关于 FaceNet 人脸识别的理解和改进

FaceNet 可以直接将人脸图像映射到欧几里得空间，空间距离的长度代表了人脸图像的相似性。只要该映射空间生成，就可以轻松完成人脸识别，验证和聚类等任务。它基于深度卷积神经网络，FaceNet 模型学习并输出的是脸部的特征表示，不同人的面部特征表示是不同的，通过计算两个面部特征的距离，来达到不同面部分类的目的。在 LFW 数据集上，准确率为 0.9963，在 YouTube Faces DB 数据集上，准确率为 0.9512。

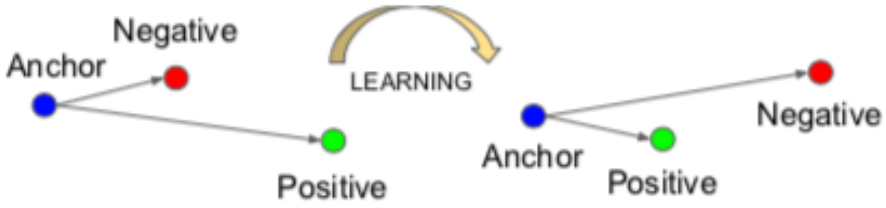
FaceNet 模型是一个通用的系统，采用 CNN 神经网络将人脸图像映射到 128 维的欧几里得空间，根据两幅人像的欧几里得距离去判断两个人像的相似程度。两个人像之间的欧几里得距离越近，说明它们越相似。当人脸特征距离小于 1.06 可看作是同一个人。从初级应用的角度来看，已经训练好的模型已经足够强大，只需将一个基准图片与待分类图片通过 FaceNet 模型比较欧几里得距离，即可完成头像分类任务。

当前存在的基于深度神经网络的人脸识别模型使用了分类层（classification layer）：中间层为人脸图像的向量映射，然后以分类层作为输出层。这类方法的弊端是不直接，效率低。与当前方法不同，FaceNet 直接使用基于 triplets 的 LMNN（最大边界近邻分类）的 loss 函数训练神经网络，网络直接输出为 128 维度的向量空间。

FaceNet 是一个学习人脸图像特征的模型，最大的创新点应该是提出不同的损失函数，直接是优化特征本身，用特征空间上的点的距离来表示两张图像是否是同一类。它的主要优点是：直接使用 CNN 网络进行端到端的训练；使用了 triplet loss，并提出了比较好的 triplet 样本挖掘方法；仅仅用 128 字节的大小表示人脸就达到 state-of-art；需要的预处理少；实验效果好，在公开数据集上取得不错的成绩。

模型介绍

假设我们先不管中间 DEEP ARCHITECTURE 的具体结构，还是将其作为一个黑盒子 f(x)。输入人脸图像数据 x，通过深度 CNN 之后就会得到一个 f(x),再对其做 L2 归一化之后就可以得到对这张人脸的表示 embedding。然后用 triplet loss(anchor, positive, negative)方法对整个模型进行端到端的训练。



triplet loss 的启发是传统 loss 函数趋向于将有一类特征的人脸图像映射到同一个空间。而 triplet loss 尝试将一个个体的人脸图像和其它人脸图像分开。对于一组 triplet sample(anchor, positive, negative)，我们旨在通过神经网络的训练与学习让(anchor,positive)的距离变小，让(anchor,negative)的距离变大；对于任意一组数据，我们都希望(anchor,positive)之间的距离+margin < (anchor,negative)之间的距离。

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2 ,$$
$$\forall (f(x_i^a), f(x_i^p), f(x_i^n)) \in \mathcal{T} .$$

根据上面的式子，得到如下的损失函数。这样的话，目标就转变为训练网络模型使得 loss function 不断变小。

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

如何选择 triplet sample 对于模型的训练以及最后的结构都非常重要，这里提出一种用于寻找较好 triplet sample 的方法，旨在让 triplet 的难度随着网络的训练而逐渐增加。

hard triplet 指的就是那些比较难判断的 sample，体现在 anchor 与正样本的距离较大，anchor 与负样本的距离较小，这样能够训练网络模型识别的难度。

$$\mathrm{argmax}_{x_i^p} \left\| f(x_i^a) - f(x_i^p) \right\|_2^2$$

$$\mathrm{argmin}_{x_i^n} \left\| f(x_i^a) - f(x_i^n) \right\|_2^2$$

对于每一个 anchor，都想选择一个 hard triplet 是有难处的；首先在所有的训练集合内选择最难的 positive 与 negative 是不现实的，需要耗费大量的时间与计算；其次这样选出来的很可能是错误标记或者是不良成像的图像，反而会导致 training 的过程并不好。因此，通过两种策略进行折中。第一种：每 n 步线下生成，用最近一次训练好的网络在训练集的自己种选择 argmax 的 positive 以及 argmin 的 negative。第二种：线上生成，在 mini-batch 中选择 hard positive/nagative。

改进

FaceNet 中脸部缩略图为紧密裁剪的脸部区域，没有使用 2d，3d 对齐以及放大转换等预处理。可以在训练神经网络前，使用对齐方法，一旦人脸经过对齐，人脸区域的特征就固定在某些领域上，网络参数如下所示。

Conv：32 个 11×11×3 的卷积核；max-pooling: 3×3， stride=2；Conv: 16 个 9×9 的卷积核；Local-Conv： 16 个 9×9 的卷积核，Local 的意思是卷积核的参数不共享；Local-Conv: 16 个 7×7 的卷积核，参数不共享；Local-Conv: 16 个 5×5 的卷积核，参数不共享；Fully-connected: 4096 维；Softmax: 4030 维

提取低水平特征，过程如下所示：

1. 预处理阶段：输入 3 通道的人脸，并进行 3D 校正，再归一化到 152*152 像素大小——152*152*3.
2. 通过卷积层 C1：C1 包含 32 个 11*11*3 的滤波器（即卷积核），得到 32 张特征图——32*142*142*3。

3. 通过 max-pooling 层 M2：M2 的滑窗大小为 3*3，滑动步长为 2，3 个通道上分别独立 pooling。
4. 通过另一个卷积层 C3：C3 包含 16 个 9*9*16 的 3 维卷积核。

上述 3 层网络是为了提取到低水平的特征，如简单的边缘特征和纹理特征。Max-pooling 层使得卷积网络对局部的变换更加鲁棒。如果输入是校正后的人脸，就能使网络对小的标记误差更加鲁棒。然而这样的 pooling 层会使网络在面部的细节结构和微小纹理的精准位置上丢失一些信息。因此，只在第一个卷积层后面接了 Max-pooling 层。

L4,L5,L6 都是局部连接层，就像卷积层使用滤波器一样，在特征图像的每一个位置都训练学习一组不同的滤波器。由于校正后不同区域有不同的统计特性，卷积网络在空间上稳定性的假设不能成立。比如说，相比于鼻子和嘴巴之间的区域，眼睛和眉毛之间的区域展现出非常不同的表观并且有很高的区分度。换句话说，通过利用输入的校正后的图像，定制通过计算叉熵损失 L 对参数的梯度以及使用随机梯度递减的方法来最小化叉熵损失。

梯度是通过误差的标准反向传播来计算的。非常有趣的是，本网络产生的特征非常稀疏。超过 75%的顶层特征元素是 0。这主要是由于使用了 ReLU 激活函数导致的。这种软阈值非线性函数在所有的卷积层，局部连接层和全连接层（除了最后一层 F8）都使用了，从而导致整体级联之后产生高度非线性和稀疏的特征。稀疏性也与使用 dropout 正则化有关，即在训练中将随机的特征元素设置为 0。只在 F7 全连接层使用了 dropout.由于训练集合很大，在训练过程中没有发现重大的过拟合。

最后，网络顶端的两层（F7，F8）是全连接的：每一个输出单元都连接到所有的输入。这两层可以捕捉到人脸图像中距离较远的区域的特征之间的关联性。比如，眼睛的位置和形状，与嘴巴的位置和形状之间的关联性（这部分也含有信息）可以由这两层得到。第一个全连接层 F7 的输出就是我们原始的人脸特征表达向量。最后一个全连接层 F8 的输出进入了一个 K-way 的 softmax（K 是类别个数）,即可产生类别标号的概率分布。

代码

- 安装以下库，保证正常使用：tensorflow==1.7,scipy,scikit-learn,open

cv-python,h5py,matplotlib,Pillow,requests,psutil

- 因为程序中神经网络使用的是谷歌的“inception resent v1”网络模型，模型的输入是 160*160 的图像，需要首先对 LFW 数据集 250*250 的图片进行预处理，即运行 facenet/src/align/align_dataset_mtcnn.py 来修改图片尺寸大小，python align_dataset_mtcnn.py facenet/data/lfw_data/lfw facenet/data/lfw_data/lfw_160 --image_size 160 --margin 32 --random_order--gpu_memory_fraction 0.25
- facenet 提供两个预训练模型，分别是基于 CASIA-WebFace 和 VGGFace2 人脸库训练的，将下载好的预训练文件解压到 facenet/src/models 目录下：添加参数 facenet/data/lfw_data/lfw_160 facenet/src/models/20180408-102900，运行 validate_on_lfw.py 文件，识别精度可以达到 98.7%。

```
Accuracy: 0.98767+-0.00544
Validation rate: 0.92500+-0.02156 @ FAR=0.00100
Area Under Curve (AUC): 0.999
Equal Error Rate (EER): 0.014
```

代码截取对应 py 文件中必要重要的部分

align_dataset_mtcnn.py

```
with open(bounding_boxes_filename, "w") as text_file:
        nrof_images_total = 0
        nrof_successfully_aligned = 0
        if args.random_order:
            random.shuffle(dataset)
        for cls in dataset:
            output_class_dir = os.path.join(output_dir, cls.name)
            if not os.path.exists(output_class_dir):
                os.makedirs(output_class_dir)
                if args.random_order:
                    random.shuffle(cls.image_paths)
            for image_path in cls.image_paths:
                nrof_images_total += 1
                filename = os.path.splitext(os.path.split(image_path)[1])[0]
                output_filename = os.path.join(output_class_dir, filename+'.png')
                print(image_path)
                if not os.path.exists(output_filename):
                    try:
                        img = misc.imread(image_path)
                    except (IOError, ValueError, IndexError) as e:
```

```
                errorMessage = '{}: {}'.format(image_path, e)
                print(errorMessage)
            else:
                if img.ndim<2:
                    print('Unable to align "%s"' % image_path)
                    text_file.write('%s\n' % (output_filename))
                    continue
                if img.ndim == 2:
                    img = facenet.to_rgb(img)
                img = img[:,:,0:3]

                bounding_boxes, _ = align.detect_face.detect_face(img, minsize, pnet, rnet,
onet, threshold, factor)
                nrof_faces = bounding_boxes.shape[0]
                if nrof_faces>0:
                    det = bounding_boxes[:,0:4]
                    det_arr = []
                    img_size = np.asarray(img.shape)[0:2]
                    if nrof_faces>1:
                        if args.detect_multiple_faces:
                            for i in range(nrof_faces):
                                det_arr.append(np.squeeze(det[i]))
                        else:
                            bounding_box_size = (det[:,2]-det[:,0])*(det[:,3]-det[:,1])
                            img_center = img_size / 2
                            offsets = np.vstack([ (det[:,0]+det[:,2])/2-img_center[1],
(det[:,1]+det[:,3])/2-img_center[0] ])
                            offset_dist_squared = np.sum(np.power(offsets,2.0),0)
                            index = np.argmax(bounding_box_size-offset_dist_squared*2.0) #
some extra weight on the centering
                            det_arr.append(det[index,:])
                    else:
                        det_arr.append(np.squeeze(det))

                    for i, det in enumerate(det_arr):
                        det = np.squeeze(det)
                        bb = np.zeros(4, dtype=np.int32)
                        bb[0] = np.maximum(det[0]-args.margin/2, 0)
                        bb[1] = np.maximum(det[1]-args.margin/2, 0)
                        bb[2] = np.minimum(det[2]+args.margin/2, img_size[1])
                        bb[3] = np.minimum(det[3]+args.margin/2, img_size[0])
                        cropped = img[bb[1]:bb[3],bb[0]:bb[2],:]
                        scaled = misc.imresize(cropped, (args.image_size, args.image_size),
interp='bilinear')
```

```
                                    nrof_successfully_aligned += 1
                                    filename_base, file_extension = os.path.splitext(output_filename)
                                    if args.detect_multiple_faces:
                                        output_filename_n = "{}_{}{}".format(filename_base, i,
file_extension)
                                    else:
                                        output_filename_n = "{}{}".format(filename_base, file_extension)
                                    misc.imsave(output_filename_n, scaled)
                                    text_file.write('%s %d %d %d %d\n' % (output_filename_n, bb[0], bb[1],
bb[2], bb[3]))
                            else:
                                print('Unable to align "%s"' % image_path)
                                text_file.write('%s\n' % (output_filename))
```

detect_face.py

```
class PNet(Network):
    def setup(self):
        (self.feed('data') #pylint: disable=no-value-for-parameter, no-member
             .conv(3, 3, 10, 1, 1, padding='VALID', relu=False, name='conv1')
             .prelu(name='PReLU1')
             .max_pool(2, 2, 2, 2, name='pool1')
             .conv(3, 3, 16, 1, 1, padding='VALID', relu=False, name='conv2')
             .prelu(name='PReLU2')
             .conv(3, 3, 32, 1, 1, padding='VALID', relu=False, name='conv3')
             .prelu(name='PReLU3')
             .conv(1, 1, 2, 1, 1, relu=False, name='conv4-1')
             .softmax(3,name='prob1'))

        (self.feed('PReLU3') #pylint: disable=no-value-for-parameter
             .conv(1, 1, 4, 1, 1, relu=False, name='conv4-2'))

class RNet(Network):
    def setup(self):
        (self.feed('data') #pylint: disable=no-value-for-parameter, no-member
             .conv(3, 3, 28, 1, 1, padding='VALID', relu=False, name='conv1')
             .prelu(name='prelu1')
             .max_pool(3, 3, 2, 2, name='pool1')
             .conv(3, 3, 48, 1, 1, padding='VALID', relu=False, name='conv2')
             .prelu(name='prelu2')
             .max_pool(3, 3, 2, 2, padding='VALID', name='pool2')
             .conv(2, 2, 64, 1, 1, padding='VALID', relu=False, name='conv3')
             .prelu(name='prelu3')
```

```
                .fc(128, relu=False, name='conv4')
                .prelu(name='prelu4')
                .fc(2, relu=False, name='conv5-1')
                .softmax(1,name='prob1'))

        (self.feed('prelu4') #pylint: disable=no-value-for-parameter
                .fc(4, relu=False, name='conv5-2'))

class ONet(Network):
    def setup(self):
        (self.feed('data') #pylint: disable=no-value-for-parameter, no-member
                .conv(3, 3, 32, 1, 1, padding='VALID', relu=False, name='conv1')
                .prelu(name='prelu1')
                .max_pool(3, 3, 2, 2, name='pool1')
                .conv(3, 3, 64, 1, 1, padding='VALID', relu=False, name='conv2')
                .prelu(name='prelu2')
                .max_pool(3, 3, 2, 2, padding='VALID', name='pool2')
                .conv(3, 3, 64, 1, 1, padding='VALID', relu=False, name='conv3')
                .prelu(name='prelu3')
                .max_pool(2, 2, 2, 2, name='pool3')
                .conv(2, 2, 128, 1, 1, padding='VALID', relu=False, name='conv4')
                .prelu(name='prelu4')
                .fc(256, relu=False, name='conv5')
                .prelu(name='prelu5')
                .fc(2, relu=False, name='conv6-1')
                .softmax(1, name='prob1'))

        (self.feed('prelu5') #pylint: disable=no-value-for-parameter
                .fc(4, relu=False, name='conv6-2'))

        (self.feed('prelu5') #pylint: disable=no-value-for-parameter
                .fc(10, relu=False, name='conv6-3'))

def create_mtcnn(sess, model_path):
    if not model_path:
        model_path,_ = os.path.split(os.path.realpath(__file__))

    with tf.variable_scope('pnet'):
        data = tf.placeholder(tf.float32, (None,None,None,3), 'input')
        pnet = PNet({'data':data})
        pnet.load(os.path.join(model_path, 'det1.npy'), sess)
    with tf.variable_scope('rnet'):
        data = tf.placeholder(tf.float32, (None,24,24,3), 'input')
        rnet = RNet({'data':data})
```

```
            rnet.load(os.path.join(model_path, 'det2.npy'), sess)
        with tf.variable_scope('onet'):
            data = tf.placeholder(tf.float32, (None,48,48,3), 'input')
            onet = ONet({'data':data})
            onet.load(os.path.join(model_path, 'det3.npy'), sess)

        pnet_fun = lambda img : sess.run(('pnet/conv4-2/BiasAdd:0', 'pnet/prob1:0'),
feed_dict={'pnet/input:0':img})
        rnet_fun = lambda img : sess.run(('rnet/conv5-2/conv5-2:0', 'rnet/prob1:0'),
feed_dict={'rnet/input:0':img})
        onet_fun = lambda img : sess.run(('onet/conv6-2/conv6-2:0', 'onet/conv6-3/conv6-3:0',
'onet/prob1:0'), feed_dict={'onet/input:0':img})
        return pnet_fun, rnet_fun, onet_fun
```

```
def detect_face(img, minsize, pnet, rnet, onet, threshold, factor):
    """Detects faces in an image, and returns bounding boxes and points for them.
    img: input image
    minsize: minimum faces' size
    pnet, rnet, onet: caffemodel
    threshold: threshold=[th1, th2, th3], th1-3 are three steps's threshold
    factor: the factor used to create a scaling pyramid of face sizes to detect in the image.
    """
    factor_count=0
    total_boxes=np.empty((0,9))
    points=np.empty(0)
    h=img.shape[0]
    w=img.shape[1]
    minl=np.amin([h, w])
    m=12.0/minsize
    minl=minl*m
    # create scale pyramid
    scales=[]
    while minl>=12:
        scales += [m*np.power(factor, factor_count)]
        minl = minl*factor
        factor_count += 1

    # first stage
    for scale in scales:
        hs=int(np.ceil(h*scale))
        ws=int(np.ceil(w*scale))
        im_data = imresample(img, (hs, ws))
        im_data = (im_data-127.5)*0.0078125
```

```
        img_x = np.expand_dims(im_data, 0)
        img_y = np.transpose(img_x, (0,2,1,3))
        out = pnet(img_y)
        out0 = np.transpose(out[0], (0,2,1,3))
        out1 = np.transpose(out[1], (0,2,1,3))

        boxes, _ = generateBoundingBox(out1[0,:,:,1].copy(), out0[0,:,:,:].copy(), scale, threshold[0])

        # inter-scale nms
        pick = nms(boxes.copy(), 0.5, 'Union')
        if boxes.size>0 and pick.size>0:
            boxes = boxes[pick,:]
            total_boxes = np.append(total_boxes, boxes, axis=0)

    numbox = total_boxes.shape[0]
    if numbox>0:
        pick = nms(total_boxes.copy(), 0.7, 'Union')
        total_boxes = total_boxes[pick,:]
        regw = total_boxes[:,2]-total_boxes[:,0]
        regh = total_boxes[:,3]-total_boxes[:,1]
        qq1 = total_boxes[:,0]+total_boxes[:,5]*regw
        qq2 = total_boxes[:,1]+total_boxes[:,6]*regh
        qq3 = total_boxes[:,2]+total_boxes[:,7]*regw
        qq4 = total_boxes[:,3]+total_boxes[:,8]*regh
        total_boxes = np.transpose(np.vstack([qq1, qq2, qq3, qq4, total_boxes[:,4]]))
        total_boxes = rerec(total_boxes.copy())
        total_boxes[:,0:4] = np.fix(total_boxes[:,0:4]).astype(np.int32)
        dy, edy, dx, edx, y, ey, x, ex, tmpw, tmph = pad(total_boxes.copy(), w, h)

    numbox = total_boxes.shape[0]
    if numbox>0:
        # second stage
        tempimg = np.zeros((24,24,3,numbox))
        for k in range(0,numbox):
            tmp = np.zeros((int(tmph[k]),int(tmpw[k]),3))
            tmp[dy[k]-1:edy[k],dx[k]-1:edx[k],:] = img[y[k]-1:ey[k],x[k]-1:ex[k],:]
            if tmp.shape[0]>0 and tmp.shape[1]>0 or tmp.shape[0]==0 and tmp.shape[1]==0:
                tempimg[:,:,:,k] = imresample(tmp, (24, 24))
            else:
                return np.empty()
        tempimg = (tempimg-127.5)*0.0078125
        tempimg1 = np.transpose(tempimg, (3,1,0,2))
        out = rnet(tempimg1)
        out0 = np.transpose(out[0])
```

```
            out1 = np.transpose(out[1])
            score = out1[1,:]
            ipass = np.where(score>threshold[1])
            total_boxes = np.hstack([total_boxes[ipass[0],0:4].copy(),
np.expand_dims(score[ipass].copy(),1)])
            mv = out0[:,ipass[0]]
            if total_boxes.shape[0]>0:
                pick = nms(total_boxes, 0.7, 'Union')
                total_boxes = total_boxes[pick,:]
                total_boxes = bbreg(total_boxes.copy(), np.transpose(mv[:,pick]))
                total_boxes = rerec(total_boxes.copy())

        numbox = total_boxes.shape[0]
        if numbox>0:
            # third stage
            total_boxes = np.fix(total_boxes).astype(np.int32)
            dy, edy, dx, edx, y, ey, x, ex, tmpw, tmph = pad(total_boxes.copy(), w, h)
            tempimg = np.zeros((48,48,3,numbox))
            for k in range(0,numbox):
                tmp = np.zeros((int(tmph[k]),int(tmpw[k]),3))
                tmp[dy[k]-1:edy[k],dx[k]-1:edx[k],:] = img[y[k]-1:ey[k],x[k]-1:ex[k],:]
                if tmp.shape[0]>0 and tmp.shape[1]>0 or tmp.shape[0]==0 and tmp.shape[1]==0:
                    tempimg[:,:,:,k] = imresample(tmp, (48, 48))
                else:
                    return np.empty()
            tempimg = (tempimg-127.5)*0.0078125
            tempimg1 = np.transpose(tempimg, (3,1,0,2))
            out = onet(tempimg1)
            out0 = np.transpose(out[0])
            out1 = np.transpose(out[1])
            out2 = np.transpose(out[2])
            score = out2[1,:]
            points = out1
            ipass = np.where(score>threshold[2])
            points = points[:,ipass[0]]
            total_boxes = np.hstack([total_boxes[ipass[0],0:4].copy(),
np.expand_dims(score[ipass].copy(),1)])
            mv = out0[:,ipass[0]]

            w = total_boxes[:,2]-total_boxes[:,0]+1
            h = total_boxes[:,3]-total_boxes[:,1]+1
            points[0:5,:] = np.tile(w,(5, 1))*points[0:5,:] + np.tile(total_boxes[:,0],(5, 1))-1
            points[5:10,:] = np.tile(h,(5, 1))*points[5:10,:] + np.tile(total_boxes[:,1],(5, 1))-1
            if total_boxes.shape[0]>0:
```

```
            total_boxes = bbreg(total_boxes.copy(), np.transpose(mv))
            pick = nms(total_boxes.copy(), 0.7, 'Min')
            total_boxes = total_boxes[pick,:]
            points = points[:,pick]

    return total_boxes, points
```

facenet.py

```
def triplet_loss(anchor, positive, negative, alpha):
    """Calculate the triplet loss according to the FaceNet paper

    Args:
      anchor: the embeddings for the anchor images.
      positive: the embeddings for the positive images.
      negative: the embeddings for the negative images.

    Returns:
      the triplet loss according to the FaceNet paper as a float tensor.
    """
    with tf.variable_scope('triplet_loss'):
        pos_dist = tf.reduce_sum(tf.square(tf.subtract(anchor, positive)), 1)
        neg_dist = tf.reduce_sum(tf.square(tf.subtract(anchor, negative)), 1)

        basic_loss = tf.add(tf.subtract(pos_dist,neg_dist), alpha)
        loss = tf.reduce_mean(tf.maximum(basic_loss, 0.0), 0)

    return loss
def train(total_loss, global_step, optimizer, learning_rate, moving_average_decay,
update_gradient_vars, log_histograms=True):
    # Generate moving averages of all losses and associated summaries.
    loss_averages_op = _add_loss_summaries(total_loss)

    # Compute gradients.
    with tf.control_dependencies([loss_averages_op]):
        if optimizer=='ADAGRAD':
            opt = tf.train.AdagradOptimizer(learning_rate)
        elif optimizer=='ADADELTA':
            opt = tf.train.AdadeltaOptimizer(learning_rate, rho=0.9, epsilon=1e-6)
        elif optimizer=='ADAM':
            opt = tf.train.AdamOptimizer(learning_rate, beta1=0.9, beta2=0.999, epsilon=0.1)
```

```
        elif optimizer=='RMSPROP':
            opt = tf.train.RMSPropOptimizer(learning_rate, decay=0.9, momentum=0.9, epsilon=1.0)
        elif optimizer=='MOM':
            opt = tf.train.MomentumOptimizer(learning_rate, 0.9, use_nesterov=True)
        else:
            raise ValueError('Invalid optimization algorithm')

        grads = opt.compute_gradients(total_loss, update_gradient_vars)

    # Apply gradients.
    apply_gradient_op = opt.apply_gradients(grads, global_step=global_step)

    # Add histograms for trainable variables.
    if log_histograms:
        for var in tf.trainable_variables():
            tf.summary.histogram(var.op.name, var)

    # Add histograms for gradients.
    if log_histograms:
        for grad, var in grads:
            if grad is not None:
                tf.summary.histogram(var.op.name + '/gradients', grad)

    # Track the moving averages of all trainable variables.
    variable_averages = tf.train.ExponentialMovingAverage(
        moving_average_decay, global_step)
    variables_averages_op = variable_averages.apply(tf.trainable_variables())

    with tf.control_dependencies([apply_gradient_op, variables_averages_op]):
        train_op = tf.no_op(name='train')

    return train_op
```

validate_on_lfw.py

```
def main(args):

    with tf.Graph().as_default():

        with tf.Session() as sess:
```

```
            # Read the file containing the pairs used for testing
            pairs = lfw.read_pairs(os.path.expanduser(args.lfw_pairs))

            # Get the paths for the corresponding images
            paths, actual_issame = lfw.get_paths(os.path.expanduser(args.lfw_dir), pairs)

            image_paths_placeholder = tf.placeholder(tf.string, shape=(None,1), name='image_paths')
            labels_placeholder = tf.placeholder(tf.int64, shape=(None,1), name='labels')
            batch_size_placeholder = tf.placeholder(tf.int32, name='batch_size')
            control_placeholder = tf.placeholder(tf.int64, shape=(None,1), name='control')
            phase_train_placeholder = tf.placeholder(tf.bool, name='phase_train')

            nrof_preprocess_threads = 4
            image_size = (args.image_size, args.image_size)
            eval_input_queue = data_flow_ops.FIFOQueue(capacity=2000000,
                                        dtypes=[tf.string, tf.int64, tf.int64],
                                        shapes=[(1,), (1,), (1,)],
                                        shared_name=None, name=None)
            eval_enqueue_op = eval_input_queue.enqueue_many([image_paths_placeholder,
labels_placeholder, control_placeholder], name='eval_enqueue_op')
            image_batch, label_batch = facenet.create_input_pipeline(eval_input_queue, image_size,
nrof_preprocess_threads, batch_size_placeholder)

            # Load the model
            input_map = {'image_batch': image_batch, 'label_batch': label_batch, 'phase_train':
phase_train_placeholder}
            facenet.load_model(args.model, input_map=input_map)

            # Get output tensor
            embeddings = tf.get_default_graph().get_tensor_by_name("embeddings:0")
#
            coord = tf.train.Coordinator()
            tf.train.start_queue_runners(coord=coord, sess=sess)

            evaluate(sess, eval_enqueue_op, image_paths_placeholder, labels_placeholder,
phase_train_placeholder, batch_size_placeholder, control_placeholder,
                    embeddings, label_batch, paths, actual_issame, args.lfw_batch_size,
args.lfw_nrof_folds, args.distance_metric, args.subtract_mean,
                    args.use_flipped_images, args.use_fixed_image_standardization)
def evaluate(sess, enqueue_op, image_paths_placeholder, labels_placeholder,
phase_train_placeholder, batch_size_placeholder, control_placeholder,
        embeddings, labels, image_paths, actual_issame, batch_size, nrof_folds, distance_metric,
subtract_mean, use_flipped_images, use_fixed_image_standardization):
    # Run forward pass to calculate embeddings
```

```
    print('Runnning forward pass on LFW images')

    # Enqueue one epoch of image paths and labels
    nrof_embeddings = len(actual_issame)*2  # nrof_pairs * nrof_images_per_pair
    nrof_flips = 2 if use_flipped_images else 1
    nrof_images = nrof_embeddings * nrof_flips
    labels_array = np.expand_dims(np.arange(0,nrof_images),1)
    image_paths_array = np.expand_dims(np.repeat(np.array(image_paths),nrof_flips),1)
    control_array = np.zeros_like(labels_array, np.int32)
    if use_fixed_image_standardization:
        control_array += np.ones_like(labels_array)*facenet.FIXED_STANDARDIZATION
    if use_flipped_images:
        # Flip every second image
        control_array += (labels_array % 2)*facenet.FLIP
    sess.run(enqueue_op, {image_paths_placeholder: image_paths_array, labels_placeholder:
labels_array, control_placeholder: control_array})

    embedding_size = int(embeddings.get_shape()[1])
    assert nrof_images % batch_size == 0, 'The number of LFW images must be an integer multiple of
the LFW batch size'
    nrof_batches = nrof_images // batch_size
    emb_array = np.zeros((nrof_images, embedding_size))
    lab_array = np.zeros((nrof_images,))
    for i in range(nrof_batches):
        feed_dict = {phase_train_placeholder:False, batch_size_placeholder:batch_size}
        emb, lab = sess.run([embeddings, labels], feed_dict=feed_dict)
        lab_array[lab] = lab
        emb_array[lab, :] = emb
        if i % 10 == 9:
            print('.', end='')
            sys.stdout.flush()
    print('')
    embeddings = np.zeros((nrof_embeddings, embedding_size*nrof_flips))
    if use_flipped_images:
        # Concatenate embeddings for flipped and non flipped version of the images
        embeddings[:,:embedding_size] = emb_array[0::2,:]
        embeddings[:,embedding_size:] = emb_array[1::2,:]
    else:
        embeddings = emb_array

    assert np.array_equal(lab_array, np.arange(nrof_images))==True, 'Wrong labels used for
evaluation, possibly caused by training examples left in the input pipeline'
    tpr, fpr, accuracy, val, val_std, far = lfw.evaluate(embeddings, actual_issame, nrof_folds=nrof_folds,
distance_metric=distance_metric, subtract_mean=subtract_mean)
```

```
print('Accuracy: %2.5f+-%2.5f' % (np.mean(accuracy), np.std(accuracy)))
print('Validation rate: %2.5f+-%2.5f @ FAR=%2.5f' % (val, val_std, far))

auc = metrics.auc(fpr, tpr)
print('Area Under Curve (AUC): %1.3f' % auc)
eer = brentq(lambda x: 1. - x - interpolate.interp1d(fpr, tpr)(x), 0., 1.)
print('Equal Error Rate (EER): %1.3f' % eer)
```