

**《人工智能实验》课程大作业**



|  |  |
| --- | --- |
| **学年学期：** | 2021-2022 学年春夏学期 |
| **课程名称：** | 人工智能实验 |
| **小组成员：** | 黄嘉欣 白骁凯 夏宇航 |
| **所属院系：** | 信息与电子工程学院 |
| **所属专业：** | 信息工程 |
| **指导老师：** | 胡浩基、魏准 |

**提交日期 2022年5月30日**

基于MTCNN与轻量化网络的人脸识别

黄嘉欣、白骁凯、夏宇航

（浙江大学，浙江 杭州 310000）

摘 要：随着现代信息技术的快速发展，进行身份认证的技术转到了生物特色层面。现代生物辨别技术利用深度学习，将人脸所包含着的丰富信息提取到高维特征，从更高级的数据维度对人类进行辨识，当然特征提取也是图像和视频中视觉感兴趣的对象之一。与指纹、虹膜、语音等其余人体生物特色对比，人脸辨别更为直接、友善，无需扰乱人们的正常行为就能较好地达到辨别成效。在身份辨别、接见控制、视频会议、档案管理、电子相册、鉴于对象的图像和视频检索等方面有着宽泛的应用，是目前模式辨别和人工智能领域的研究热门之一。本文主要从基于MTCNN的人脸检测与基于fmoblienet的人脸识别这两个方向入手，对基本的研究原理进行说明，同时对创新点以及UI交互界面进行陈述，以达到完整搭建人脸识别应用的目的。

关 键 词：人脸检测；人脸识别；UI交互界面；

中图分类号：TU 000文献标志码：A文章编号：000-00(0000)00-000-00

Face recognition based on MTCNN and lightweight network

Huang JiaXin, Xiao KaiBai, Xia YuHang

*（Zhejiang University, Hangzhou 310000, China)*

**Abstract:** With the rapid development of modern information technology, identity authentication technology has been transferred to the level of biological characteristics. Modern biological discrimination technology uses deep learning to extract rich information contained in human faces to high-dimensional features, and to identify human beings from higher-level data dimensions. Of course, feature extraction is also one of the visual objects of interest in images and videos. Compared with other biological features such as fingerprints, iris and voice, face recognition is more direct and friendly, and can be achieved without disrupting people's normal behavior. It is one of the hot researches in the field of pattern discrimination and artificial intelligence. This paper mainly starts from the two directions of face detection based on MTCNN and face recognition based on FMOBlienet, explains the basic research principles, and at the same time states the innovation points and UI interactive interface, in order to achieve the purpose of building a complete face recognition application.

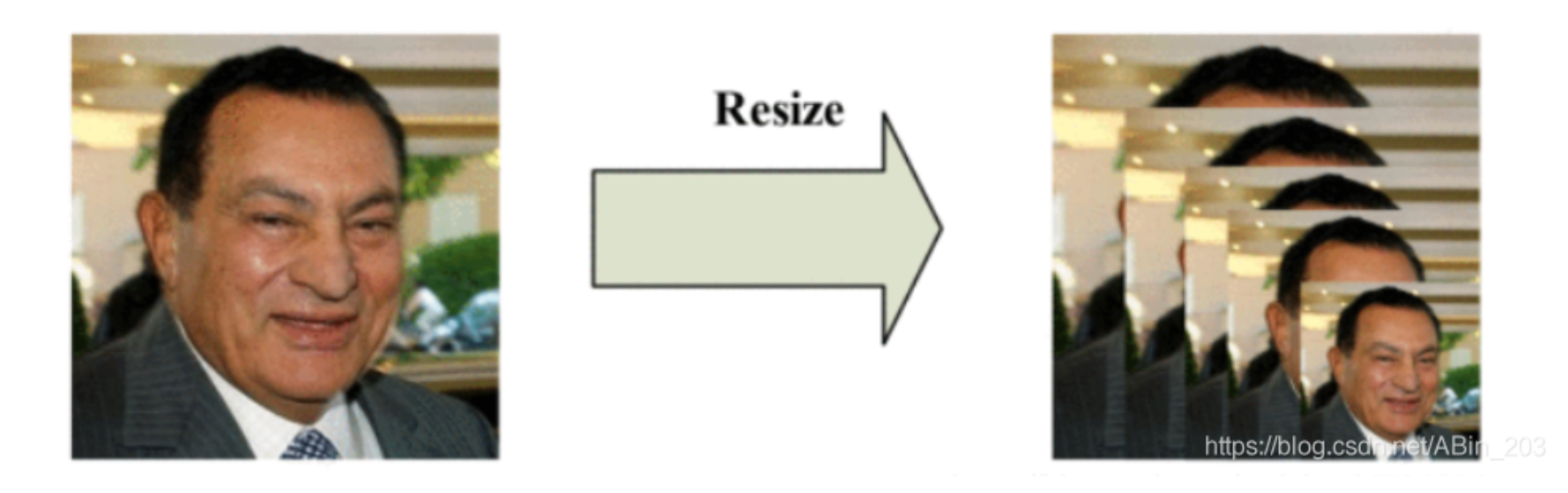
Chinese Library Classification Number：TU 000 Document Code：A Article Number：000-000(0000) 00-000-00

Keywords: Face detection; Face recognition; UI interactive interface;

第一部分: 基于MTCNN的人脸检测

MTCNN是2016年中国科学院深圳研究院提出的用于人脸检测任务的多任务神经网络模型，该模型主要采用了三个级联的网络，采用候选框加分类器的思想，进行快速高效的人脸检测。这三个级联的网络分别是快速生成候选窗口的P-Net、进行高精度候选窗口过滤选择的R-Net和生成最终边界框与人脸关键点的O-Net。和很多处理图像问题的卷积神经网络模型类似，该模型也用到了图像金字塔、边框回归、非最大值抑制等技术。

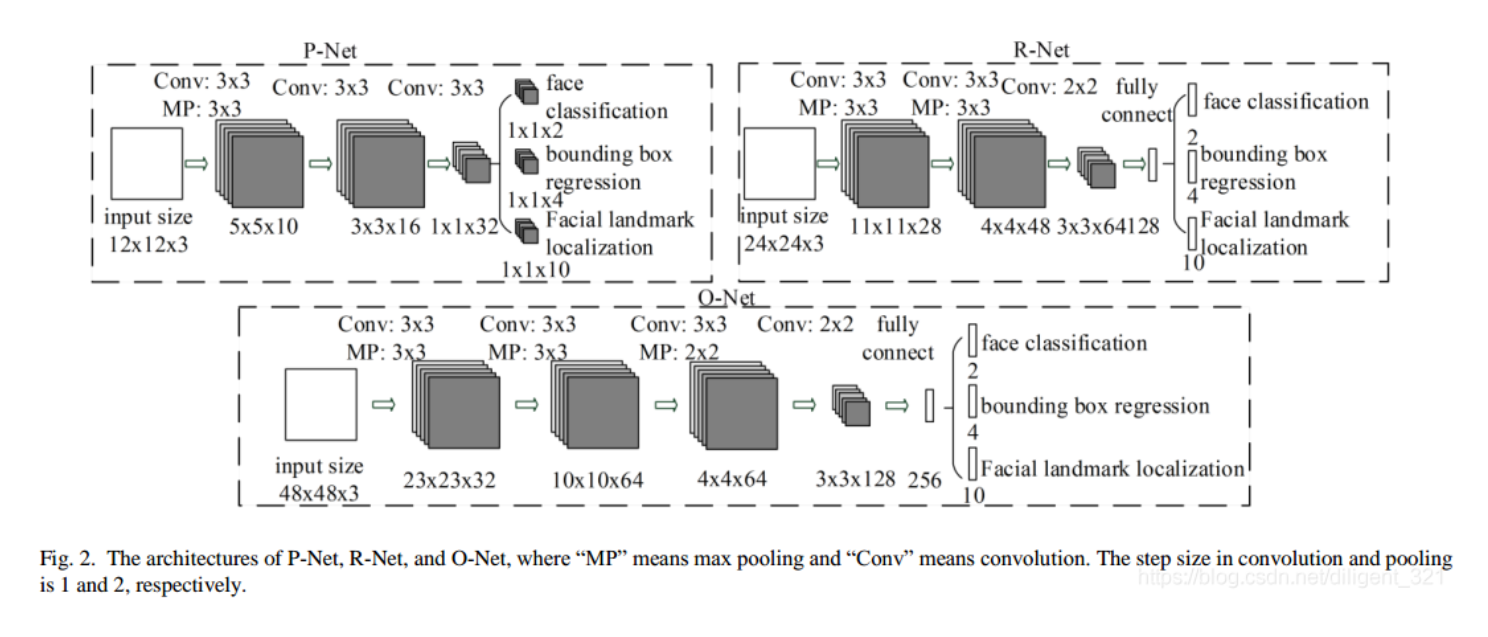
1. **图像金字塔**



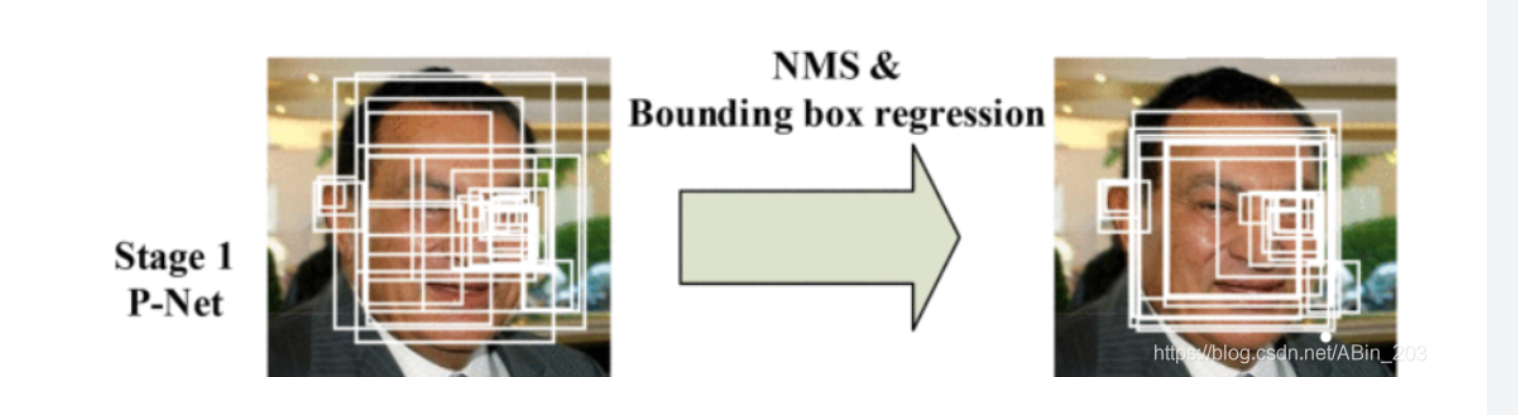
我们可以看到左图通过不同的缩放系数factor对图片进行缩放，每次将其缩小为原来的factor大小，从而得到右侧的一系列“图像金字塔”。在图中，取factor = 0.709 ≈ sqrt(2)/2，此时缩放后图像的宽高变为原来的sqrt(2)/2，面积变为原来的1/2。通过处理后，我们可以生成更多的原样本，在进行识别时就会有更多数据、更加准确，但同时也会带来处理时间太长等缺点。

在实现时，参数如min\_detection\_size=12代表检测方框的大小，min\_face\_size=20代表原图像中可识别人脸的最小尺寸，min\_length = min(height, width)是原图像可识别最大人脸的大小。取m=12/20= 0.6为初始缩放因子，以后以不断乘factor缩小规模，直至min\_length小于12。至此，得到一系列图像作为输入。从本质上来说，图像金字塔就是一种为了检测原图像中不同大小人脸而采取的策略。

1. **P-net**



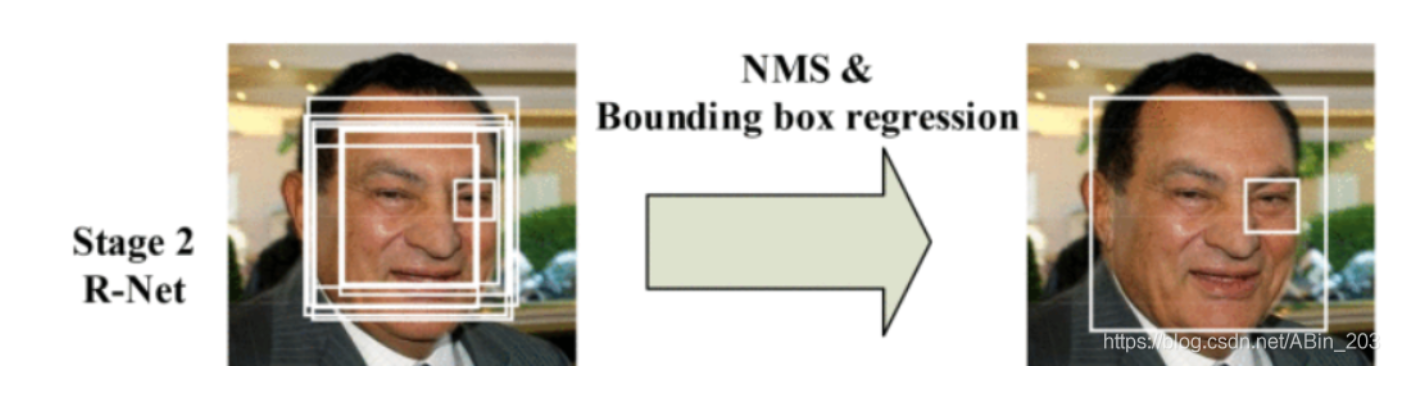
P-net输入维度: (12,12,3)，其需要判断该12\*12\*3的图像中是否有人脸，并给出人脸框和关键点的位置。网络的第一部分输出用来判断该图像是否存在人脸，输出向量大小1×1×2，也就是两个值；网络的第二部分给出框的精确位置，一般称为框回归；网络的第三部分给出人脸的5个关键点的位置，即左眼、右眼、鼻子、嘴角左侧、嘴角右侧。每个关键点需要使用两维来表示，因此输出向量大小为1×1×10。



P-net对图像进行初步的人脸识别，相当于一把筛子，将图像中所有可能的人脸全部筛选进来。在这个过程当中，可能会产生好几百个框，如此巨大的数量将会严重影响程序的运行速度，故需引入NMS和bbr以调整框的位置。

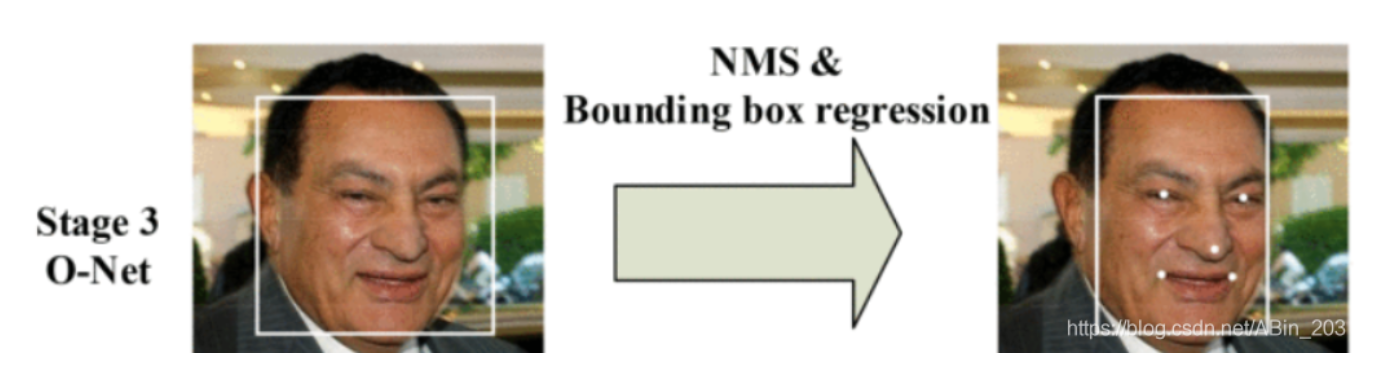
1. **R-net**

P-Net的所有候选框都将被输入到R-Net中，该网络结构通过边界框回归和NMS来滤除大量的false-positive区域。从网络图可以看到，R-net网络结构与P-Net存在一定差异，其多出一个全连接层，所以能够取得更好的抑制false-positive的效果。在将数据输入R-Net之前，需要将其维度缩放到24x24x3，经过处理后，R-Net将会去除大量的非人脸框，得到与P-net类型相同的三种输出结果。



1. **O-net**

但O-net阶段，我们需要通过更多的监督来识别面部区域，特别是五个面部关键点的位置。从网络图中我们可以发现，O-net与R-net相比多了一层卷积层，所以处理的结果会更加精细。此网络输入的图像大小为48x48x3，输出包括N个边界框的坐标信息，score以及关键点的位置。



1. **模型优化之水平矫正**

考虑到MTCNN只能给出人脸区域对应的矩形窗，在后续的应用中，我们将会截取矩形窗内的人脸，对其进行识别。为了应对不同状态的人脸图像，获得相对稳定的识别效果，我们需要根据检测得到的两眼角度对图像进行旋转，利用cv2.getRotationMatrix2D以及cv2.warpAffine进行仿射变换，使两眼连线保持水平。经过实验，水平校正极大地提高了算法的鲁棒性，保证了后续人脸识别的精度。在另一个方面，经过仿射变换后，图像在原有窗内会新增很多无效信息（填充0），因此，我们添加了自适应图片边框大小功能，完成后的代码如下：

def wrap\_affine(frame, first\_eye, second\_eye, scale=1.):

    eye\_center = ((first\_eye[0] + first\_eye[1])/2,

                  (second\_eye[0] + second\_eye[1])/2)

    dy = second\_eye[1] - first\_eye[1]

    dx = second\_eye[0] - first\_eye[0]

    rows, cols = frame.shape[:2]

    # 计算旋转角度

    angle = cv2.fastAtan2(dy, dx)

    rot = cv2.getRotationMatrix2D(eye\_center, angle, scale=scale)

    # 自适应图片边框大小

    cos = np.abs(rot[0, 0])

    sin = np.abs(rot[0, 1])

    new\_w = rows \* sin + cols \* cos

    new\_h = rows \* cos + cols \* sin

    rot[0, 2] += (new\_w - cols) \* 0.5

    rot[1, 2] += (new\_h - rows) \* 0.5

    w = int(np.round(new\_w))

    h = int(np.round(new\_h))

    rot\_img = cv2.warpAffine(frame, rot, dsize=(w, h))

    # rot\_img = cv2.flip(rot\_img, 0)

    return rot\_img

至此，我们已经完成了人脸检测的任务，从原始图片中得到了裁剪后的人脸图像。接下来，我们会将图像统一到96×112大小，送入FaceMobileNet模块进行人脸识别。

第二部分: 基于FaceMobileNet的人脸识别

1. **模型架构**
2. Flatten()

class Flatten(nn.Module):

    def forward(self, x):

        return x.view(x.shape[0], -1)

Flatten模块可将一个张量展平。它用于一系列的卷积操作之后，全连接层之前，可将四维空间平铺成二维空间。

1. 卷积层

为了对图像进行处理，我们需要构建卷积模块，包括ConvBn()、ConvBnPrelu()、DepthWise()、DepthWiseRes()、MultiDepthWiseRes()等等，其中，ConvBn()由一层卷积层和一层BN组成；ConvBnPrelu()在ConvBn()的基础上添加了Prelu激活层；DepthWise()利用ConvBn()和ConvBnPrelu()进行卷积操作，实现高效运算；DepthWiseRes()受ResNet启发，在DepthWise()返回结果的基础上加上了原始输入；而MultiDepthWiseRes()即为多个DepthWiseRes()的级联，由参数num\_block控制其层数。

1. FaceMobileNet

self.conv1 = ConvBnPrelu(1, 64, kernel=(3, 3), stride=2, padding=1)

self.conv2 = ConvBn(64, 64, kernel=(3, 3), stride=1, padding=1, groups=64)

self.conv3 = DepthWise(64, 64, kernel=(3, 3), stride=2, padding=1, groups=128)

self.conv4 = MultiDepthWiseRes(num\_block=4, channels=64, kernel=3, stride=1, padding=1, groups=128)

self.conv5 = DepthWise(64, 128, kernel=(3, 3), stride=2, padding=1, groups=256)

self.conv6 = MultiDepthWiseRes(num\_block=6, channels=128, kernel=(3, 3), stride=1, padding=1, groups=256)

self.conv7 = DepthWise(128, 128, kernel=(3, 3), stride=2, padding=1, groups=512)

self.conv8 = MultiDepthWiseRes(num\_block=2, channels=128, kernel=(3, 3), stride=1, padding=1, groups=256)

self.conv9 = ConvBnPrelu(128, 512, kernel=(1, 1))

self.conv10 = ConvBn(512, 512, groups=512, kernel=(7, 7))

self.flatten = Flatten()

self.linear = nn.Linear(2048, embedding\_size, bias=False)

self.bn = nn.BatchNorm1d(embedding\_size)

基于前面定义的六种网络模块，可以搭建出FaceMobileNet网络，其结构的代码实现如上所示。我们首先堆叠了10种不同的卷积模块，然后使用Flatten模块将输入展平，最后接一个全连接层和1维的BatchNorm层。在全连接层的代码中：

self.linear = nn.Linear(2048, embedding\_size, bias=False)

输入的数据维度为1 x 128 x 128，经过多层卷积之后，其变成512 x 2 x 2，也就是2048。此外，参数embedding\_size由外部传入，它表示用多大的向量来表示一张人脸。FaceNet使用了128维的向量来表征一张人脸，而我们这里将向量维度提高到512。提高人脸特征的存储维度，可以获得较好的存储性质，也能够很快地根据每个维度之间的差异实现人脸识别（判决）。

经过调研，我们发现FaceMobileNet网络拥有较小的体积，更少的计算量，更高的精度，是轻量级神经网络的典型代表。因此，为了更小的模型计算量，我们决定采用FaceMobileNet作为我们人脸识别的深度模型，在此基础上进行训练和应用的开发。在具体使用时，我们将人脸图像输入网络，计算其特征，再使用特征向量之间的余弦距离来估计其相似度，部分代码如下：

def cosin\_metric(x1, x2):

    x1 = np.squeeze(np.asarray(x1))

    x2 = np.squeeze(np.asarray(x2))

    return np.dot(x1, x2) / (np.linalg.norm(x1) \* np.linalg.norm(x2))

1. **Additional Margin Metric Loss函数**

后 Facenet 时代，研究员觉得三元组的选择过于麻烦，于是在原有的 Softmax Loss 基础上进行了改进，演绎出了SphereFace，CosFace，ArcFace等Additional Margin Metric Loss 。其目的是使模型的训练过程更加困难，通过这种困难磨砺模型，使其训练得更好。此处将对CosFace进行一个简单的介绍，其实现代码如下：

class CosFace(nn.Module):

    def \_\_init\_\_(self, in\_features, out\_features, s=30.0, m=0.40):

        super().\_\_init\_\_()

        self.in\_features = in\_features

        self.out\_features = out\_features

        self.s = s

        self.m = m

        self.weight = nn.Parameter(torch.FloatTensor(out\_features, in\_features))

        nn.init.xavier\_uniform\_(self.weight)

    def forward(self, input, label):

        cosine = F.linear(F.normalize(input), F.normalize(self.weight))

        phi = cosine - self.m

        output = cosine \* 1.0  # make backward works

        batch\_size = len(output)

        output[range(batch\_size), label] = phi[range(batch\_size), label]

        return output \* self.s

经过CosFace训练后，不同的类别之间会有一个额外的差距，即所谓的 Additional Margin。 SphereFace，CosFace，ArcFace的差别只在于此 margin 的位置。

总而言之，在训练过程中，CosFace完成了以下工作：

·对backbone网络的输出，也就是embedding进行L2规范化；

·将CosFace度量函数的权重进行L2规范化，再与embedding线性相乘，得到其cosine值；

·对正确标签的输出进行强化，也就是减小概率值；

·对强化后的cosine进行放大，以使后续的反向传播可以工作。

1. **Focal Loss函数**

当然，除了使用Additional Margin Metric Loss，我们也使用了Focal Loss对网络进行约束，其可以有效应对负样本loss值主导整个梯度下降，正样本占比小，导致模型只专注学习负样本的情况。相关代码实现如下：

class FocalLoss(nn.Module):

    def \_\_init\_\_(self, gamma=2):

        super().\_\_init\_\_()

        self.gamma = gamma

        self.ce = torch.nn.CrossEntropyLoss()

    def forward(self, input, target):

        logp = self.ce(input, target)

        p = torch.exp(-logp)

        loss = (1 - p) \*\* self.gamma \* logp

        return loss.mean()

整个深度模型的训练过程可以概括为以下几个步骤：

·初始化，包括数据的加载、优化器、网络的初始化等；

·将图像输入网络network，得到512维的特征向量；再将此特征向量输入metric，得到预测的标签，进而计算网络预测结果与ground truth之间的Focal Loss；

·反向传播，优化网络权重

·重复上述步骤，直到预设的训练轮数epoch

1. **性能指标**
2. 特征提取维度及时间

由(一)中分析可知，我们所采用模型的特征提取维度为512，即每张图像输入人脸识别网络后，将会得到和它相对应的一个512维特征向量。在LFW测试集进行测试的同时，统计特征提取所用时间，得到结果如下：



即平均每提取一张人脸的特征，需要花费秒。

1. LFW数据集上的识别率

在LFW测试集的6000对图像上进行测试，得到此模型的识别率为0.96767，如下：



第三部分：主函数与文件结构

1. **主函数（文件recognition.py）**

def recognize(is\_saving, is\_1N, is\_11, frame, identity, search\_identity):

主要参数由is\_saving（代表数据录入模式），is\_1N（1:N人脸识别模式），is\_11（1:1身份验证模式）确定，identity与search\_identity分别代表数据录入模式中以及身份验证模式的名字输入。输入frame为经MTCNN检测并水平矫正后裁剪得到的人脸图像。算法的基本流程如下：

·首次调用MTCNN，对摄像头截取的图像进行检测，得到人脸关键点，将两眼坐标传入wrap\_affine函数进行水平矫正与图像适应；

·再次调用MTCNN检测水平矫正后得到的图像rot\_frame，得到裁剪后的人脸fin\_frame。

·若is\_saving=1，则进入“数据录入模式”：利用featurize函数，调用训练好的FaceMobileNet对人像进行特征提取，得到fearture。将输入的identity与feature组成键值对，添加到字典当中并保存。

def featurize(frame, transform, network, device, is\_saving, identity=None):

    frame = transform(frame)[:, None, :, :]

    frame = frame.to(device)

    network = network.to(device)

    with torch.no\_grad():

        feature = network(frame)

    if is\_saving:

        ret = {identity: feature}

    else:

        ret = feature

    return ret

·当然，若希望重新录入（如希望重拍照片、修改名字），可以调用del\_dict函数，其将从保存的字典中删除该键值对：

def del\_dict(del\_identity):

    device = 'cuda' if torch.cuda.is\_available() else 'cpu'

    feature\_dir = './features/feature\_dict.pth'

    if os.path.exists(feature\_dir):

        feature\_dict = torch.load(feature\_dir, map\_location=device)

    else:

        feature\_dict = {}

    del feature\_dict[del\_identity]

    torch.save(feature\_dict, feature\_dir)

    return 'Deleted successfully!'

·若is\_saving!=1，is\_1N=1，则进入“人脸识别模式”：计算此帧人脸的特征向量与数据库（字典）中特征之间的余弦相似度，找出最大相似度所对应的identity，即为该人的识别结果：注意，当进行下一次识别时，前一帧提取的特征cur\_feature将会被舍弃：

cur\_feature = featurize(

fin\_pil, transfrom, network, device, is\_saving)

feature1 = cur\_feature.cpu().numpy()

for identity in feature\_dict.keys():

feature2 = feature\_dict[identity].cpu().numpy()

similarity = cosin\_metric(feature1, feature2)

similarities.update({identity: similarity})

max\_identity = max(similarities.items(),

                   key=operator.itemgetter(1))[0]

return ('Identity is %s, and the similarity is %f' %

         (max\_identity, similarities[max\_identity]))

·若is\_saving!=1，is\_11=1，则进入“身份验证模式”：若cur\_feature与数据库中该identity对应feature之间的余弦相似度超过0.66，则认为两者为同一个人。

PS：1、0.66的阈值设置有待考量；

2、前面提过，我们发现水平矫正对于这一similarity的区分度有很大的优化作用！

·如果希望查看所有录入人的名字信息，可调用person\_list函数：

def person\_list():

    device = 'cuda' if torch.cuda.is\_available() else 'cpu'

    feature\_dir = './features/feature\_dict.pth'

    identities = []

    if os.path.exists(feature\_dir):

        feature\_dict = torch.load(feature\_dir, map\_location=device)

    else:

        feature\_dict = {}

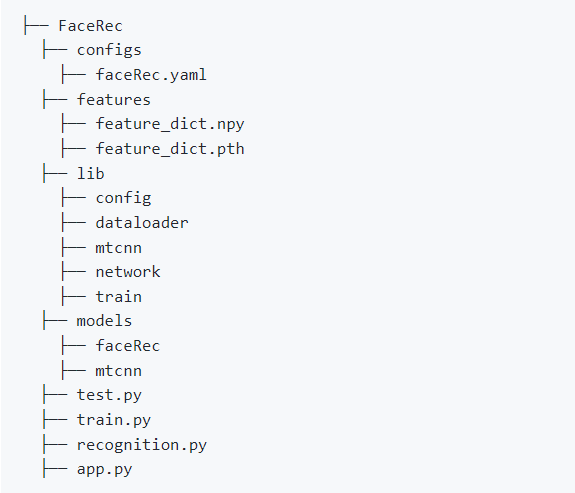
    for key in feature\_dict.keys():

        identities.append(key)

    return identities

1. **文件结构**

此项目的文件结构和说明如下：





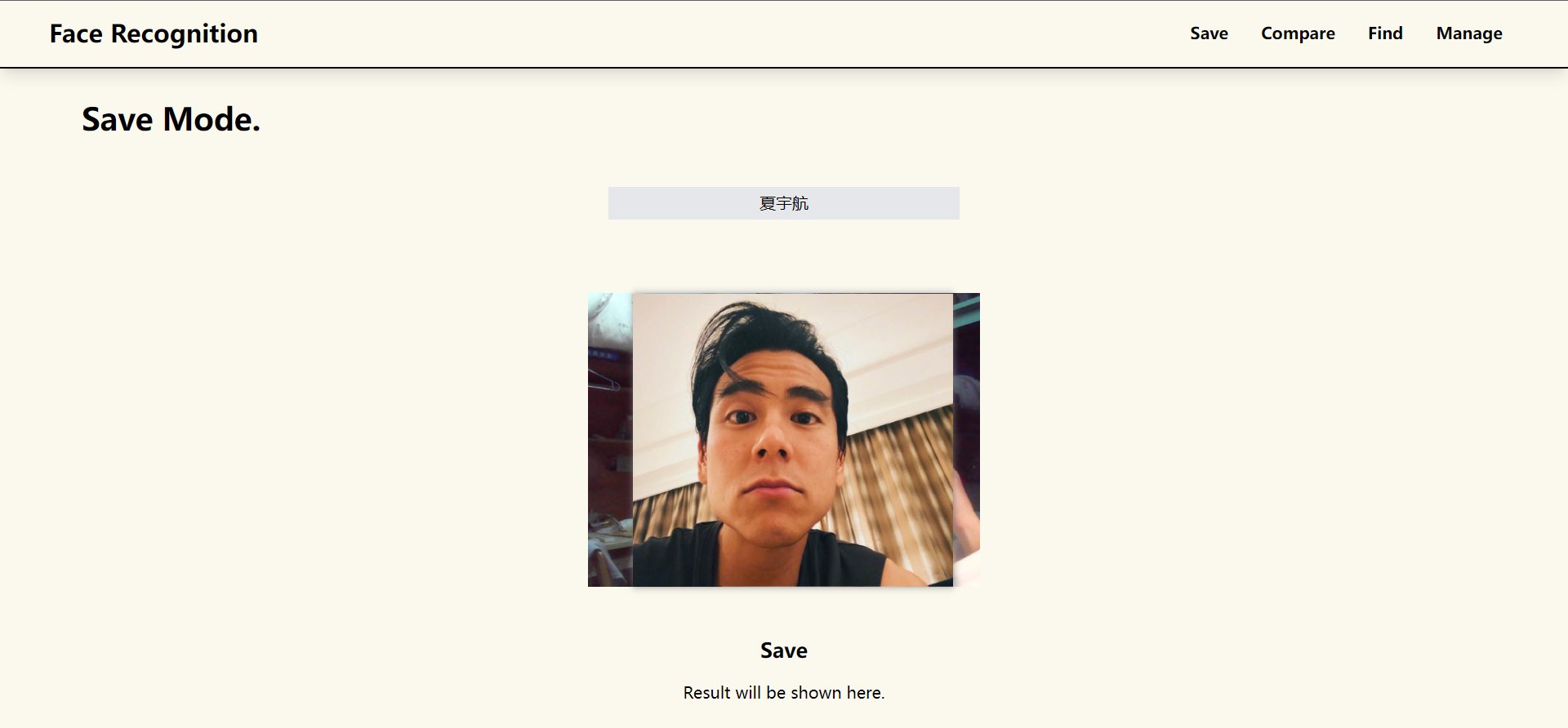
第四部分：接口封装与前后端构建

1. **数据录入**

·输入：名字、拍摄的照片。

·输出：是否成功录入。

·原理：通过MTCNN检测人脸，并将两眼连线旋转到水平，再通过已经训练好的FaceMobileNet模型得到512维特征的数据，将这个数组存放在feature\_dict.pth中，对应字典结构为：{identity：feature}，即完成了人脸数据的录入。

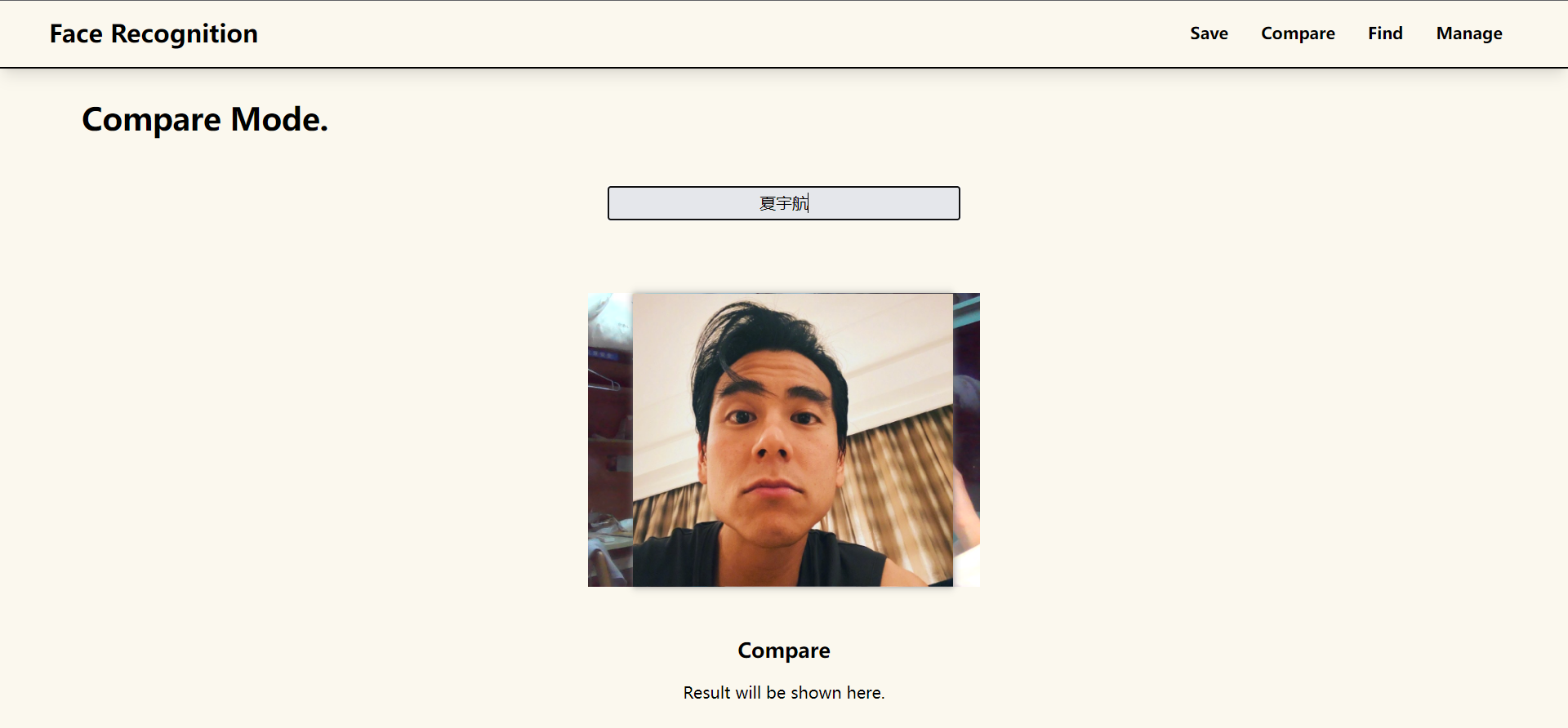


1. **人脸验证（1:1）**

·输入：拍摄照片、对应数据库中的名字。

·输出：是否为同一人。

·原理：首先计算临时所摄照片的512维特征向量，再查找feature\_dict.npy中对应名字的512维特征，计算余弦相似度，若超过阈值则认为是同一个人。

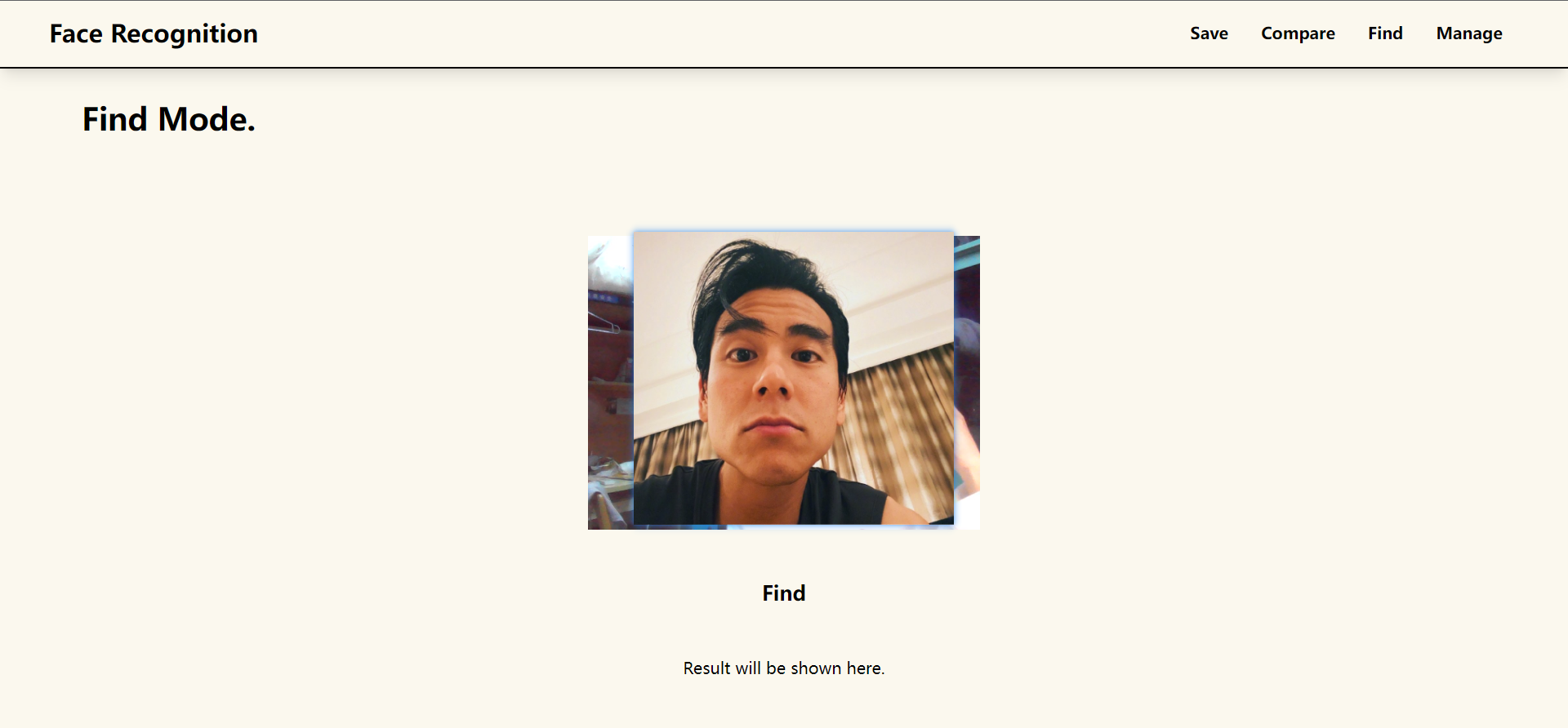


1. **人脸识别（1:N）**

·输入：拍摄照片。

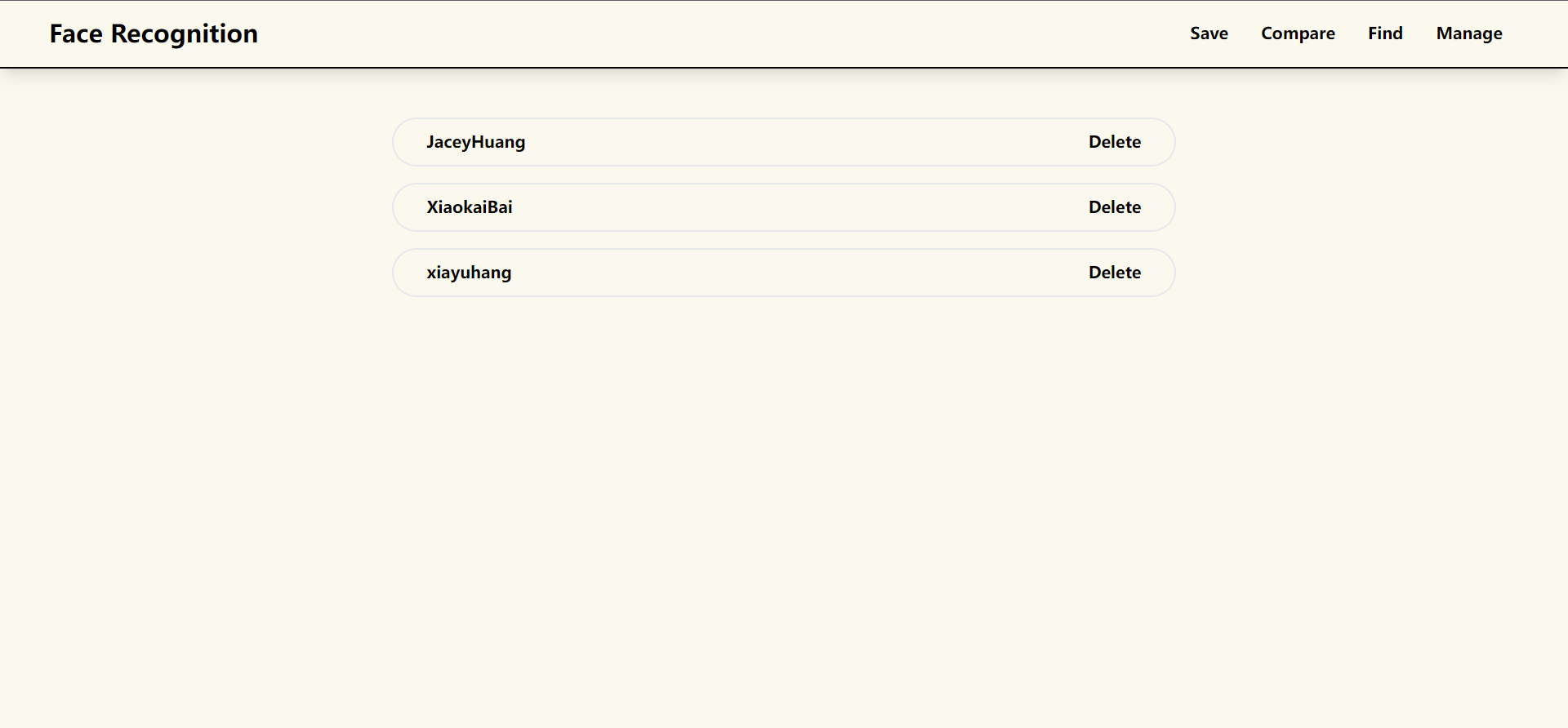
·输出：对应数据库中的名字。

·原理：首先计算临时所拍照片的512维特征，与feature\_dict.npy中所有的数据进行相似度计算，选取相似度最高的identity，作为该照片对应的身份（也可设置阈值，当相似度小于阈值时输出未能识别该照片所对应的身份）。



1. **数据管理**

在Manage页面可以看到已经录入的所有数据，并可以删除已经录入的数据：



1. **前端框架**

·前端部分使用 React 框架搭建，样式部分采用 tailwind 编写；

·相机部分调用 Web Camera，点击 Compare 或者 Save 时就截取当前帧，转化为 blob 格式的数据传递给后端；

·使用路由的结构来分割应用，可在手机或电脑上通过网页访问界面。

export default function App() {

  return (

    <div className="w-full h-full font-sans bg-primary-brown">

      <Toaster />

      <BrowserRouter>

        <Routes>

          <Route path="/" element={<Home />} />

          <Route path="/save" element={<Save />} />

          <Route path="/compare" element={<Compare />} />

          <Route path="/find" element={<Find />} />

          <Route path="/manage" element={<Manage />} />

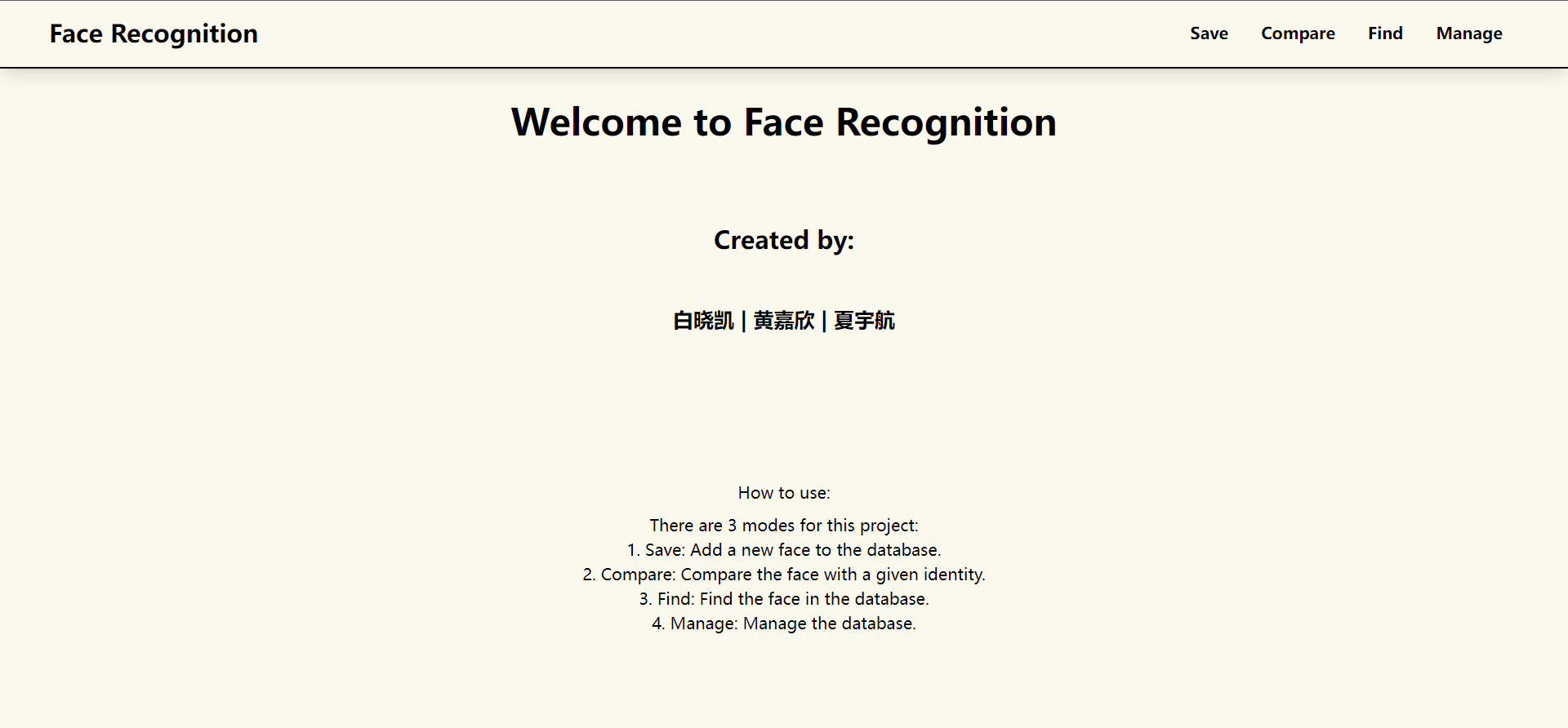
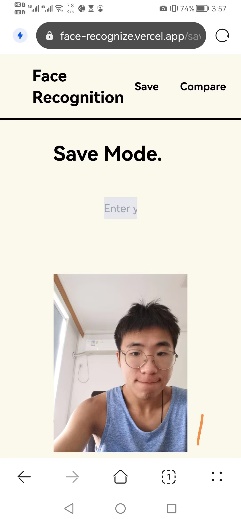
        </Routes>

      </BrowserRouter>

    </div>

  );

}

1. **后端框架**

·后端采用的是 flask 框架，挂载 python 脚本 app.py 到本地的81端口，监听前端传递来的数据和请求，做出相应的回应；

·Save、Compare和Find都传递了拍摄的照片，采用POST请求方法；

·获取数据使用GET请求方法，删除则是DELETE；

·采用CORS跨域资源代理来解决跨域问题。

@app.route("/getRecognition", methods=['POST', 'GET', 'DELETE'])

async def get\_recognition():

    response = await handle\_pic\_get\_recognition(request)

    return response

async def handle\_pic\_get\_recognition(request):

    try:

        if request.method == 'POST':

            # decide mode

            mode = request.form['mode']

            res = ''

            print(request.form)

            if mode == 'delete':

                identity = request.form['identity']

                res = rec.del\_dict(identity)

            else:

                # get the image from the request

                f = request.files['file']

                img = cv2.imdecode(np.frombuffer(

                    f.stream.read(), np.uint8), cv2.IMREAD\_COLOR)

                if mode == 'find':

                    res = rec.recognize(False, True, False, img, '', '')

                elif mode == 'save':

                    identity = request.form['identity']

                    res = rec.recognize(True, False, False, img, identity, '')

                elif mode == 'compare':

                    search\_identity = request.form['searchIdentity']

                    res = rec.recognize(False, False, True,

                                        img, '', search\_identity)

        elif request.method == 'GET':

            res = rec.person\_list()

    # if there is an error, return a 400 bad request

    except Exception as e:

        print(e)

        return make\_response(jsonify({'data': None, 'status': 0, 'error': 'Something is wrong'}), 0)

    # return the result as json

    response = jsonify({'data': res, 'status': 200, 'error': None})

    response.headers['Access-Control-Allow-Origin'] = '\*'

    response.headers['Access-Control-Allow-Methods'] = 'POST'

    response.headers['Access-Control-Allow-Headers'] = 'x-requested-with,content-type'

    return make\_response(response, 200)

app.run(host='0.0.0.0', port=81)

第五部分: 本系统的创新点与优势

1. **人脸检测**

我们调用了两次MTCNN，首次MTCNN用于检测基本的人脸位置（主要是双眼位置的检测），然后通过仿射变换对图像进行调整矫正，考虑到仿射变换会增加大量无效信息，我们还对图像进行了自适应大小调节；接着再次调用MTCNN，得到水平的人脸图像。实验证明，经过水平矫正和大小调整之后的人脸识别准确精度大大提升，也会避免在调用bounding\_box时下标越界的情况，大大地提高了整个人脸检测、识别系统的稳定性。

1. **人脸识别**

我们采用了轻量级的FaceMobileNet神经网络，其具有较小的参数量和较小的模型体积，从而获得了更小的计算量、更高的准确率和更快的速度，适合多种场景应用且识别效果较好，具有十分广阔的发展空间。

1. **使用体验**

我们采用Web应用的形式来代替传统的qt软件，界面上更加美观，交互也更顺畅自然。编写了安装和启动的脚本，便于用户使用。具体使用过程可参见演示视频。

第六部分：总结与收获

在本次大作业中，我们首先完成了基于MTCNN的人脸检测工作，接着对老师提供的轻量化人脸识别网络FaceMobileNet进行优化改进，最后还搭建了美观实用的UI界面，对于业界人脸识别的基本方式有了较为深入的理解与认识。

总的来说，这次人脸识别大作业让我们收益匪浅，从学习pytorch语法到学习MTCNN与一般的人脸识别网络，再到搭建与实现UI界面，我们对于pytorch与神经网络有了更加深刻的理解。非常感谢老师给我们这次机会，我们受益匪浅！谢谢老师！

分工以及具体贡献如下：

·白骁凯：33.3%-MTCNN人脸检测，水平矫正与大小自适应，模型优化。

·黄嘉欣：33.3%-face\_rec主体框架与网络搭建，人脸识别模型优化，工程实现。

·夏宇航：33.3%-前后端框架的构建，接口的封装。