台上进行如此大规模的计算非常耗时，所以需要网络进行剪枝处理。


对每个通道添加缩放因子 γ ，让通道的输出和 γ 相乘，乘积近似为 0 的项即为对训练集不敏感的神经元，将其舍去。虽然当剪枝率较高时，可能会暂时导致一些精度损失，但通过后续的微调过程对网络进行修剪，可以弥补剪枝时丢失的精度。

4.6 结果展示

在此参数配置下，在 1000 张测试集上的准确率（一张图片全部数字都正确才算预测成功一次）为 93.27%。以下是部分在真实环境中的测试结果：

BankCard Recognition

Select




识别结果:

[6, '2', '1', '6', '7', '1', '0', '4', '7', '0', '0', '2', '4', '2', '8', '3', '3', '7', '8']

BankCard Recognition

Select



识别结果:

[6, '2', '2', '8', '4', '8', '0', '3', '3', '0', '3', '4', '6', '4', '4', '0', '5', '1', '5']

BankCard Recognition

Select



识别结果:

[6, '2', '0', '0', '5', '8', '3', '7', '0', '0', '0', '0', '5', '8', '0', '3', '0', '8']

最终的 UI 界面为:

BankCard Recognition

Select



识别结果:

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

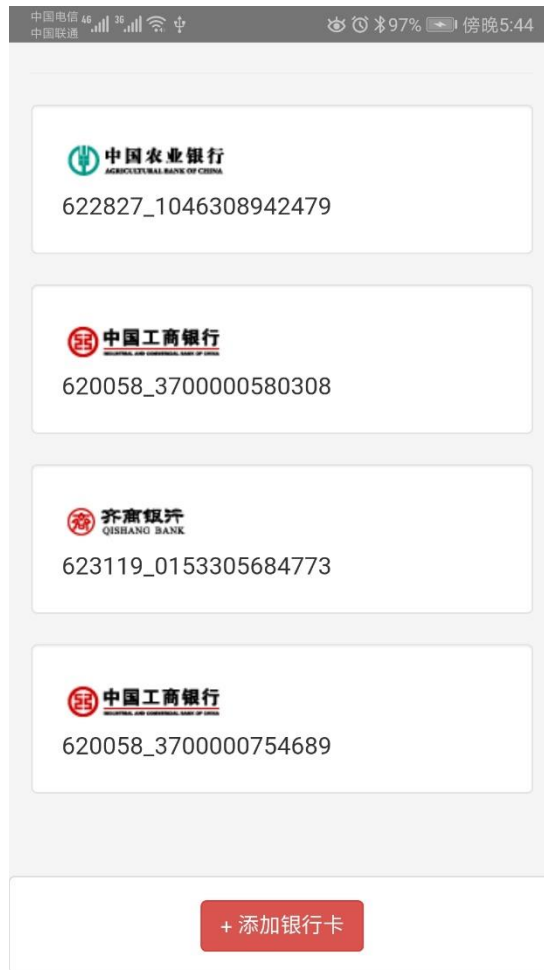


图 16 App 上的测结果展示

4.7 未来改进方向

尽管目前我们已经取得不错的成果，但如果时间充裕的话，还可以往如下几个方面改进：

1. 完善图形界面应用程序。增加数据持久化功能，构造一个完整的系统，改善 UI，给用户更为优雅的界面。
2. 支持扫描即时取号。用户无需做「按下拍照键」、「确认」等多余的操作，减少用户的操作成本。
3. 支持卡号手动修改。由于模型识别结果准确率无法达到 100%，所以该系统需要有一定的容错空间，可由用户自行修改卡号。