浙江大学

《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.

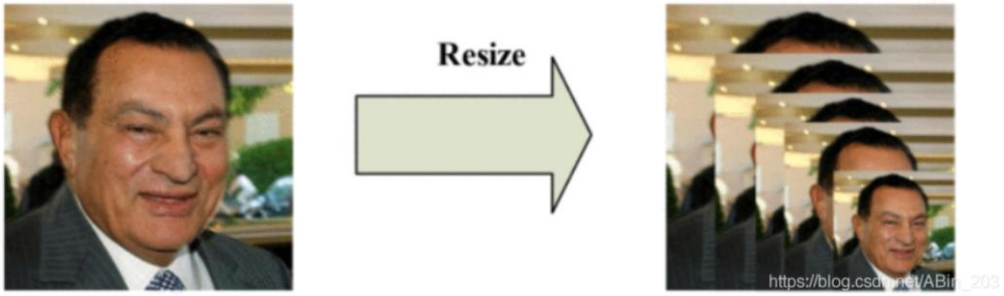
Chinese Library Classification Number：TU 000 Document Code：A Article Number：000-000(0000) 00-000-00

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。

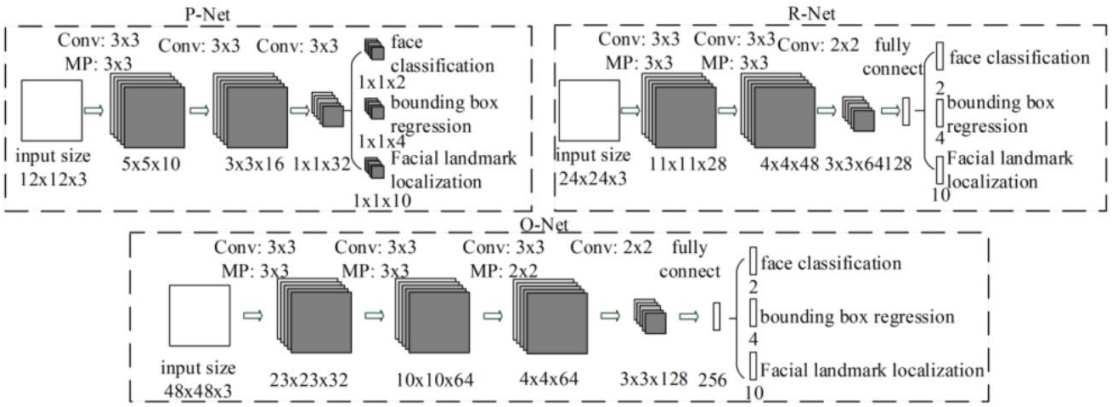
一、 图像金字塔



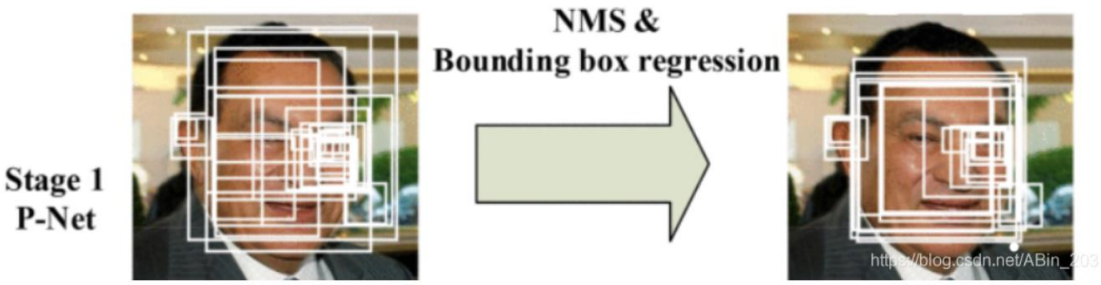
我们可以看到左图通过不同的缩放系数 factor 对图片进行缩放，每次将其缩小为原来的 factor 大小，从而得到右侧的一系列“图像金字塔”。在图中，取 factor = 0.709 ≈ sqrt(2)/2，此时缩放后图像的宽高变为原来的 sqrt(2)/2，面积变为原来的 1/2。通过处理后，我们可以生成更多的原样本，在进行识别时就会有更多数据、更加准确，但同时也会带来处理时间太长等缺点。

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。

二、 P-net



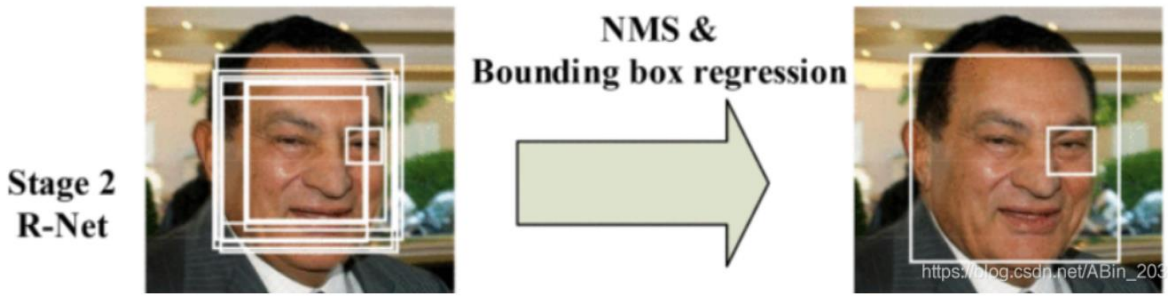
P-net 输入维度: (12,12,3)，其需要判断该 12*12*3 的图像中是否有人脸，并给出人脸框和关键点的位置。网络的第一部分输出用来判断该图像是否存在人脸，输出向量大小 1×1×2，也就是两个值；网络的第二部分给出框的精确位置，一般称为框回归；网络的第三部分给出人脸的 5 个关键点的位置，即左眼、右眼、鼻子、嘴角左侧、嘴角右侧。每个关键点需要使用两维来表示，因此输出向量大小为 1×1×10。



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。

三、 R-net

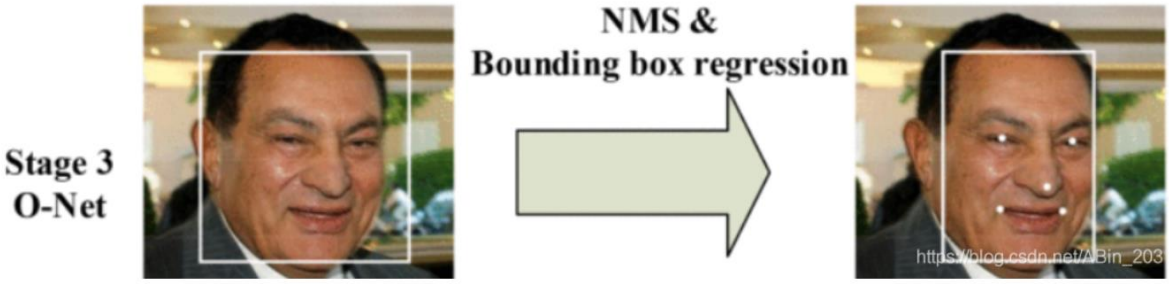
P-Net 的所有候选框都将被输入到 R-Net 中，该网络结构通过边界框回归和 NMS 来滤除大量的 false-positive 区域。从网络图可以看到，R-net 网络结构与 P-Net 存在一定差异，其多出一个全连接层，所以能够取得更好的抑制 false-positive 的效果。在将数据输入 R-Net 之前，需要将其维度缩放到 24x24x3，经过处理后，R-Net 将会去除大量的非人脸框，得到与 P-net 类型相同的三种输出结果。



四、 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。



五、 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

一、 模型架构

1、 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。

2、 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。

3、 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))
```

二、 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。

三、 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

四、 性能指标

1、 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：

```
The time every picture cost on average is: 0.0002746548851331075s
```

即平均每提取一张人脸的特征，需要花费 2.7×10^{-3} 秒。

2、 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：

```
The accuracy on the test set is: 0.9676666666666667
```

第三部分：主函数与文件结构

一、 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
```

二、 文件结构

此项目的文件结构和说明如下：

```
├── FaceRec
  ├── configs
    ├── faceRec.yaml
  ├── features
    ├── feature_dict.npy
    ├── feature_dict.pth
  ├── lib
    ├── config
    ├── dataloader
    ├── mtcnn
    ├── network
    ├── train
  ├── models
    ├── faceRec
    ├── mtcnn
  ├── test.py
  ├── train.py
  ├── recognition.py
  ├── app.py
```

file	Intro
`configs`	config 文件，需要用到的一些全局变量可以在此处修改
`features`	保存的人脸特征字典，形式为 `{identity: feature}`
`lib`	训练需要用到的一些文件，如 `dataloader` 以及网络的结构等
`models`	网络模型的权重
`test.py`	在LFW上测试模型的正确率
`train.py`	训练代码
`recognition.py`	人脸检测与识别所需用到的函数
`app.py`	后端代码

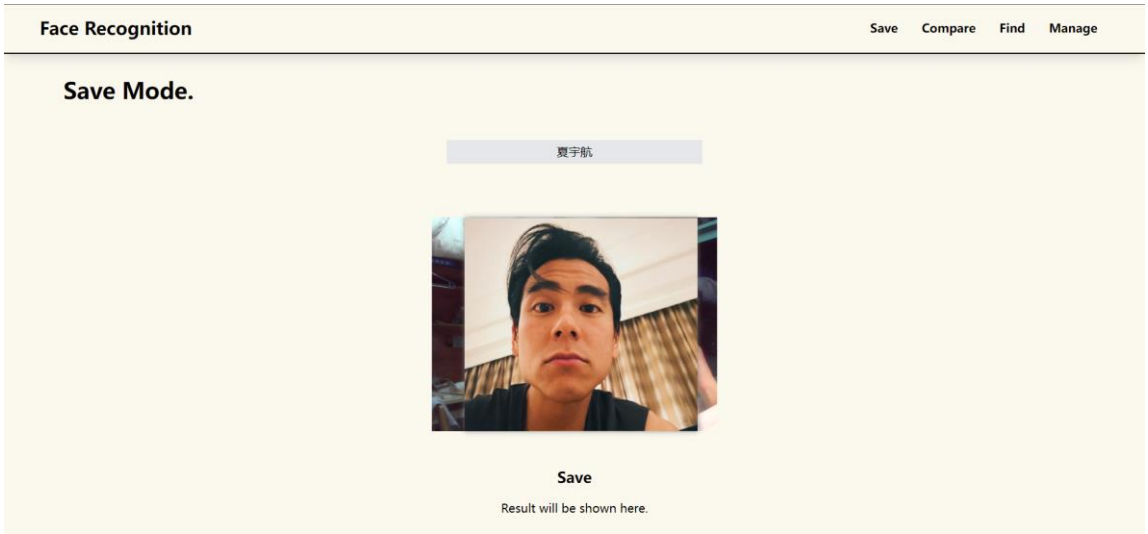
第四部分：接口封装与前后端构建

一、 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。

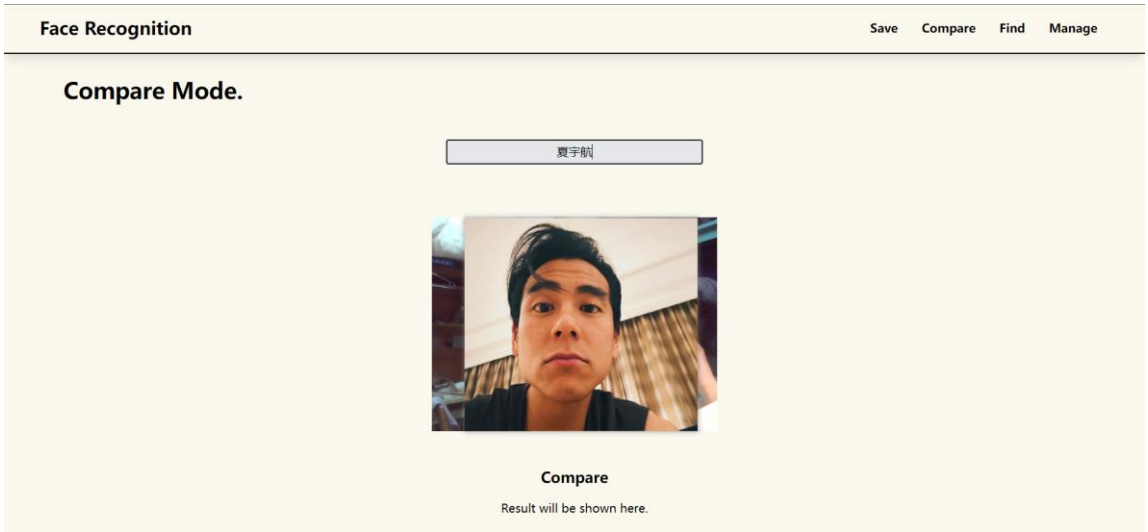


二、 人脸验证（1:1）

• 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。

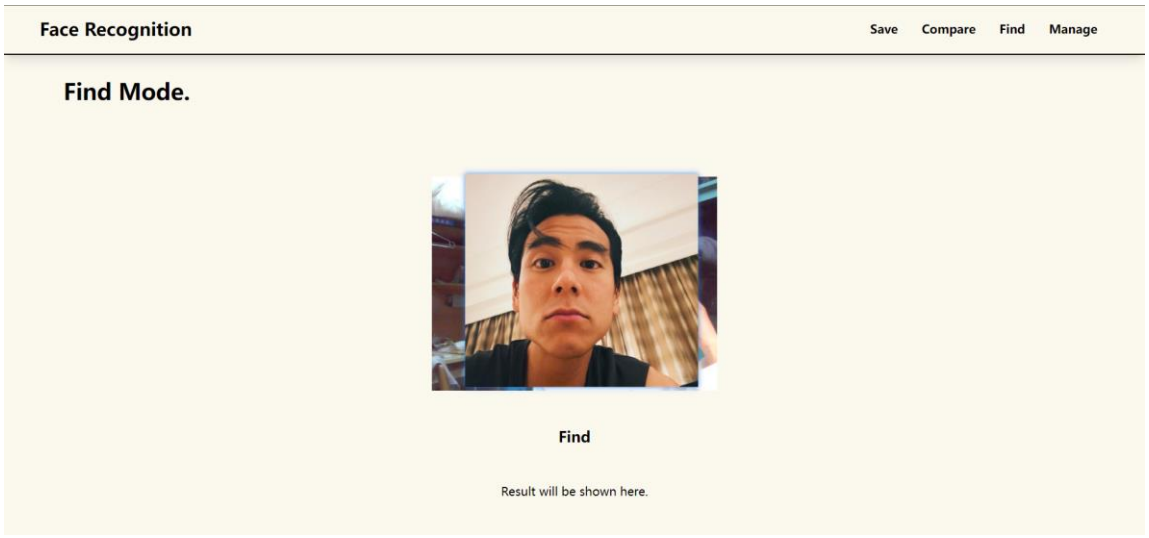


三、 人脸识别（1:N）

• 输入：拍摄照片。

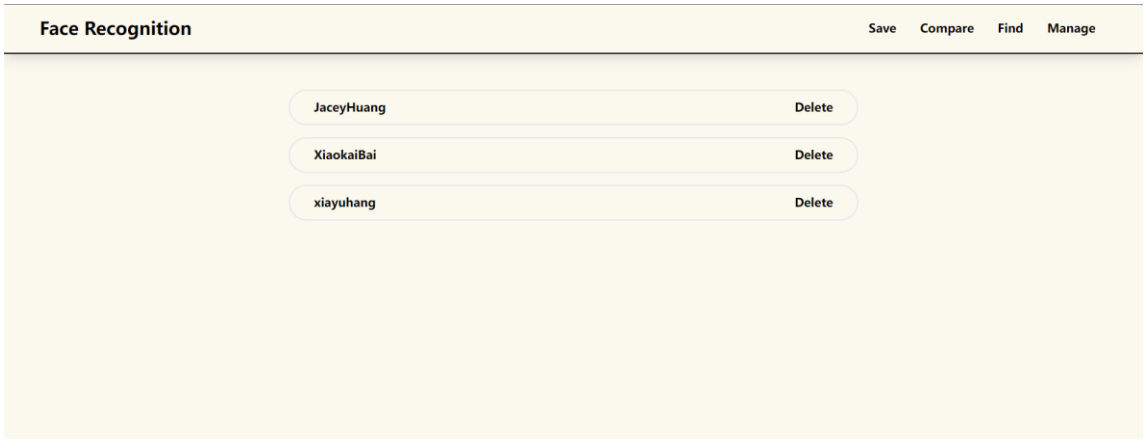
• 输出：对应数据库中的名字。

• 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）。



四、 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：



五、 前端框架

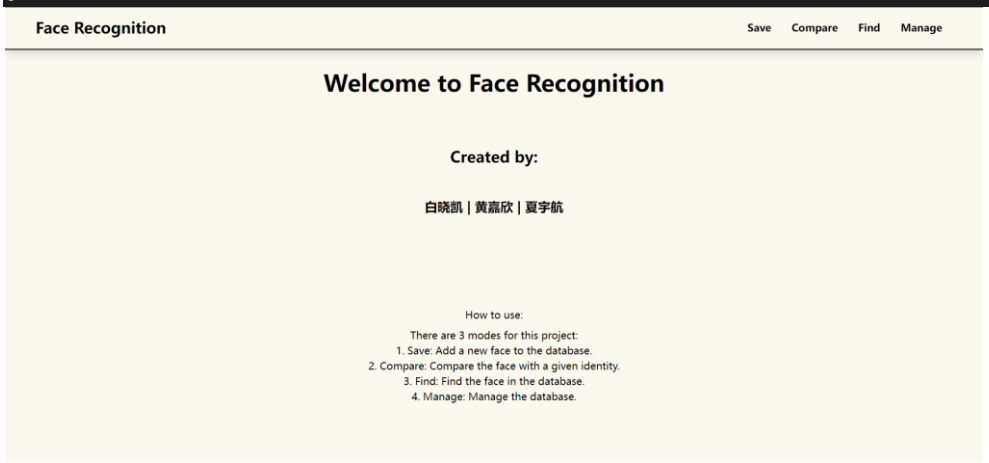
• 前端部分使用 React 框架搭建，样式部分采用 tailwind 编写；

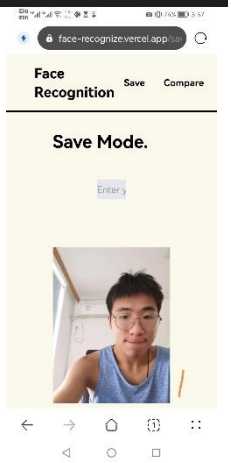
• 相机部分调用 Web Camera，点击 Compare 或者 Save 时就截取当前帧，转化为 blob 格式的数据传递给后端；

• 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>
  );
}
```





六、 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

一、 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。

二、 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。

三、 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。