1 项目概述

人脸识别在现代社会中有着非常广泛的应用。此次数字图像处理课程中，我选择做一个基于深度学习的人脸识别系统。我设计的基于深度学习的人脸识别系统，通过摄像头获取图片，使用 dlib 分类器定位人脸，使用卷积神经网络进行人脸识别。我设计的系统有三个输出类别。除了可以识别我和另外一个同学外，我还在网络上的 LFW 人脸图片库中下载了一定的图片，来作为第三个类别为图片标记标签。关于系统的详细设计思路和实现过程将在下面几个部分进行详述。

2 模块功能和结构划分

我实现的系统共分为摄像头采集和处理图片模块、本地图片处理模块、卷积神经网络搭建和训练模块以及测试模块四个部分，每个部分保存在一个.py 文件中。

摄像头采集和处理图片模块主要负责采集人脸图片，通过调用摄像头截取照片，定位人脸，指定规格化，并最终保存人脸图片。

本地图片处理模块主要用于处理我在网络上下载下来作为第三个分类类别使用的图片，其主要功能是将图片读入，使用 dlib 定位人脸，然后将人脸图片保存成我设定好的规格以便使用。

卷积神经网络搭建和训练模块主要完成前向传播、划分训练集与测试集、反向传播等功能，是整个系统的关键部分。

测试部分主要来调用摄像头捕获人脸图片，然后读入之前保存好的训练成熟的网络进行识别。

系统结构如下图所示：

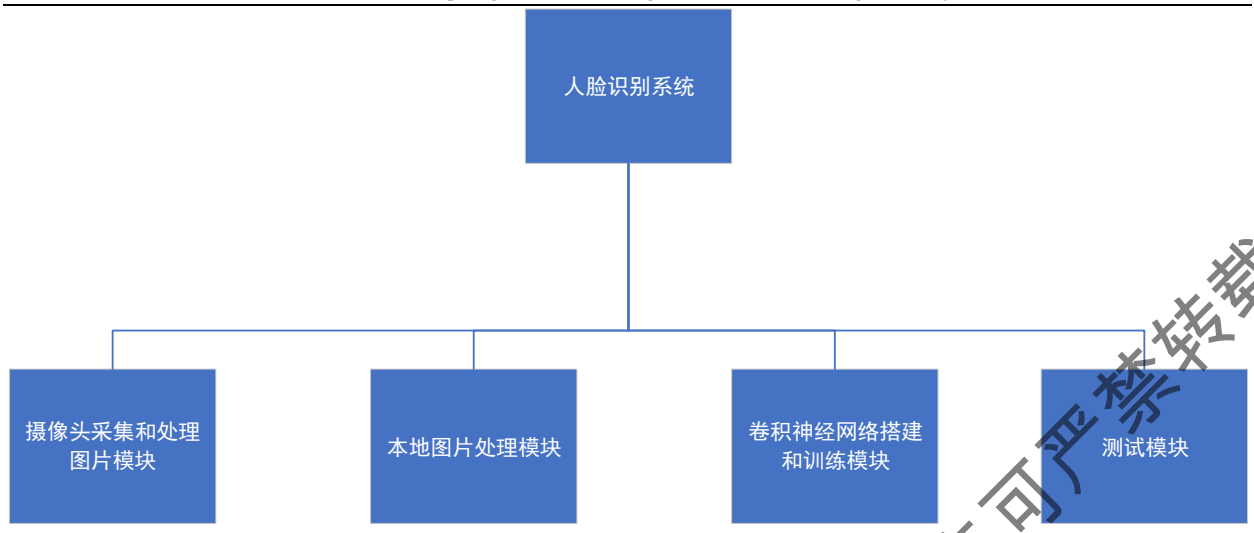

图 1 系统结构

3 项目实现

3.1 样本的采集和处理

样本的采集和处理主要集中在摄像头采集和处理图片模块、本地图片处理两个模块中。

首先，使用摄像头采集和处理图片模块来采集和处理我和另外一个同学的图片。使用 OpenCV 调用摄像头，然后截取图片并灰度化，之后使用 dlib 找到人脸并以合适的规格保存图片。具体流程如下图所示：

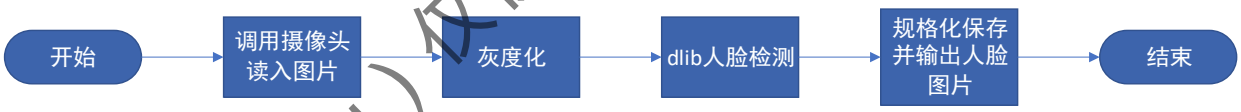

图 2 采集流程

在网上下载 LFW 图片库的图片并不符合我想使用的规格，所以设计了本地照片处理模块。在本地照片处理中，流程基本和上面一致，不同的地方在于输入不再从摄像头中读入，而是本地的图片。具体流程如下图所示：

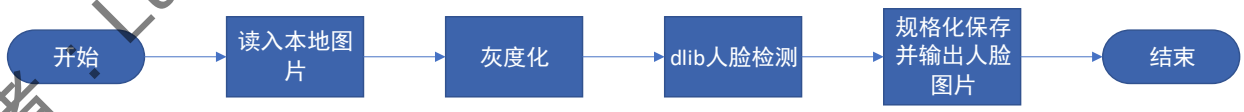

图 3 本地图片处理流程

下面来解释与样本采集与处理有关的关键代码。

首先使用 camera = cv2.VideoCapture(0)来打开摄像头，在经过灰度化处理后，使用 dlib 分类器定位人脸，获取到相应的坐标。

```
dets = detector(gray_img, 1)  #detector会返回识别到的人脸的矩形的左下角和右上角的坐标

for i, d in enumerate(dets):  #enumerate用于遍历括号中的元素及其下标，其中i对应元素下标，d对应元素
    x1 = d.top() if d.top() > 0 else 0  #通过left、right、top、down获取矩形的四个坐标x1、x2、y1、y2
    y1 = d.bottom() if d.bottom() > 0 else 0
    x2 = d.left() if d.left() > 0 else 0
    y2 = d.right() if d.right() > 0 else 0
```

图 4 定位人脸坐标

获取到人脸的左下角和右上角坐标后，在通过摄像头获取的图片中把人脸截取出来，保存成我们想要的规格。可通过代码 face = img[x1:y1,x2:y2]，face = cv2.resize(face,(size,size))来完成。

在完成对图片的采集和处理后，接下来进行神经网络的搭建工作。

3.2 卷积神经网络的构建和训练

3.2.1 卷积、池化、激活和舍弃

卷积层在整个网络中具有重要的作用，用来提取样本的特征并进行处理，选择好卷积层对整个卷积神经网络识别的准确率有重要的影响。在选择卷积核的时候，一般采用“小卷积核，多卷积层数”的原则。因此，我最终采用了在图像识别中非常常用的三层卷积的结构。卷积核大小采用 3*3，在行和列上的移动步长均定为 1。为了保证样本在经过卷积层后规格不变，方便下一步处理，我使用了“全零填充”，也就是在样本外面加一层“0”。

在每一层卷积层后，我都连接了一个池化层，采用最大池化来进行下采样。我将池化层的分辨率定为 2*2，也就是将卷积后的样本分成了很多个 2*2 的小区域，在每个小区域里采集最大值，然后合并起来作为池化层的输出结果。用于池化的原因，虽然卷积过后样本规格并没有发生变化，但是池化会使样本的长宽减小一半。

在完成池化后，我又让样本过了一次激活函数。激活函数的作用是给神经网络加入一些非线性的因素，使得神经网络可以更好的解决分类问题。在选择激活函数的时候，我采用了 relu 函数。我经过查阅文献发现，当前很多文献都推荐使用 relu 函数，原因是它的计算代价小并且能够减轻梯度消失的问题。为了减小训练时间，我最终选择了使用 relu 函数。

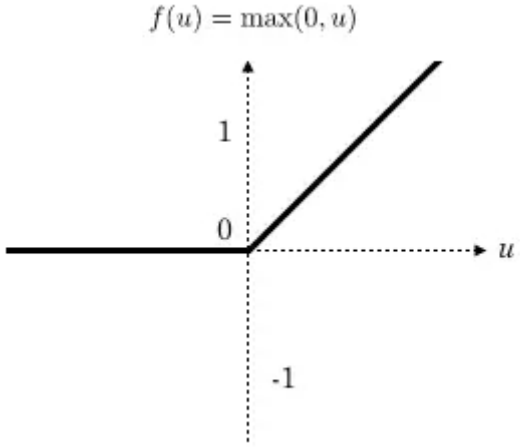

图 5relu 函数示意图

在完成激活往下一层网络输入前，我又引入了舍弃。引入舍弃的原因在于，我们构造的网络需要训练的参数普遍很大，而且很容易产生过拟合的问题。为了增强模型的泛化性，可以通过舍弃，随机的让某些参数不参与训练。但是，舍弃仅仅是针对训练阶段而言的，在模型真实使用的时候，所有的参数都是要参与到计算中来的。

最终，这一部分的网络结构如下图所示：

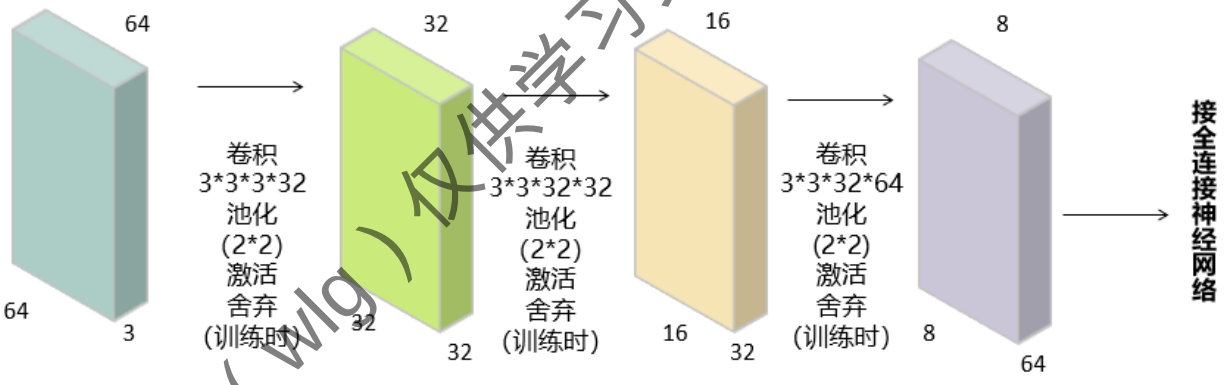

图 6 结构示意

在我设计的网络结构中，首先将准备好的样本输入（64*64*3），经过卷积后，样本规格不变，但是通道数由 3 变为了 32。然后经过池化层时，样本的长宽各减半，所以在经过第一层卷积和池化后，样本规格变为了 32*32*32。其他层的计算方法与第一层完全相同。在经过了第二层卷积核池化后，样本规格变为了 16*16*32。在经过了第三层卷积核池化过后，样本的规格变为了 8*8*64，并作为输入进入到全连接神经网络中。

代码上，配置卷积层、池化层以及使用激活函数和引入舍弃，都可以通过调用 TensorFlow 里的函数实现，为了方便起见，可以自己定义函数，在里面再调用

TensorFlow 的函数，这样只需要在自己的函数里把 TensorFlow 需要的参数配置好，以后只需要调用自己的函数传样本和待优化参数就可以了。具体如下所示：

```
def conv2d(x, W):  #卷积，参数分别是喂入图片的描述、对卷积核的描述、卷积核移动的步长
    return tf.nn.conv2d(x, W, strides=[1,1,1,1], padding='SAME')

def maxPool(x):  #池化，参数分别是喂入图片的描述，池化核的描述、池化核移动的步长
    return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

def dropout(x, p):  #定义在训练过程中的舍弃，参数分别是来自上层的输出和暂时舍弃的概率
    return tf.nn.dropout(x, p)
```

图 7 自己包装好创建卷积、池化、舍弃的函数

3.2.2 全连接神经网络

在全连接神经网络中，我主要设置了一层隐藏层和一层输出层。根据我查阅的文献，增加隐藏层的层数可以降低网络误差，但是同时增加了网络的复杂性，容易使网络产生过拟合的倾向，因此往往通过增加隐藏层节点个数来获得较低的误差比增加层数要好一些。

为了确定设置多少个隐藏层节点较好，我进行了一些尝试，分别尝试了 256、512、1024 三种隐藏层节点情况下网络在测试集上的准确率。

为了控制变量，将输入样本确定为 6000，其中我、我 srf 的搭档王宇佳以及第三分类图片各 2000 张。训练集和测试集的比例设置为 95:5，batch_size 设置为 100 个样本，执行 400 个 batch_size 后停止。

256、512、1024 个隐藏层节点下的测试集准确率和 loss 如下图所示：

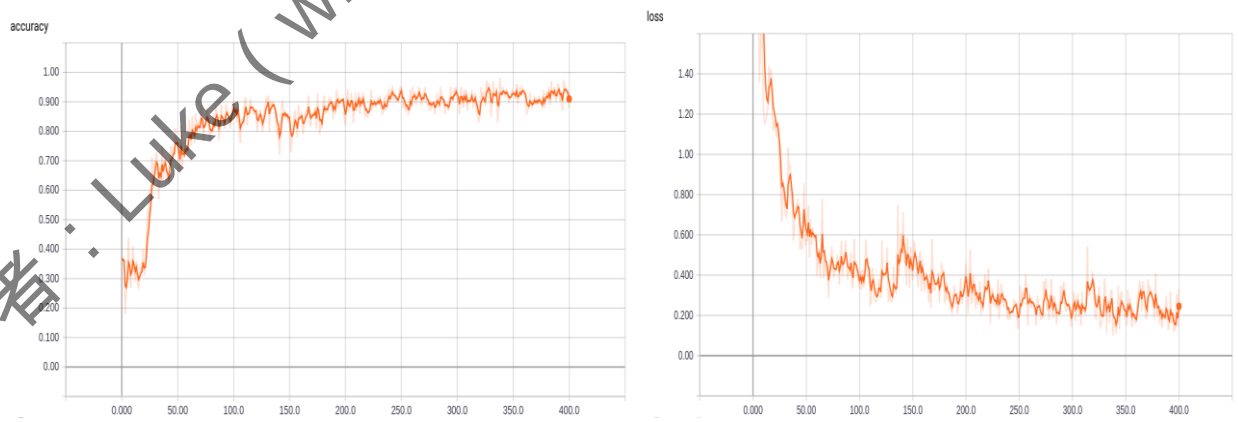

图 8 256 节点下测试集的准确率以及网络 loss

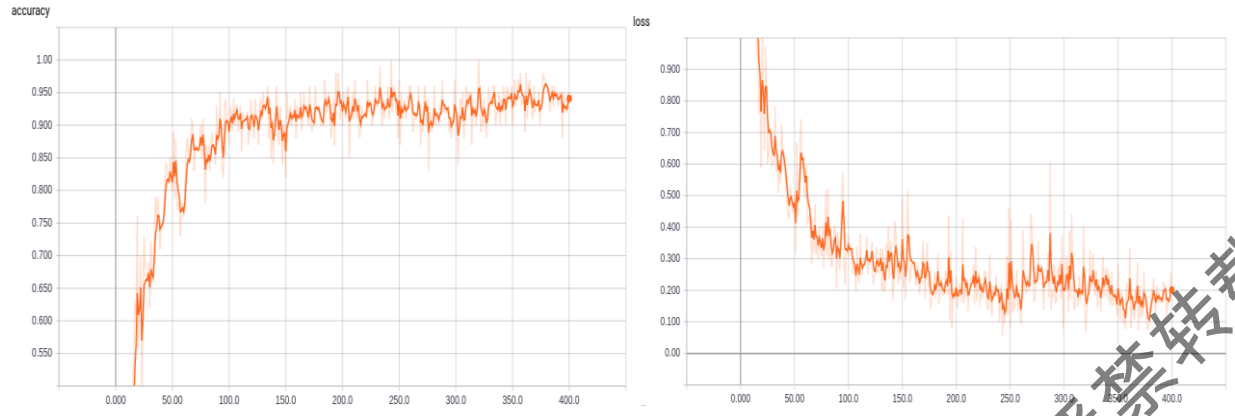

图 9 512 节点下测试集的准确率以及网络 loss

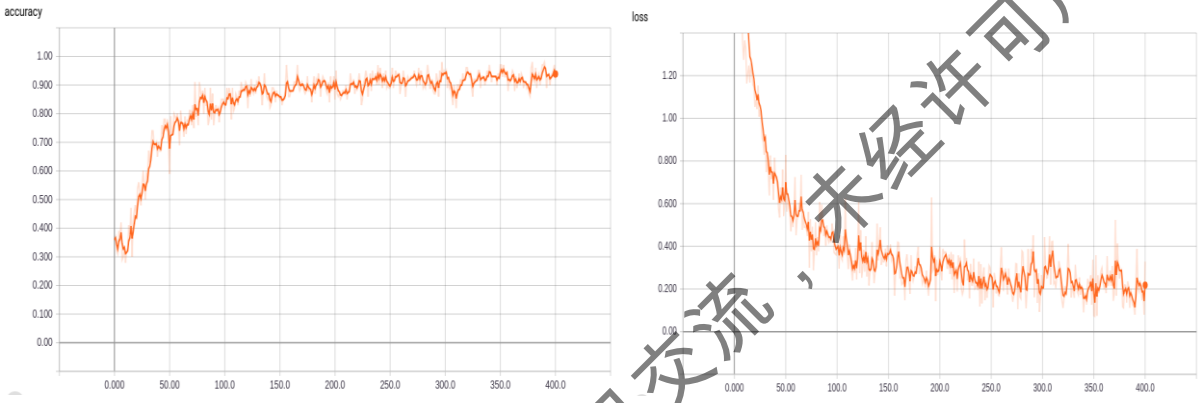

图 10 1024 节点下测试集的准确率以及网络的 loss

经过分析发现，当隐藏层节点数设置为 256 时，网络在测试集上的准确率大概在 0.96 左右，而 512 节点和 1024 节点的网络在测试集上的准确率均可达到 0.987 以上且相差不多。从时间来看的话，256 节点的网络和 512 节点的网络在完成所有训练所需要的时间上花费相当，而 1024 节点的网络所需的时间要长一些。因此我最终选中的准确率和时间俱佳的 512 个隐藏层节点。

3.2.3 训练方法的确定

在前两个小节中已将网络的搭建过程叙述完毕。在这一小节中主要叙述反向传播，也就是卷积神经网络的训练过程。

首先要确定的一个问题是采用什么方法，选用什么样的学习率进行训练。在阅读文献后，我主要选择了随机梯度下降、momentum 优化器、自适应梯度下降这三种方法来进行比较，以便选择一种合适的训练方法。

对于随机梯度下降这种训练方法，它使参数沿着梯度下降方向，即总损失减小的方向移动，实现更新参数。这种方法在训练过程中设置的学习率是固定不变的，不会根据训练情况动态调整。对于 momentum 优化器来说，它同样是和梯度有关，但是需要的参数除了学习率外还有一个超参数。对于自适应梯度下降来说，它和随机梯度下降一样，

都是使参数沿 loss 梯度下降方向减小，但是参数的学习率是自适应的，会根据训练过程动态调整学习率。

在确定训练方法的时候，同样为了控制变量，我同样将输入样本确定为 6000，每一类所占的比例仍然是三分之一，其他参数设置也与前面相同。为了简便起见，我将所有训练集图片训练完一次作为一轮，训练 8 轮后停止，然后查看结果。

首先，对于随机梯度下降，测试结果如下图所示：

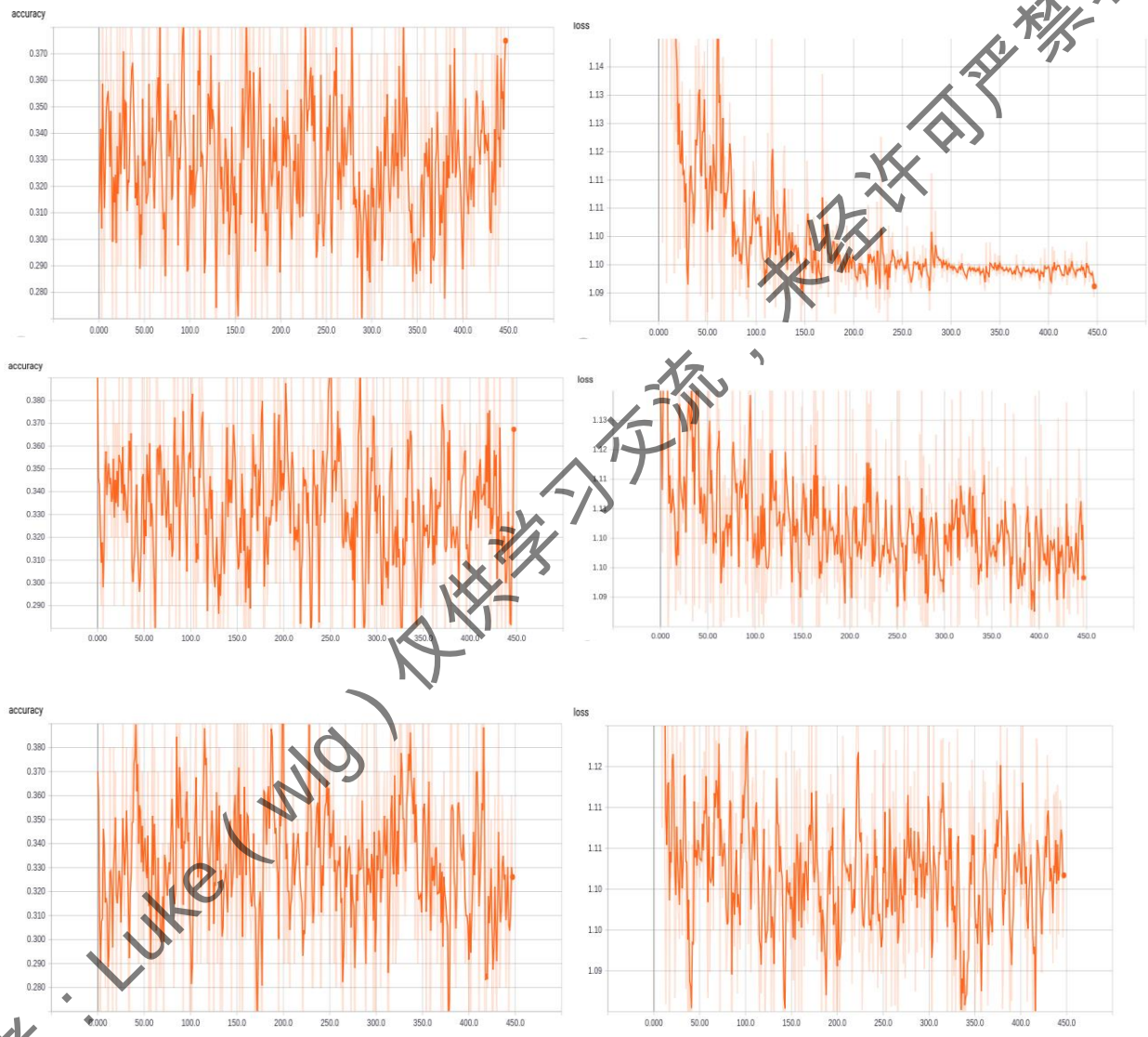

图 11 随机梯度下降测试结果

（左边为 Accurac，右边为 loss，从上至下学习率分别为 0.1，0.01，0.001）

可以发现，随机梯度下降在规定的训练轮数内，在三种学习率下 loss 都比较高，甚至震荡不收敛。

接下来，对于 momentum 优化器，测试结果如下：

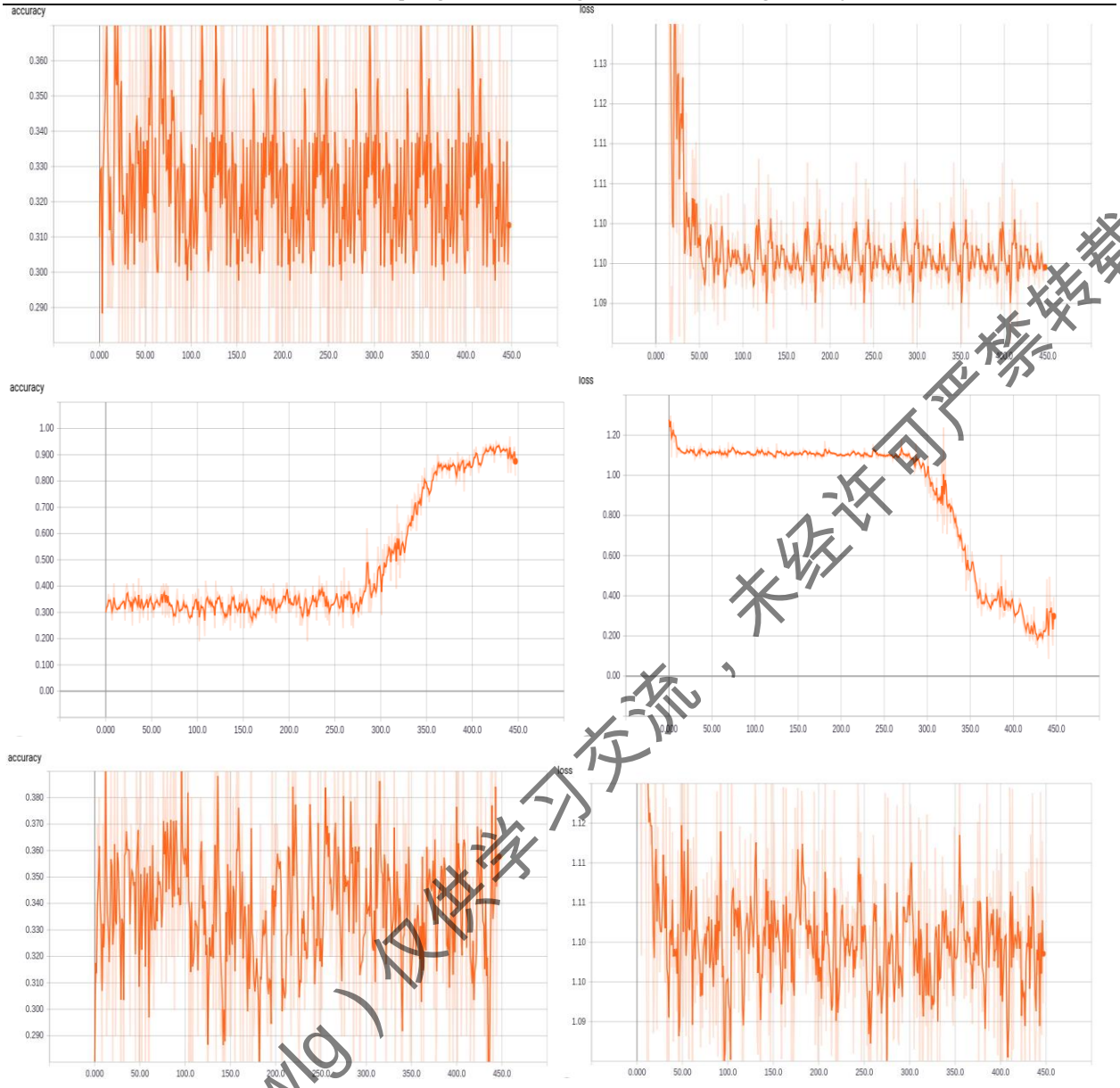

图 12momentum 优化器测试结果

（左边为 Accurac，右边为 loss，从上至下学习率分别为 0.1，0.01，0.001）

最终可以发现，momentum 优化器在学习率设置为 0.1、0.001 时，loss 居于高位或者震荡不收敛。而当学习率设置为 0.01 时，在经过较多轮数的训练后，逐渐收敛，在测试集上的准确率也在 0.9 左右。

最后测试自适应梯度下降的方法，测试结果如下图所示：

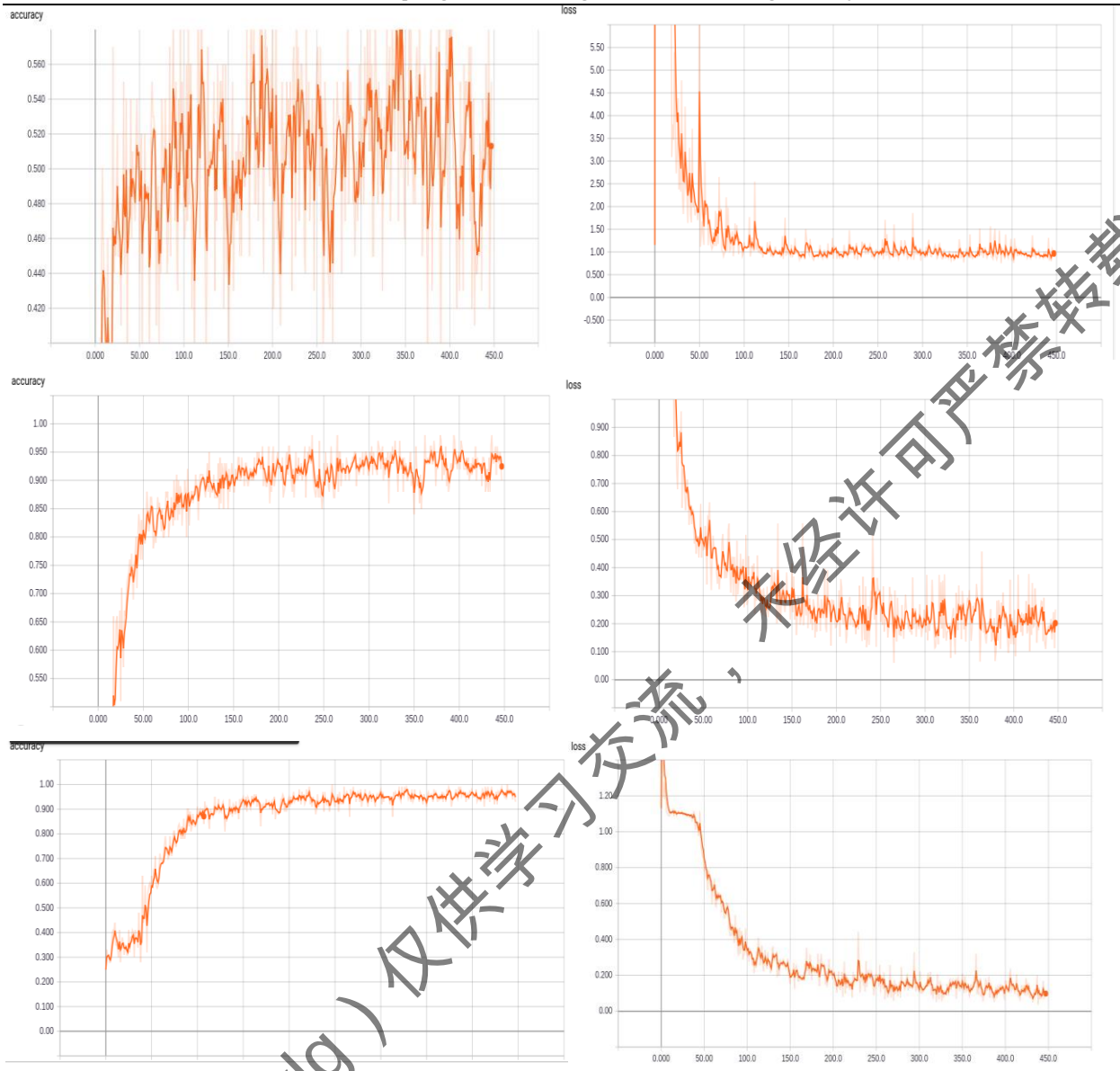

图 13 自适应梯度下降测试结果

（左边为 Accurac，右边为 loss，从上至下学习率分别为 0.1，0.01，0.001）

分析发现，当学习设置为 0.1 时，自适应梯度下降的方法的 loss 居高不下，在 1 左右徘徊。而当学习率下降到 0.01 或者 0.001 的时候，自适应梯度下降的方法能够迅速收敛，loss 小且在测试集上的准确率高。经过比较，学习率 0.001 和学习率 0.01 所用时间差不多但学习率为 0.001 时，在测试集准确率上稍微高一点。

总的来看，发现随机梯度下降收敛很慢，在规定的训练轮数里震荡不收敛或者 loss 维持在很高的水平。momentum 优化器参数多，收敛慢，耗时长，不太占优势。而自适应梯度下降的方法在规定训练轮数中收敛迅速，效果好，用时最短，故最终选用自适应梯度下降的方法，学习率定为 0.001。

3.3 部分代码细节解释

首先，在对网络进行训练的时候，很重要的一个工作就是划分训练集和测试集。在这里，我直接调用了 train_test_split 这个包里的函数进行了划分。

```
# 随机划分测试集与训练集，训练集占95%，测试集占5%
train_x,test_x,train_y,test_y = train_test_split(imgs, labs, test_size=0.05, random_state=random.randint(0,100))
```

图 14 划分测试集和训练集

此外，卷积神经网络如何确定输入的图片是哪一类，又如何将计算结果归到哪一类呢？要解决这个问题，首先要给我们我们向神经网络里输入的每一张图片打上标签。当从第一个类别的图片保存的文件夹里读入图片的时候，打一种标签；从第二个类别的图片保存的文件夹里读入图片的时候，打另外一种标签，以此类推。从可以通过如下代码完成：

```
labs = np.array([[0,1,0] if lab == face2_path else [0,0,1] if lab == my_faces_path else [1,0,0] for lab in labs])
```

图 15 打标签

在确定输出结果究竟是哪一类的时候，需要用到 tf.argmax 函数。我们可以将概率最大的位置所在的标号找出来，然后和我们的设定进行对比，从而找到输出结果究竟是哪一类。

```
def is_my_face(image):
    res = sess.run(predict, feed_dict={x: [image/255.0], keep_prob_5:1.0, keep_prob_75: 1.0})
    if res[0] == 0:
        return 1
    elif res[0] == 1:
        return 2
    elif res[0] == 2:
        return 3
```

图 16 我用来判断输出结果属于哪一类的函数

此外，为了能够将识别结果实时显示出来，我们可以在画面上将人脸框出，然后将类别的名字写出来。这就需要用到 OpenCV 里的一些函数，把方框的位置（在识别时已经由 dlib 得到）、字号、颜色、大小等参数传入，就可以实现了。具体如下：

```
name = "wlg"
font = cv2.FONT_HERSHEY_SIMPLEX;
cv2.rectangle(img, (x2,x1),(y2,y1), (255,0,0),3)
cv2.putText(img, name, (x2, x1), font, 1, (0, 255, 0), 1, cv2.LINE_AA);
cv2.imshow('image',img)
```

图 17 在人脸上画框和显示识别结果