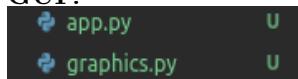


GUI :

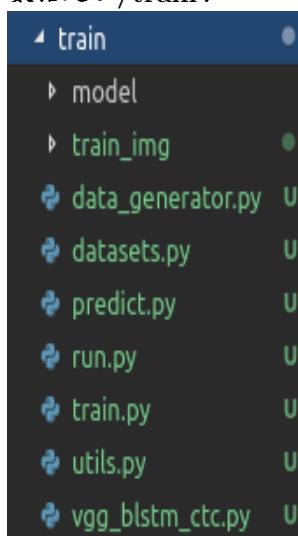
存放图形界面模块，使用PyQt5编写，UI整体使用Qt Designer完成，扁平化简洁界面设计。

positioning :

银行卡卡号定位模块，Image.py中定义了Image类，用于处理银行卡图片，利用opencv对图片进行相关处理并对定位卡号区域进行定位。

identification :

卡号识别模块，vgg_blstm_ctc.py中定义了识别所用的网络模型，predict.py提供了单张卡号识别的接口。

数据处理/train :

该文件存放了模型的具体训练代码，包括训练图片、数据增强模块、网络模型构建代码、训练代码，网络模型是使用keras构建而成的crnn网络模型。train.img存放的是原始训练数据。data_generator.py生成网络输入数据迭代器。datasets.py数据增强，生成训练数据、测试数据的图片文件名和标签文件，predict.py预测输出，train.py定义了训练模型，vgg_blstm_ctc.py网络模型搭建，run.py开始训练模型，utils.py一些工具函数。

首先项目采用的网络模型为当下较为流行的图文识别模型CRNN来对银行卡卡号进行识别。项目的第一步是对银行卡卡号进行定位，然后将银行卡卡号区域图片作为CRNN模型的输入，输出结果为识别出的卡号。

银行卡卡号定位模块具体设计思路：采取opencv来对其进行定位。通过查阅相关信息我们可以知道银行卡宽高分别为86.60毫米、53.98，高宽之比约为0.623，所以我们首先判断待测图片的高宽之比是否在0.60~0.69之间，如果在我们先调整图片像素大小为550x350。否则我们通过对原图像进行灰度、中值滤波、Scale边缘检测、二值处理，去除多余部分的背景。具体实现如下图所示：

```

def removeBackground(self):
    resize_img = cv2.resize(self.img,(self.HEIGHT,self.WIDTH),0,0,cv2.INTER_NEAREST) #调整图片大小
    salt_img = resize_img
    gray_img = cv2.cvtColor(resize_img, cv2.COLOR_BGR2GRAY) #灰度处理
    blur_img = cv2.medianBlur(gray_img,9) #中值滤波去除噪点
    x = cv2.Sobel(blur_img,cv2.CV_32F,0,1,3) #Sobel边缘检测
    y = cv2.Sobel(blur_img,cv2.CV_32F,0,1,3)
    absX = cv2.convertScaleAbs(x)
    absY = cv2.convertScaleAbs(y)
    sobel_img = cv2.addWeighted(absX,0.5,absY,0.5,0)
    thresh_img = cv2.adaptiveThreshold(sobel_img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,3,0) #自适应二值化
    cnts,_ = cv2.findContours(thresh_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) #找到最大连通区域
    temp = 0
    W = 0
    H = 0
    X = 0
    Y = 0
    for i in range(0,len(cnts)):
        x,y,W,h = cv2.boundingRect(cnts[i])
        if(temp < w + h):
            temp = w+h
            W = w
            H = h
            X = x
            Y = y
    remove_back_img = resize_img[Y:H,X:X+W]
    return cv2.resize(remove_back_img,(self.HEIGHT,self.WIDTH),cv2.INTER_NEAREST)

```

以测试图片1.jpeg为例,不仅对其大小进行了改变并且将多余背景部分剔除掉了。



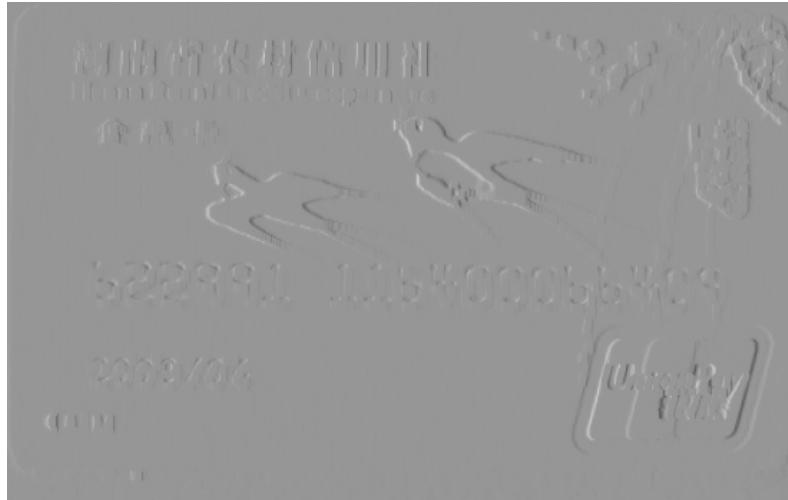
然后通过依然通过opencv对调整过的图片进行相关的图形学变换，以此来获得银行卡号的相关区域，具体实现过程为，首先对调整过后的图片进行灰度处理，并且进行膨胀腐蚀变化，而后对图片进行浮雕化处理，即突出图像的边缘部分。经过这几部变化后图片变为如下所示。

```

def position(self,img):
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray_img = cv2.dilate(gray_img,None,iterations=2)
    gray_img = cv2.erode(gray_img,None,iterations=2)

    emboss_img = self.embossment(gray_img)

```



接下来继续对图片进行相关图形学变化，在经过Sobel边缘检测，自适应二值化以及膨胀模糊后的图像最后呈现如下所示图形：

```

sobel_x = cv2.Sobel(emboss_img,cv2.CV_32F,1,0,3) #边缘检测
sobel_y = cv2.Sobel(emboss_img,cv2.CV_32F,0,1,3)
absX = cv2.convertScaleAbs(sobel_x)
absY = cv2.convertScaleAbs(sobel_y)
sobel_img = cv2.addWeighted(absX,0.5,absY,0.5,0)
sobel_img = cv2.medianBlur(sobel_img,11) #中值模糊
sobel_img = cv2.dilate(sobel_img,None,iterations=2) #膨胀

_,threshold = cv2.threshold(sobel_img,10,255,cv2.THRESH_BINARY) #二值化
threshold = cv2.GaussianBlur(threshold,(9,9),0) #高斯模糊

```



很容易可以看出在图像中间位置靠下部分是我们银行卡卡号所在区域，又因为该图像为二值处理过后的图片，通过对该图像水平方向的黑色像素进行统计，统计结果如下图所示



通过上面这幅图发现，除了顶部位置印有银行名称部分，剩下黑色像素最少的区域就是所要找的银行卡卡号区域。验证结果查看其他测试图发现都是如此。



除去顶部，计算平均值最小的区域就是银行卡卡号位置所在，选取图片高的1/10为一个区间用于计算不同区域的平均值。具体实现以及定位结果如下。

```
H = len(array)
label_H = int(H / 10)
min_ = sum(array)
ans = 0
for i in range(int(1/2 * H) - label_H): #从图像高2/5位置处开始进行平均值计算。
    a = int(2/5 * H) + i
    b = int(2/5 * H) + i + label_H
    mean = array[a:b].mean()
    if mean < min_:
        ans = a
        min_ = mean
    if a > 0.6 * H:
        return ans,ans+label_H
```



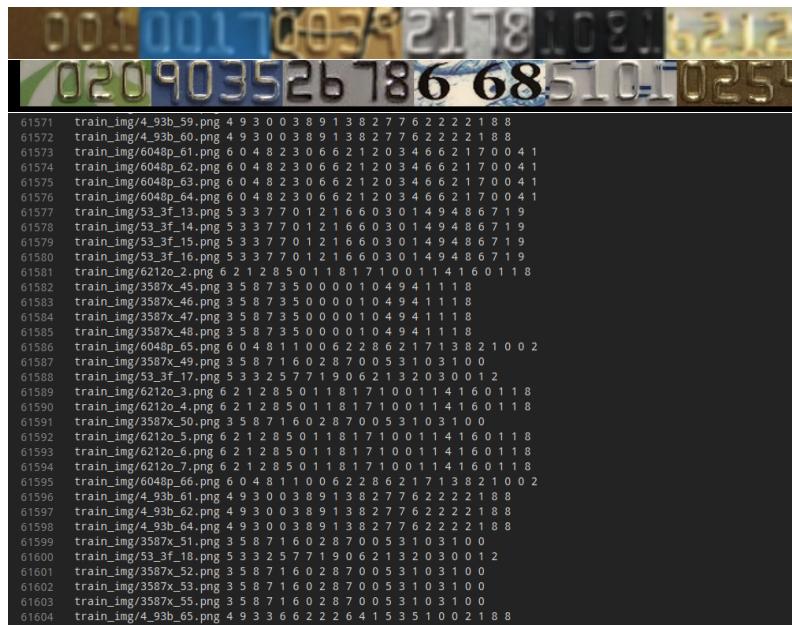
但是这其实并不是完美的结果，通过测试后发现这样定位后的图片识别效果并不是很好，仍然需要对其进行一些裁剪。思路与定位卡号区域大体相同，但有很多细节问题。测试结果如下所示：



数据处理及卡号识别具体设计思路：直接利用上面定位后的结果来进行识别，因此对训练数据要增加其数字长度，所以对训练数据集中的图片随机进行拼接。实现步骤如下：

```
H,W,C = img.shape
num = int(rand(4,6))
img_list = []
for i in range(num):
    imgs = random.sample(img_list,1)
    dst = np.zeros((H,W*(num+1),C),np.uint8)
    for h in range(H):
        for w in range(W):
            dst[h,w] = img[h,w]
```

除此还对数据集进行了其他方面的扩充，比如改变图片的大小，对图片进行部分位移，增加高斯噪声，对图像进行模糊处理，颜色变换等。具体实现代码放在datasets.py中，直接运行即可，运行结果除了增强后的数据集，还将这些图片的相对路径以及标签写在txt文件当中，并分成了训练集以及测试集。因为是对每张图片进行80张左右的数据增强，所以使用了多进程加速程序的运行速度。示例图如下所示：



网络识别使用的是CRNN文本识别网络。不同银行的银行卡卡号长度不同，因此对卡号的识别的过程中涉及到对不定字长的文本预测。如果将银行卡卡号进行字符分割，将其数字一个个分离开，再使用CNN分别对单个数字进行识别是可以的并且很好的避免了对不定字长字符识别的问题，但是银行卡卡号区域部分的字符分割并不简单，而且单个数字识别极大的降低了识别的效率。因此我们使用CNN与RNN相结合的图像序列识别来解决这一问题，CNN用于提取图像特征图谱，而RNN实现对不定字长字符的处理。CNN我们具体采用的是VGG模型，而RNN我们采用的是LSTM网络模型。CNN部分网络模型搭建如下图所示：

```
#CNN part
inputs = Input(shape=(picture_height, picture_width, 1), name='pic_inputs') # H*W*1 32*256*1
x = Conv2D(64, (3,3), strides=(1,1), padding='same', kernel_initializer=initializer, use_bias=True, name='conv2d_1')(inputs) # 32*256*64
x = BatchNormalization(name='BN_1')(x)
x = Activation('relu', name='relu_1')(x)
x = MaxPooling2D(pool_size=(2,2), strides=2, padding='valid', name='maxpool_1')(x) # 16*128*64

x = Conv2D(128, (3,3), strides=(1,1), padding='same', kernel_initializer=initializer, use_bias=True, name='conv2d_2')(x) # 16*128*128
x = BatchNormalization(name='BN_2')(x)
x = Activation('relu', name='relu_2')(x)
x = MaxPooling2D(pool_size=(2,2), strides=2, padding='valid', name='maxpool_2')(x) # 8*64*128

x = Conv2D(256, (3,3), strides=(1,1), padding='same', kernel_initializer=initializer, use_bias=True, name='conv2d_3')(x) # 8*64*256
x = BatchNormalization(name='BN_3')(x)
x = Activation('relu', name='relu_3')(x)
x = Conv2D(512, (3,3), strides=(1,1), padding='same', kernel_initializer=initializer, use_bias=True, name='conv2d_4')(x) # 4*64*512
x = BatchNormalization(name='BN_4')(x)
x = Activation('relu', name='relu_4')(x)
x = MaxPooling2D(pool_size=(2,1), strides=2, name='maxpool_3')(x) # 4*64*256

x = Conv2D(512, (3,3), strides=(1,1), padding='same', kernel_initializer=initializer, use_bias=True, name='conv2d_5')(x) # 4*64*512
x = BatchNormalization(name='BN_5')(x)
x = Activation('relu', name='relu_5')(x)
x = Conv2D(512, (3,3), strides=(1,1), padding='same', kernel_initializer=initializer, use_bias=True, name='conv2d_6')(x) # 4*64*512
x = BatchNormalization(name='BN_6')(x)
x = Activation('relu', name='relu_6')(x)
x = MaxPooling2D(pool_size=(2,1), strides=2, name='maxpool_4')(x) # 2*64*512

x = Conv2D(512, (2,2), strides=(1,1), padding='same', activation='relu', kernel_initializer=initializer, use_bias=True, name='conv2d_7')(x) # 2*64*512
x = BatchNormalization(name='BN_7')(x)
x = Activation('relu', name='relu_7')(x)
conv_output = MaxPooling2D(pool_size=(2, 1), name='conv_output')(x) # 1*64*512
```

RNN部分网络模型搭建如下图所示：

```

# RNN part
y = Bidirectional(LSTM(256, kernel_initializer=initializer, return_sequences=True), merge_mode='sum', name='LSTM_1')(rnn_input) # 64*512
y = BatchNormalization(name='BN_0')(y)
y = Bidirectional(LSTM(256, kernel_initializer=initializer, return_sequences=True), name='LSTM_2')(y) # 64*512

y_pred = Dense(num_classes, activation='softmax', name='y_pred')(y) # 64*11 试用test_rnn层
# 在backend的实现中，loss的计算没有执行softmax操作所以这里必须在使用softmax!!!
base_model = keras.models.Model(inputs=inputs, outputs=y_pred)
print('BASE_MODEL: ')
base_model.summary()

```

除了上面两个网络搭建部分我们还需要构建CNN与RNN之间的转换层，以及将RNN所做的每帧预测转换成标签序列。整个网络模型如下表所示。

Layer (type)	Output Shape	Param #
pic_inputs (InputLayer)	(None, 32, 256, 1)	0
conv2d_1 (Conv2D)	(None, 32, 256, 64)	640
BN_1 (BatchNormalization)	(None, 32, 256, 64)	256
relu_1 (Activation)	(None, 32, 256, 64)	0
maxpl_1 (MaxPooling2D)	(None, 16, 128, 64)	0
conv2d_2 (Conv2D)	(None, 16, 128, 128)	73856
BN_2 (BatchNormalization)	(None, 16, 128, 128)	512
relu_2 (Activation)	(None, 16, 128, 128)	0
maxpl_2 (MaxPooling2D)	(None, 8, 64, 128)	0
conv2d_3 (Conv2D)	(None, 8, 64, 256)	295168
BN_3 (BatchNormalization)	(None, 8, 64, 256)	1024
relu_3 (Activation)	(None, 8, 64, 256)	0
conv2d_4 (Conv2D)	(None, 8, 64, 256)	590080
BN_4 (BatchNormalization)	(None, 8, 64, 256)	1024
relu_4 (Activation)	(None, 8, 64, 256)	0
maxpl_3 (MaxPooling2D)	(None, 4, 64, 256)	0
conv2d_5 (Conv2D)	(None, 4, 64, 512)	1180160
BN_5 (BatchNormalization)	(None, 4, 64, 512)	2048
relu_5 (Activation)	(None, 4, 64, 512)	0
maxpl_4 (MaxPooling2D)	(None, 2, 64, 512)	0
conv2d_7 (Conv2D)	(None, 2, 64, 512)	1049088
BN_7 (BatchNormalization)	(None, 2, 64, 512)	2048
relu_7 (Activation)	(None, 2, 64, 512)	0
conv_output (MaxPooling2D)	(None, 1, 64, 512)	0
permute (Permute)	(None, 64, 512, 1)	0
for_flatten_by_time (TimeDistributed)	(None, 64, 512)	0
LSTM_1 (Bidirectional)	(None, 64, 256)	1574912
BN_8 (BatchNormalization)	(None, 64, 256)	1024
LSTM_2 (Bidirectional)	(None, 64, 512)	1050624
y_pred (Dense)	(None, 64, 11)	5643

以上模型搭建完毕后，制作数据生成器，在data_generator.py文件中定义了DataGenerator类用于声明网络输入数据制作器，从train.txt以及val.txt中获取数据图片，以及对应标签，get_data()方法用于生成一个batch的数据。执行run.py开始模型的训练。

```

1237/1237 [=====] - 918s 742ms/step - loss: 6.8803 - val_loss: 3.3761
Epoch 2/100
1237/1237 [=====] - 915s 739ms/step - loss: 3.0095 - val_loss: 3.4074
Epoch 3/100
1237/1237 [=====] - 914s 739ms/step - loss: 2.9022 - val_loss: 3.3496
Epoch 4/100
1237/1237 [=====] - 911s 736ms/step - loss: 2.8390 - val_loss: 3.0356
Epoch 5/100
1237/1237 [=====] - 916s 740ms/step - loss: 2.8298 - val_loss: 3.0434
Epoch 6/100
1237/1237 [=====] - 911s 737ms/step - loss: 2.8153 - val_loss: 3.0788
Epoch 7/100
1237/1237 [=====] - 920s 743ms/step - loss: 2.7812 - val_loss: 3.0254
Epoch 8/100
1237/1237 [=====] - 916s 740ms/step - loss: 2.7621 - val_loss: 2.8778
Epoch 9/100
1237/1237 [=====] - 912s 737ms/step - loss: 2.7546 - val_loss: 2.9118
Epoch 10/100
1237/1237 [=====] - 912s 738ms/step - loss: 2.7811 - val_loss: 2.9785
Epoch 11/100
1237/1237 [=====] - 910s 736ms/step - loss: 2.8117 - val_loss: 2.8367
Epoch 12/100
1237/1237 [=====] - 909s 735ms/step - loss: 2.7341 - val_loss: 2.9510
Epoch 13/100
1237/1237 [=====] - 908s 734ms/step - loss: 2.7249 - val_loss: 2.9412
Epoch 14/100
173/1237 [==>.....] - ETA: 12:41 - loss: 2.8259

```



从上图看出，模型在训练8个epoch后，loss基本稳定在2.75左右，所以我们这个时候可以选择停止模型训练。最后将模型放入项目目录下的model目录中。

最后GUI模块的实现，使用Qt Designer先设计好整体布局，然后修改相关属性。最后UI效果如下图所示：



点击打开图片选择单张图片进行测试。



打开图片后，点击卡号定位。



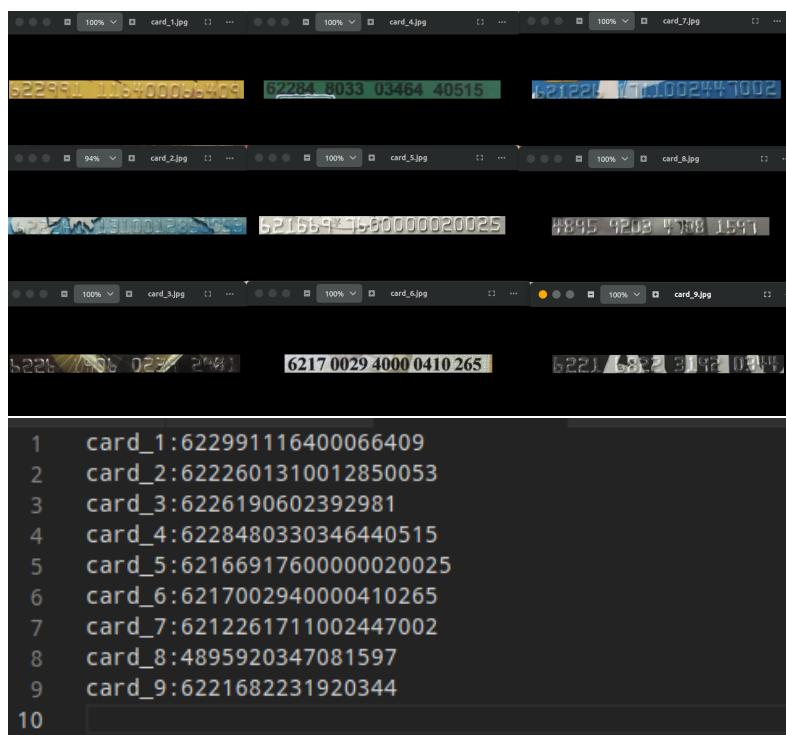
最后点击图片识别。



如果识别有误，可以对识别结果进行修改，或者选择手动定位，更加精准的定位银行卡卡号区域，定位精准识别的效果也就会更好一些。



执行batch_test.py可实现对test_images目录下的图片进行批量定位与识别。定位以及输出结果在test_results目录下。



识别结果受卡号定位模块很大影响，如果定位不够精确或者定位长度过长都会影响到最后的识别结果如下图：



当我们选择“手动定位”时，准确定位卡号区域识别结果正确：



常见的错误还有当分割号码中间有明显图案时也会影响最后的识别结果，对于这种错误只能通过优化训练模型来改进。





补充：

模型额外使用了两张图片作为训练数据。

上述部分测试图片来自网络。

可执行文件应在全英文路径下测试运行

参考资料：

<https://www.jianshu.com/p/14141f8b94e5>

https://github.com/Liumihan/CRNN_kreas

https://blog.csdn.net/qq_37053885/article/details/79248986

https://blog.csdn.net/this_is_id/article/details/84966349