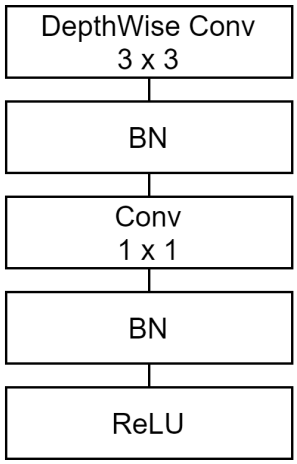
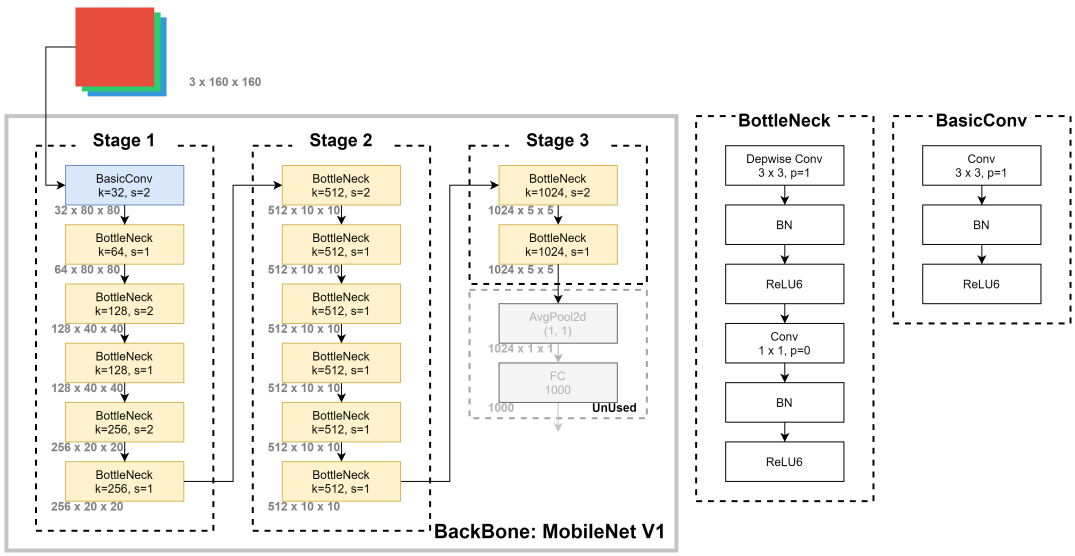
人脸识别

1. FaceNet人脸识别

- **识别步骤**：人脸检测（目标检测） → 利用坐标进行截取 → 人脸识别算法
- **概述**：
  - FaceNet作为人脸识别部分，从截取的人脸图片中抽取128维度的特征向量，作为输入人脸的特征向量
  - 同时具有一个人脸，里面存放了所有人的人脸特征向量（均是用FaceNet提取的128维度特征项向量）
  - 将当前输入的人脸特征向量和数据库里存放的人脸特征向量进行比对（计算欧式距离），最终筛选出距离最短，且大于阈值的数据库内特征向量的标签作为输出
- **主干网络**：
  - **作用**：facenet的主干网络起到提取特征的作用
  - **模型**：原版是以Inception-ResNetV1为主干特征提取网络，这里会使用mobilenetv1作为主干特征提取网络
  - **MobileNet V1**：更多详见ImageNet学习资料
    - **Depthwise Convolution （深度可分离卷积）**：每个输入通道，都有自己的卷积核，通道之间不共享卷积核，这样输入和输出的通道数保持不变
    - **bottleneck组成**：如下图



    - **MobileNet v1构架图**：



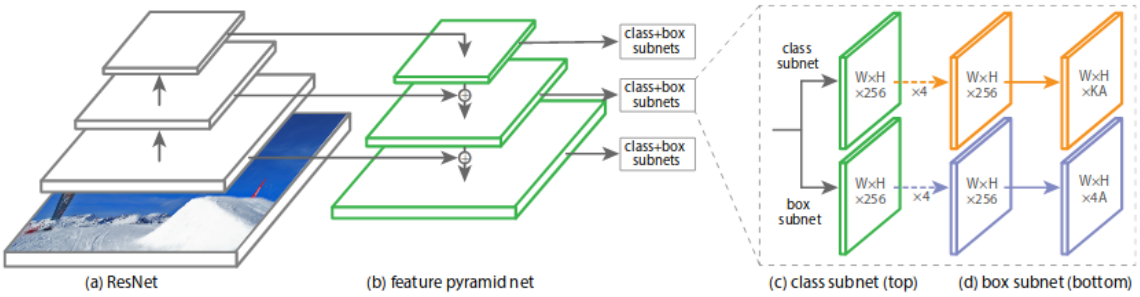
- **向量抽取步骤**：
  1. **输入**：首先是从MobileNet v1的骨干网络抽取出$1024 \times 5 \times 5$的特征图
  2. **拉平**：通过AdaptiveAvgPool2d方法拉平网络为（$1024 \times 1 \times 1$）
  3. **dropout**
  4. **全连接层**：由1024转为128维的embedding输出（128）
  5. **BatchNorm1d**
  6. **L2标准化**：使得不同人脸的特征向量可以属于同一数量级，方便比较
     - **L2-范数**：$||x||_2 = \sqrt{\sum_{i=1}^{N} x_i^2}$，也就是欧几里得范数
     - **L2标准化**：每个元素除以L2范数
     - **pytorch实现**：`F.normalize(x, p=2, dim=1)`

- **构建分类器**：
  - **作用**：采用了交叉熵损失用于辅助Triplet Loss（下面会说）的收敛（因为Triplet Loss会对不同人脸特征距离进行扩张，同一人不同状态的脸差距缩小，从而导致收敛困难）
  - **使用方式**：
    - 在训练时使用分类器（Cross-Entropy Loss）辅助训练，预测时则不需要
    - 使用方式就是在**L2标准化结束前**的128维度向量后加一层全连接层（神经元个数是num_classes），再接一个交叉熵损失
- **训练过程**：
  - **损失函数**：Triplet Loss作为loss
    - **输入**：输入是一个三元组
      - **a**：anchor，基准图片获得的128维人脸特征向量
      - **p**：positive，与基准图片属于同一张人脸的图片获得的128维人脸特征向量
      - **n**：negative，与基准图片不属于同一张人脸的图片获得的128维人脸特征向量
    - **原理**：
      - 将anchor和positive求欧几里得距离，并使其尽量小，同一个人的不同状态的人脸特征向量欧几里得距离小
      - 将negative和positive求欧几里得距离，并使其尽量大，不同人的人脸特征向量欧几里得距离大
    - **公式**：$L = max(d(a,p) - d(a,n) + margin, 0)$
      - $d(a,p)$：anchor和positive的欧几里得距离，前面为正符号，所以我们期望其越来越小
      - $d(a,n)$：negative和positive的欧几里得距离，前面为负符号，所以我们期望其越来越大
      - $margin$：常数
    - **步骤**：
      - 首先计算anchor和positive的距离dist（`pos_dist`），以及anchor和negative的距离dist（`neg_dist`）
      - 如果`neg_dist - pos_dist`小于`alpha`才会保留该样本来计算损失（`alpha=0.2`，就是margin）—— `hard_triplets`
      - 基础损失值计算`basic_loss = pos_dist - neg_dist + alpha`
      - 损失值：`sum(basic_loss) / max(1, len(hard_triplets[0]))`
    - **辅助收敛**：
      - **Triplet Loss**：用于进行不同人的人脸特征向量欧几里得距离的扩张，同一个人的不同状态的人脸特征向量欧几里得距离的缩小
      - **Cross-Entropy Loss**：用于人脸分类，具体作用是辅助Triplet Loss收敛

        - **损失值**：
            - Triplet Loss输入值，在L2标准化之前的128维的特征输入Triplet Loss做损失
            - Cross Entropy输入值，在L2标准化之后，再接着分类全连接为分类的logits，最后对其做交叉上损失
            - Loss = Triplet loss + Cross Entropy loss，做返回迭代
    - **评估**：lfw文件中包含了每个人的一张照片，最后将预测的照片和数据中所有的照片的特征进行比对，这里比对的是L2标准化之后的128维向量进行距离计算
- **预测过程**：
    - **预处理**：
        - **Letter box**：因为FaceNet输入的图片都是正方形，普通的resize会导致画面失真，这里通过以图片的最长边调整到标准输入的尺寸，并再另一条边的上下方平均的不上灰条（图示详见目标检测）
        - **归一化**：所有维度除以255，并转化成float64
    - **预测**：将两张图片传入网络后，求两个输出的欧式距离，就得到了两张照片的距离
        - 预测的是L2标准化之后的128维向量进行距离计算

2. RetinaFace + FaceNet人脸识别：

- **人脸数据库的初始化**：
    - **目的**：要实现人脸识别，首先要知道自己需要识别哪些人脸，需要将人脸进行编码并放入数据库中
    - **格式**：数据库中每一张图片对应一个人的人脸，图片名字中"_"靠左的部分就是这个人的名字
    - **执行步骤**：
        - 遍历数据库中所有的图片
        - 利用Retinaface检测每个图片中的人脸位置
        - 将人脸截取下来
        - 将获取到的人脸进行对齐
        - 利用Facenet将人脸进行编码
        - 将所有人脸编码的结果放在一个列表中
        - 保存成npy的形式
- **RetinaFace人脸目标识别**：
    - **介绍**：
        - RetinaFace是Insight Face在2019年提出的one-stage人脸检测模型，原模型使用了deformable 、convolution和dense regression loss， 在 WiderFace 数据集上达到 SOTA
        - 骨干网络包括ResNet50、ResNet152以及轻量版的mnet（mobilenet0.25）
    - **简化版mnet结构**：
        - **FPN模块**：
            - RetinaFace的mnet本质是基于RetinaNet的结构，采用了特征金字塔的技术，实现了多尺度信息的融合，对检测小物体有重要的作用。
            - 简化版的mnet与RetinaNet采用了相同的proposal策略，即保留了在FPN的3层特征图每一层检测框分别proposal，生成3个不同尺度上的检测框，每个尺度上又引入了不同尺寸的anchor大小，保证可以检测到不同大小的物体。



- **SSH检测模块**：
  - mnet使用了SSH检测网络的检测模块，SSH检测模块由SSH上下文模块组成
  - **SSH上下文模块**：上下文模块的作用是扩张预检测区域的上下文信息（下图一）
  - **SSH网络的检测模块**：将一个上下文模块与conv叠加后生成分类头和回归头得到网络的输出

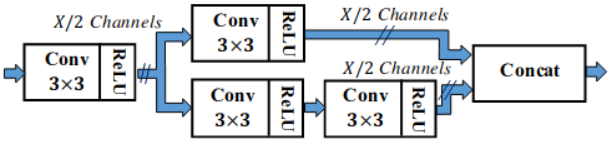


Figure 4: *SSH* context module.

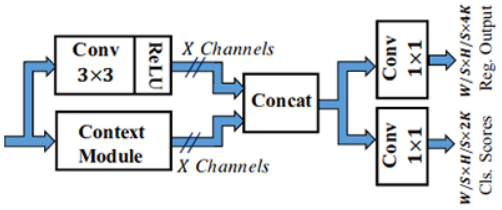


Figure 3: *SSH* detection module.

- **预测输出**：
  - mnet网络在使用SSH检测模块的同时实现了多任务学习，即在分类和回归的基础上加入了目标点(landmark点)的回归
  - 官方的网络结构采用了5个目标点的学习，后续也可以修改为更多目标点（比如AFLW中的21个目标点以及常用的68或者106个目标点）
  - 本次案例中只采用5个点，不包含下图中Self-supervision的部分
  - **这是该模型最大创新**，除了传统的人脸分类损失函数和包围框回归损失函数，作者**额外标注了人脸 5 点信息**，并以此引入人脸对齐的额外监督信息损失函数，还引入了self-supervised解码分支预测3D人脸信息分支
  - 回归部分的损失计算用的都是smooth L1，和faster rcnn相同
  - label中的只有正例将参与计算回归损失，其余将不参与

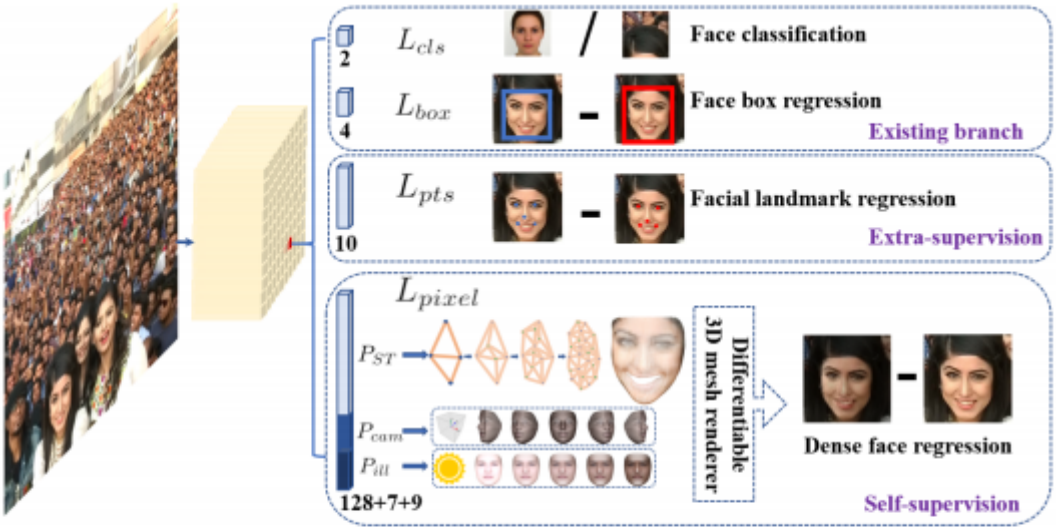
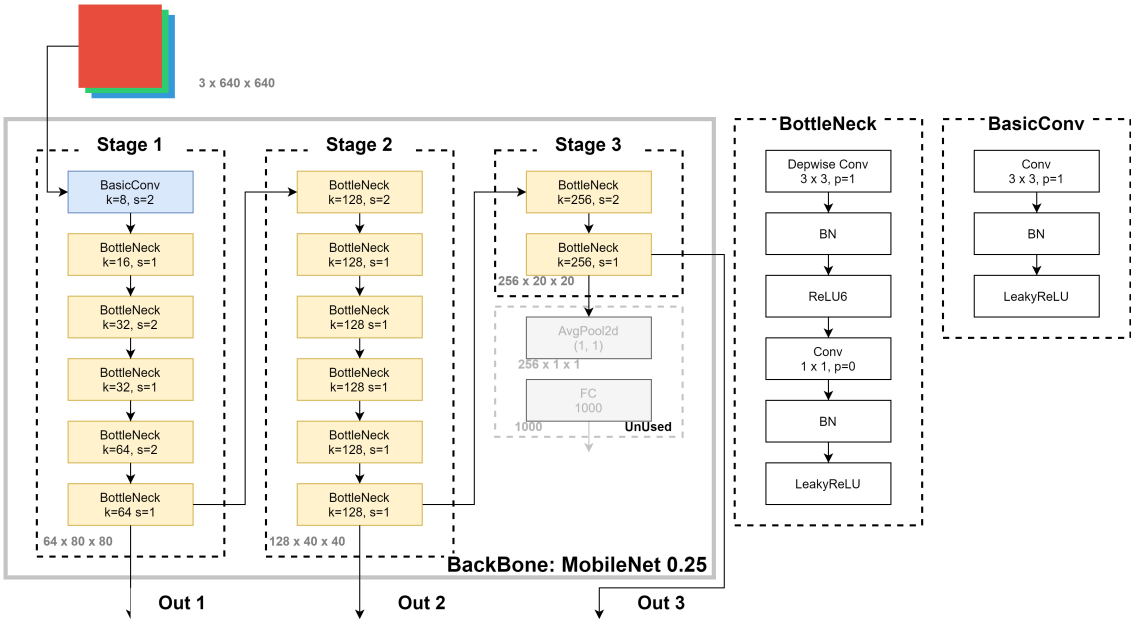
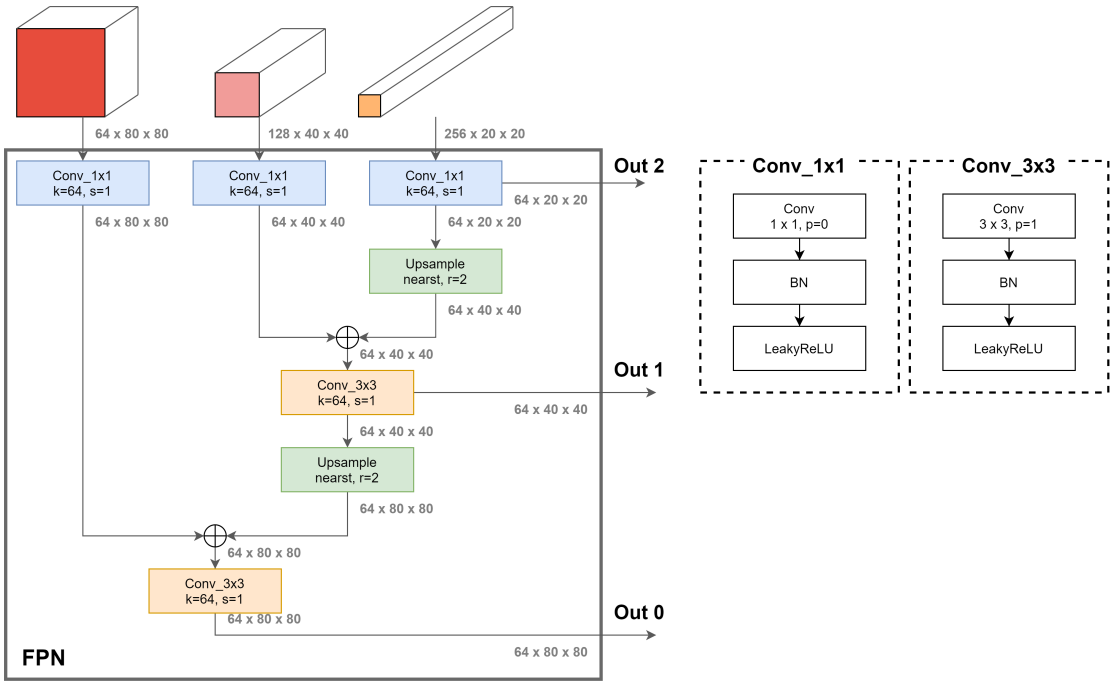


Figure 1. The proposed single-stage pixel-wise face localisation method employs extra-supervised and self-supervised multi-task learning in parallel with the existing box classification and regression branches. Each positive anchor outputs (1) a face score, (2) a face box, (3) five facial landmarks, and (4) dense 3D face vertices projected on the image plane.

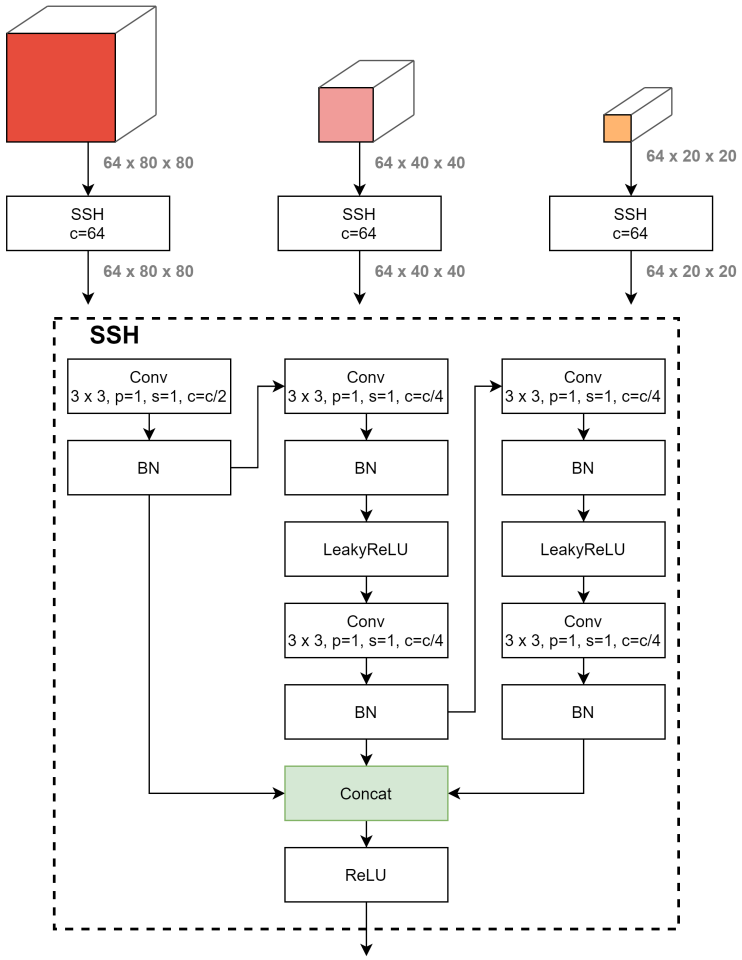
- **超参数设置**:
  - **threshold**：分类概率的阈值，超过这个阈值的检测被判定为正例
  - **nms_threshold**：非极大值抑制中的IOU阈值，即在nms中与正例的IOU超过这个阈值的检测将被舍弃
  - **scale**：图像金字塔的缩放值，通过对原图进行由scale值指定的大小缩放得到网络图片的输入大小，注意在检测时网络的输入不必保持相同的大小
- **损失函数**:
  - **分类损失（cls）**：交叉熵损失，`loss_c = F.cross_entropy(conf_p, targets_weighted, reduction='sum')`
  - **坐标损失（bbox）**：Smooth L1损失，`loss_l = F.smooth_l1_loss(loc_p, loc_t, reduction='sum')`
  - **面部地标损失（landm）**：Smooth L1损失，`loss_landm = F.smooth_l1_loss(landm_p, landm_t, reduction='sum')`
- **代码框架**:
  - **骨干网络**：MobileNet V1（0.25），将三个stage输出


3 x 640 x 640
Stage 1
BasicConv
k=8, s=2
BottleNeck
k=16, s=1
BottleNeck
k=32, s=2
BottleNeck
k=32, s=1
BottleNeck
k=64, s=2
BottleNeck
k=64 s=1
64 x 80 x 80
Stage 2
BottleNeck
k=128, s=2
BottleNeck
k=128, s=1
BottleNeck
k=128 s=1
BottleNeck
k=128 s=1
BottleNeck
k=128, s=1
BottleNeck
k=128, s=1
128 x 40 x 40
Stage 3
BottleNeck
k=256, s=2
BottleNeck
k=256, s=1
256 x 20 x 20
AvgPool2d
(1, 1)
256 x 1 x 1
FC
1000
1000
UnUsed
BackBone: MobileNet 0.25
Out 1
Out 2
Out 3
BottleNeck
Depwise Conv
3 x 3, p=1
BN
ReLU6
Conv
1 x 1, p=0
BN
LeakyReLU
BasicConv
Conv
3 x 3, p=1
BN
LeakyReLU

- **FPN网络**:


64 x 80 x 80
128 x 40 x 40
256 x 20 x 20
Conv_1x1
k=64, s=1
Conv_1x1
k=64, s=1
Conv_1x1
k=64, s=1
Out 2
64 x 20 x 20
64 x 80 x 80
64 x 40 x 40
64 x 20 x 20
Upsample
nearst, r=2
64 x 40 x 40
64 x 40 x 40
Conv_3x3
k=64, s=1
Out 1
64 x 40 x 40
64 x 40 x 40
Upsample
nearst, r=2
64 x 80 x 80
64 x 80 x 80
Conv_3x3
k=64, s=1
64 x 80 x 80
Out 0
64 x 80 x 80
FPN
Conv_1x1
Conv
1 x 1, p=0
BN
LeakyReLU
Conv_3x3
Conv
3 x 3, p=1
BN
LeakyReLU

- **SSH网络**:



- **输出**：
  - **输出内容**：类别（0/1，人脸或其他，所以是$2 * num_anchor$）、BboxHead（人脸的外接框坐标，所以是$4 * num_anchor$）、LandmarkHead（人脸共5个点，所以是$5 * 2 * num_anchor$）
  - **维度转换**：三个输出都是先通过一层1×1，$stride = 1$的卷积转换成输出的维度

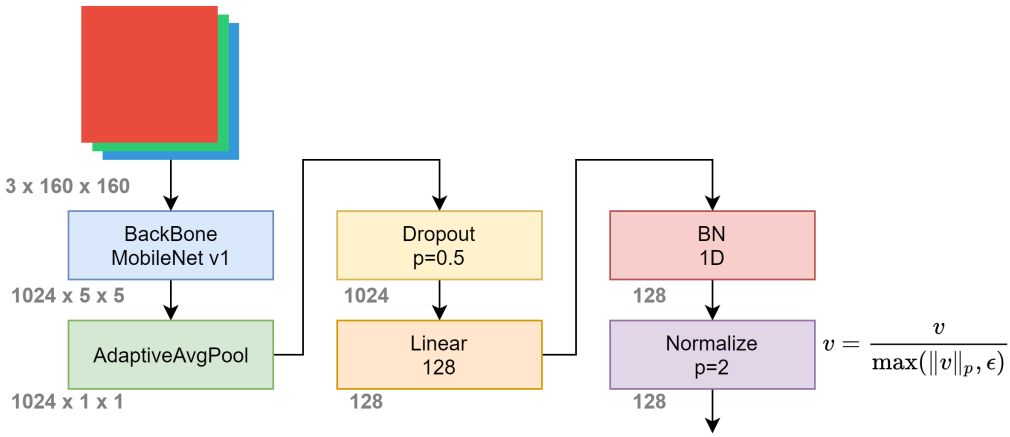
- **预测部分**：—— `predict.py`
  - **人脸的截取与对齐**：
    - **问题**：在人脸识别裁剪后，人脸的位置是外斜的，需要进行矫正，这对人脸的特征提取非常有好处，如下图所示

    - **双眼坐标对齐**：
      - **参数**：

        - 眼睛连线相对于水平线的倾斜角：了解到需要旋转的角度是多少（RetinaFace推理得出）
        - 图片的中心：了解到旋转的中心坐标（图片宽高计算可得）
    - **代码**：

```
#-------------------------------------#
#    人脸对齐
#-------------------------------------#
def Alignment_1(img,landmark):

    if landmark.shape[0]==68:
        x = landmark[36,0] - landmark[45,0]
        y = landmark[36,1] - landmark[45,1]
    elif landmark.shape[0]==5:
        x = landmark[0,0] - landmark[1,0]
        y = landmark[0,1] - landmark[1,1]
    # 眼睛连线相对于水平线的倾斜角
    if x==0:
        angle = 0
    else:
        # 计算它的弧度制
        angle = math.atan(y/x)*180/math.pi

    center = (img.shape[1]//2, img.shape[0]//2)

    RotationMatrix = cv2.getRotationMatrix2D(center, angle, 1)
    # 仿射函数
    new_img = cv2.warpAffine(img,RotationMatrix,
(img.shape[1],img.shape[0]))

    RotationMatrix = np.array(RotationMatrix)
    new_landmark = []
    for i in range(landmark.shape[0]):
        pts = []

 pts.append(RotationMatrix[0,0]*landmark[i,0]+RotationMatrix[0,1]*la
ndmark[i,1]+RotationMatrix[0,2])

 pts.append(RotationMatrix[1,0]*landmark[i,0]+RotationMatrix[1,1]*la
ndmark[i,1]+RotationMatrix[1,2])
        new_landmark.append(pts)

    new_landmark = np.array(new_landmark)

    return new_img, new_landmark
```

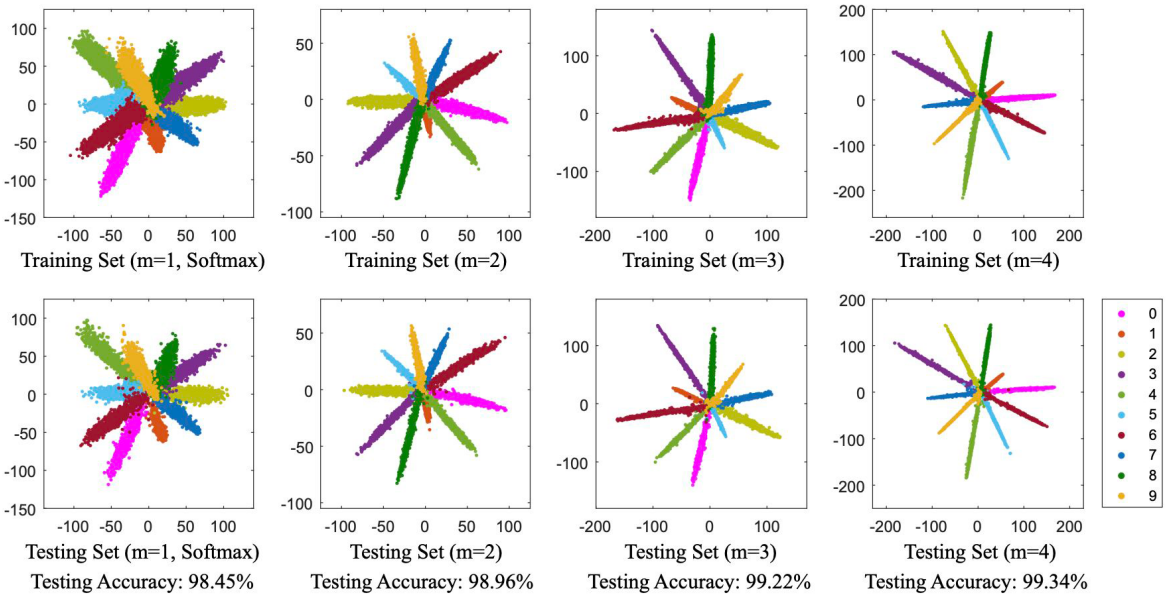
- **FaceNet网络**：
    - **数据预处理**：
        - 首先对识别的人脸的尺寸进行缩放（160×160），即在人脸的周围填充上灰条，使得每张照片都是等尺寸的正方形
        - 然后对图像图片进行归一化
    - **facenet编码**：骨干网络详见第一章介绍的MobileNet v1



    - **人脸特征比对**：
        - 将当前的人脸特征向量和库中所有的人脸特征向量进行对比分析 —— `compare_faces`
        - 计算人脸向量之间的欧式距离
        - 将距离和设置的阈值进行对比，符合（小于等于）的是True，不符合是False
        - 返回和数据库中阈值对比得到的True/False的list，以及所有的distance
        - 获取当前人脸和数据库中人脸欧式距离的最小的人脸的序号
        - 如果当前最小值符合阈值需求，则返回这个人的名字，否则返回"Unknown"
- **已知人脸库的编码**： 对人物照片进行编码（如果需要丰富人脸数据库的话） —— `encoding.py`

3. 损失函数：

- **交叉熵损失函数的问题**：无法确保类内紧凑和类间疏离，尤其是在人脸识别这样的类别很多的情况下
- **L-Softmax loss**：Large-Margin Softmax
    - **代码**：https://github.com/wy1iu/LargeMargin_Softmax_Loss
    - **论文**：https://arxiv.org/abs/1612.02295
    - **Softmax Loss**：$L = -\frac{1}{N}\sum log(\frac{exp(W_i^T x)}{\sum_j^K exp(W_j^T x)})$
    - **内积公式**：$W_j^T x = ||W_j||||x||cos(\theta_j)$
    - **推理**：
        - 假设是一个二分类问题，x是关于属于类别1还是2，我们希望最后输出类别1的得分高
        - 类别1的得分高即$W_1^T x > W_2^T x$，由内积公式可推理 $||W_1||||x||cos(\theta_1) > ||W_2||||x||cos(\theta_2)$
        - 由余弦的定理可知，cos函数在$[0, \pi]$内单调递减，其中当$m \times \theta_1 \in [\theta_1, \pi]$，即 $m \in [1, \frac{\pi}{\theta_1}]$, $m\ is\ integer$时
            - $||W_1||||x||cos(\theta_1) > ||W_1||||x||cos(m\theta_1) > ||W_2||||x||cos(\theta_2)$
            - 所以乘以m后需要满足能否分类的条件更难了，也就是需要类间距离更大，类内距离更小



- **A-Softmax loss**：SphereFace
  - **改进点**：在L-Softmax loss的基础上再加了一个假设，即$||w_i|| = 1$，也就是在代码实现时，首先对w_i进行归一化，这样就是x的分类只和夹角θ有关
  - **注意**：不可以对x同时归一化，这样会导致wx的值域为[-1, 1]，会导致梯度很低
  - **NormFace**：
    - **问题**：对于多分类问题，即使预测完全正确，目标类别的概率也远小于1
    - **改进点**：引入了固定缩放因子$s = l^2$，一般取值固定大于30
- **AM-Softmax loss**：
  - **改进点**：固定固定$||w|| = 1$，$||x|| = 1$
  - **公式**：$L_{AMS} = -\frac{1}{n}\sum_{i=1}^{n} log\frac{e^{s\cdot(cos\theta_{y_i}-m)}}{e^{s\cdot(cos\theta_{y_i}-m)}+\sum_{j=1,j\neq y_i}^{c} e^{s\cdot cos\theta_j}}$，$s = 30$，$m = 0.35$
- **ArcFace loss**：
  - **改进点**：
    - L-Softmax loss和A-Softmax loss都是在x和权重w之间的夹角θ基础上乘以一个正整数m来拉大类间距离同时缩小类内距离。
    - ArcFace是在基于加法的方式，其中x和w都要归一化，那么两者的内积就是夹角的余弦值
  - **公式**：$L_3 = -\frac{1}{n}\sum_{i=1}^{n} log\frac{e^{s\cdot(cos\theta_{y_i}+m)}}{e^{s\cdot(cos\theta_{y_i}+m)}+\sum_{j=1,j\neq y_i}^{c} e^{s\cdot cos\theta_j}}$，$s = 64$，$m = 0.5$

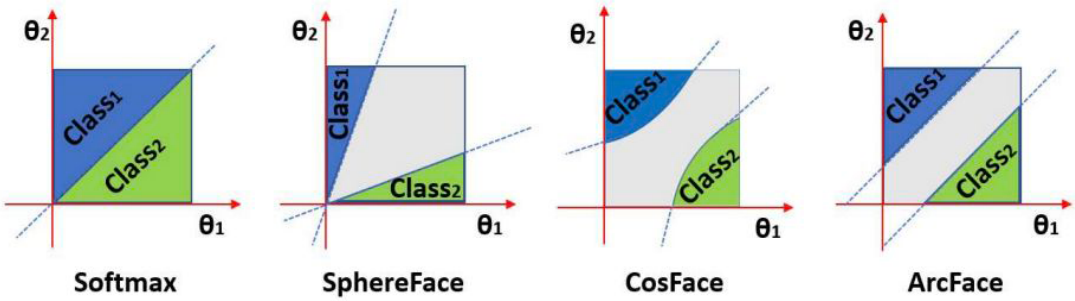
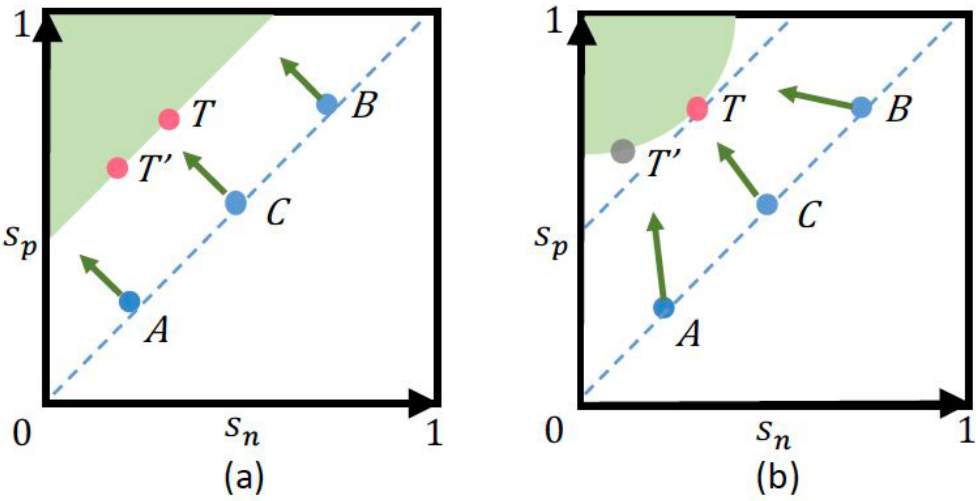


Figure 5. Decision margins of different loss functions under binary classification case. The dashed line represents the decision boundary, and the grey areas are the decision margins.

- **Circle Loss**：https://zhuanlan.zhihu.com/p/445313444

- **理念**：作者认为分类问题中的交叉熵损失函数（样本和类别代理的相似性）和pair对的**triple loss**（样本间的相似性）本质都是优化（ $s_n - s_p$ ），其中s_n表示类间相似度，s_p表示类内相似度。
- **Triple Loss**：
  - **出现**：在FaceNet中提出，用于更好的学习到人脸
  - **输入**：三元组$< a, p, n >$
    - a：anchor，放入训练的数据
    - p：positive，表示和a同一类别的样本
    - n：negative，表示和a不同别的样本
  - **公式**：$L = max(d(a,p) - d(a,n) + margin, 0)$
    - **优化目标**：拉近a和目标类别样本p之间的距离
    - **easy triplets**：$L = 0$即$d(a,p) + margin < d(a,n)$，这种情况a和目标p的距离足够接近，和来分类，和负样本n足够远，不需要优化
    - **hard triplets**：$d(a,n) < d(a,p)$，即a和p的距离过远
    - **semi-hard triplets**：$d(a,p) < d(a,n) < d(a,p) + margin$，即确实$a$和$p$的距离很近，但是$a$和$n$的距离也不远
    - **训练**：选取semi-hard triplets或是再加上hard-triplets进行训练
  - **训练方式**：
    - **离线方式**：训练集所有数据经过计算得到对应的embeddiungs，可以得到很多三元组，再计算损失，但是效率不高，需要遍历所有的三元组然后才能反向更新网络
    - **在线方式**：每次从训练集中抽取一个batch的样本计算损失
      - 假设$B = PK$，P表示有P个有身份的人，每个身份的人取K张照片（一般$K = 4$)
      - **Batch All**：保留semi-hard triplets + hard-triplets的三元组，计算平均损失
        - **平均损失**：因为easy triplets数量较大，所以只对保留的部分做平均损失
        - **样本个数**：一共可以产生$PK(K-1)(PK-K)$个triplets
          - PK个anchor + $K-1$个positive + $(PK-K)$个negative
      - **Batch Hard**：对于每个anchor，选择距离最大的$d(a,p)$和距离最大的$d(a,n)$计算损失
        - 一共有PK个三元组
    - **实现**：`triplet_loss = nn.TripletMarginLoss(margin=1.0, p=2)`
      - **初始化**：
        - **margin**：间隔大小，默认1
        - **p**：归一化维度，默认2
      - **输入**：anchor, positive, negative
- **问题**：
  - **不够灵活**：如下图(a)，A/B/C三个样本的梯度方向都是一样的，但是对于A点，更希望s_p分量上梯度更大点；对于B点，更希望s_n分量上的梯度更大点
  - **收敛状态不明确**：优化（$s_p - s_n - margin$）得到的分类决策面（个人理解应该是loss=0）是$s_n - s_p = margin$（上图(a)中绿色和白色的分界线），取分界线上两个点T和T'，这两个点都是loss=0，但是T的类内相似度和T'的类间相似度差值只有0.1，这种不明确的聚集（其实就是指s_p和s_n没有绝对的数值作为优化目标)会降低特征空间的特征可分性



- **改进点**：$loss = (\alpha_n s_n - \alpha_p s_p)$，即类内和类间相似度有自己不同的步调来学习
- **公式**：$L_{circle} = log[1 + \sum_{j=1}^{L} exp(\gamma \alpha_n^j (s_n^j - \Delta_n)) \sum_{i=1}^{K} exp(-\gamma \alpha_p^i (s_p^i - \Delta_p))]$
  - K：一共有L个参与类内打分（negative），相似度为s_p^j：$(j = 1, 2, \cdots, K)$
  - L：一共有K个参与类间打分（positive），相似度为s_n^i：$(i = 1, 2, \cdots, L)$
  - Δ_p：类内margin大小，$= 1 - m$
  - Δ_n：类间margin大小，$= m$
  - s_p：类内相似度
  - s_n：类间相似度
  - α_p，α_n：s_n和s_p的线性函数
- **超参**：
  - γ：类似于NormFace的s，是给一个大值
  - m：松弛因子，表示分类边界的半径
- **代码实现**：https://github.com/TinyZeaMays/CircleLoss/blob/master/circle_loss.py