使用Deepfakes换脸及检测方法的探究

郭鸿儒 赵屹东

版本：0.08

日期：2020 年 6 月 19 日

摘　　要

本文探究了deepfakes换脸的基本原理，基于python的dlib尝试实现了一种简单的换脸方式，并在探究deepfacelab的基础上，使用不同的模型进行了换脸，最后，我们试图通过脸的边缘检测，眨眼检测对换脸视频进行检测，并取得了一定的效果。

关键词: deepfakes，dlib，人脸识别

1 绪论

1.1 选题背景

2017年12月，女明星Gal Gadot的换脸视频被一个网名叫deepfakes的程序员发布，AI换脸进入到人们的视野当中，Deepfakes也逐渐变得火热，各种各样的换脸视频层出不穷，视频中的换脸也都达到以假乱真的程度，而其开发者甚至推出简单的换脸软件（如faceapp），使得换脸难度进一步降低。与此同时，对于换脸技术的担忧也越来越强，也有许多反deepfakes技术的出现。基于以上情况，我们希望能够在了解deepfakes的基础上，实现换脸，并且能够针对其技术核心，提出检测deepfakes的思路及方法。

1.2 实验环境

- 本次实验的代码主要基于python3.6
- 使用deepfacelab换脸
- 显卡配置为NVIDIA GeForce GTX 1050 4G内存
- cuda版本为9.2
- 组内任务主要有查资料，跑程序，写代码，两人共同完成

2 Deepfakes

Deepfakes换脸主要分成以下几个步骤

1. 对替换视频和源视频逐帧提取人脸
2. 通过提取的图片对模型进行训练
3. 使用训练得到的模型对要替换视频的图片进行逐帧变换

4. 利用新得到的图片合成视频

2.1 人脸提取

deepfakes使用ffmpeg来进行视频转图片的操作，调用main函数中的process_extract()函数,并可以选择'dlib','mt','s3fd','manual'四种算法进行人脸检测，接着调用facelib中对应的算法及模型进行图片切割。在此过程中，对于data_src.mp4转的图片存储在data_src文件夹下，而提取到的人脸则存储在data_src/aligned文件夹下。对于data_dst.mp4，得到的图片则存储在data_dst文件夹下，此外，在这一过程中，同时进行了基于特征点提取进行的人脸校准工作(如dlib的68个关键点检测+普氏分析)，目的是保持前后的表情一致，得到的图片存储在data_dst/aligned_debug文件夹下。

2.2 Encoder&Decoder

deepfakes的关键在于人脸的转换，deepface使用Autoencoder模型来进行学习，Autoencoder的编码器(Encoder)把图片进行压缩，而解码器(Decoder)把图片进行还原。为了学习人脸共性的地方，对所有的脸用一个统一的编码器Encoder，而对于每个脸，有一个单独的解码器Decoder，来学习个性的地方。比如用A的脸通过编码器，而使用B的解码器，就会得到一个与A表情一致的B的脸，原理如图1所示。在deepfacelab中，这一过程被可视化，每经过一次训练，会得到一个图2类型的结果，假如我们要将A的脸换到B的脸上，则其中五列分别是A脸的图片，A脸经过Encoder和A脸的decoder后得到的图片，B脸的照片，B脸经过Encoder和B脸的decoder后得到的图片，B脸经过Encoder和A脸的decoder后得到的图片。而图片上的两条曲线为迭代曲线。

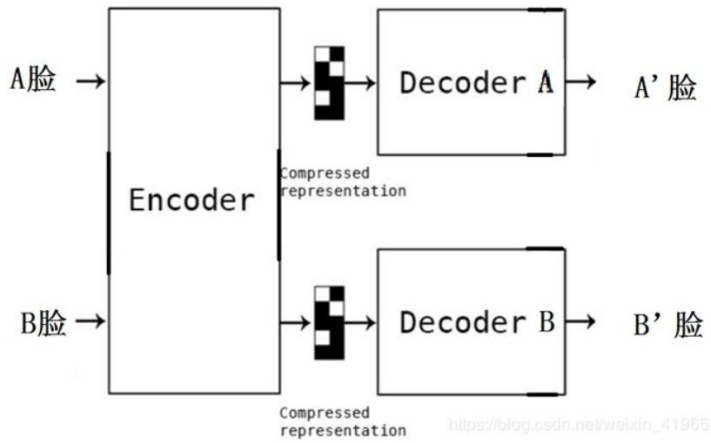


图 1: Encoder和Decoder的原理

在deepfacelab中，目前有8个换脸模型：AVATAR,H64,H128,DF,LIAEF128,Quick96,SAE,SAEHD。不同的模型有不同的参数和训练过程。其中H64，H128，DF三者的模型结构类似，以H64和H128为例，其不同在于选择的范围的大小，H64选择64×64大小的区域，而H128则选择128×128大小的区域。两个模型的Encoder都是四层卷积层(downscale)+两层全连接层(Dense）+一层upscale，而Decoder都是三层upscale+一层卷积层(downscale)。其不同就在于全连接层的参数不同。upscale的

核心是PixelShuffler()，该函数对图像进行了一定的扭曲，增加了学习的难度，也帮助模型实现了最后的结果。因为实验中使用A的扭曲来还原A，而最后要使用B来还原A，所以扭曲是这一过程的核心，也是关键所在，一个恰到好处的扭曲是很重要的。

而SAE和SAEHD结构类似，两个模型的解码器基本相同，而编码器有较大不同，SAE使用四层卷积层+两层全连接层+一层upscale，SAEHD使用四种不同的四层卷积层(downscale)的串联(Concatenate)+两层全连接层+一层upscale。LIAEF128模型的Encoder为四层卷积层，Decoder为三层upscale+一层卷积层，此外，还有一个Intermediate层，其由两个全连接层和一个upscale组成。而AVATAR模型就像阿凡达模型，要求素材的分辨率比例相同，达到换头的效果。Quick96模型的Encoder由两种不同的四层卷积层(downscale)的串联(Concatenate)+两个全连接的maxout层+残差网络+upscale+残差网络构成，decoder由三层upscale+残差网络构成。

此外，模型是选择半脸还是全脸，模型参数的选择，两个脸的特征(如肤色，脸型等)，两个视频的质量，显存的大小等等，都影响着换脸的质量。半脸和全脸的不同在于面部的选择不同，一个形象的解释是孙悟空的脸，全部的脸(包括毛不包括额头)是全脸，而没有毛的部分则是半脸。H64和H128是半脸模型，适合脸型非常相似的两个面部换脸，而DF以及LIAEF128是全脸模型，适合稍有不同的面部，SAE和SAEHD可以通过调参得到很好的效果，但训练比较慢，对显存要求较高。

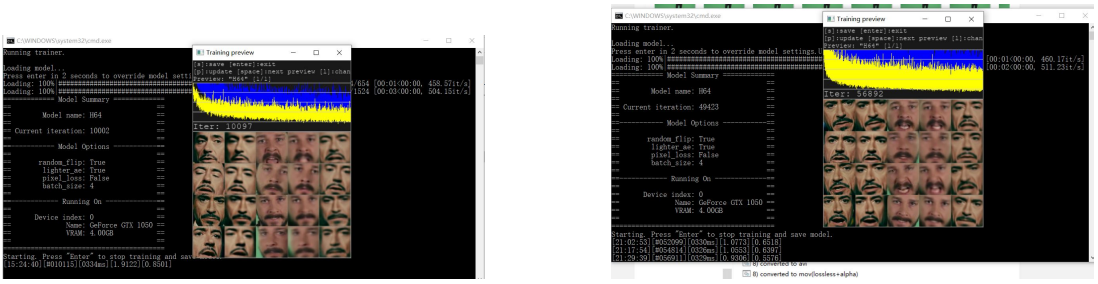


图 2: 左图和右图为使用H64模型迭代10097次和56892次后得到的结果

2.3 人脸转换及合成视频

其实这一部分在之前的训练过程中就有所体现，要将B的脸换到A的脸上，就要把每一张A脸经过Encoder和B脸的decoder，然后替换掉之前人脸检测得到的部分，生成一系列新图存储在data_dst/merged文件夹下，在这一过程中，可以对每一张图片进行调参，直到所有的图片确认，这一过程十分消耗时间和精力，最后通过ffmpeg合成视频，并加入原来的音频，换脸完成。

2.4 实践与结果

由于显卡的限制，我们利用其自带的视频，完成了使用H64和SAE模型的换脸，并在H64迭代10000次和60000次，SAE迭代10000次的情况下，分别合成了视频H64_10000.mp4，H64_60000.mp4，SAE_10000.mp4，截图如下：

可以看到，由于两个人的脸型，肤色，脸的大小不一样，使用H64半脸模型得到的结果有很强的ps痕迹，就好像硬生生贴了一张图上去，但是迭代60000次得到的结果比10000次的结果

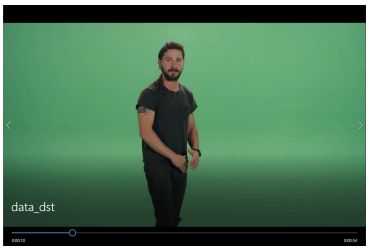
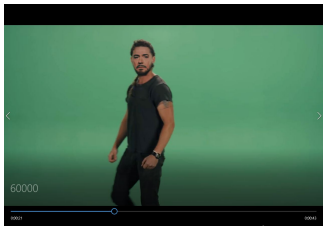


图 3: 左图和右图分别为data_src.mp4和data_dst.mp4的截图

(a) H64 10000itr



(b) H64 60000itr

(c) SAE 10000itr

图 4: 三个视频较好的结果图

有了较大的优化，像眼睛这类细节更加清晰。而使用SAE全脸模型得到的结果比较好，换完的脸更像源视频中的脸，可还是比较容易看出换脸的痕迹，因为换脸不包括额头，所以额头以上的部分和以下的部分会有很明显的肤色差异，而这一点是全脸模型也无法避免的，只可以在转换的过程通过调参进行一定优化。 以面部遮挡为例，deepfacelab提供了是否保留面部遮挡的参

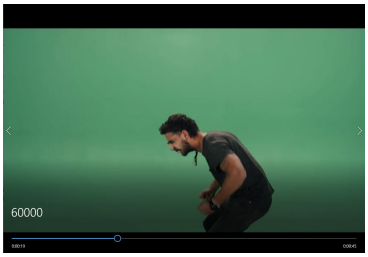
(a) H64 10000itr 面部不遮挡

(b) SAE 10000itr 面部遮挡

图 5: 面部遮挡参数的影响

数，而假如不遮挡，则视频的真假就会一目了然。 SAE模型的侧脸也比H64模型的侧脸更加真实。

综上，SAE模型的换脸更加成功，而deepfakes技术的一些不足之处也被暴露了出来：首先是因为其是将脸替换，所以不可避免地，会忽略其他的细节，也就会导致换完的脸与原来边缘的不融合甚至互斥的地步。其次，由于是逐帧替换，其势必不会考虑帧与帧之间的关系，从而造成一些细节失真，比如眨眼，呼吸频率的变化或者不自然。此外，一旦脸开始运动，尤其是进行转头，抬手动作时，换脸就遇到了最大的挑战，视频的真假也就会比较容易判别。



(a) H64 60000itr

(b) SAE 10000itr

图 6: 侧脸的换脸

3 换脸的简单实现

在了解deepfakes换脸原理之后，我们尝试着复现使用dlib来进行人脸的68个关键点检测，并利用普氏变换来进行换脸，之后我们又使用了dlib的81个关键点检测来进行换脸。

3.1 基于dlib68关键点

通过get_landmarks()函数调用dlib得到68个关键点的坐标，在得到两张图片的关键点之后，使用普氏分析法来旋转，翻译，规模化第一个图片的向量，使它们尽可能适合第二个向量的点（transformation_from_points()），其将输入矩阵转换为浮点数之后，每一个点集减去它的矩心，再将每一个点集除以它的标准偏差，使用Singular Value Decomposition计算旋转部分，利用仿射变换矩阵返回完整的转化。

之后，函数warp_im()调用OpenCV的cv2.warpAffine函数，将图像二映射到图像一。考虑到肤色的不同，使用correct_colour()函数修正覆盖区域的不连续问题，其主要思想是用第二张图片除以第二张图片的高斯模糊，然后乘以第一张图片的高斯模糊。用RGB缩放校色，而不是用所有图像的整体常数比例因子，因为每个像素都有自己的局部比例因子。这里的关键在于一个适当大小的高斯核。如果过大，内核之外区域像素被覆盖。太小，第一张图的面部特征将显示在第二张图中，并发生变色。

最后，用一个遮罩来显示最终显示的部分，函数get_face_mask()用标记矩阵生成了一个遮罩，两张图片都生成一个遮罩，通过一个element-wise最大值，将两个遮罩结合成一个。

以下为原图和使用该程序得到的换脸图

图 7: 后面分别是使用68关键点和81关键点得到的结果

3.2 关于dlib81关键点的探索

我们尝试这个方法的目的在于想解决deepfakes换脸不考虑额头的问题，想试着解决眉毛与额头之间的像素突兀，于是使用了81关键点检测并尝试强调额头部分，但结果如图所示，并没

有变得更加优秀，我们考虑原因是原代码的可迁移性不强，过度重视眼睛和鼻子的部分，而我们也没有进行很好地调参，所以结果不尽人意。

(a) 68点　　(b) 81点

图 8: 不同的检测模型

4 检测deepfakes

在前面我们已经对deepfakes的不足之处进行了分析，所以我们针对这些不足之处想到了眨眼检测和边缘检测的方法。

4.1 眨眼检测

正如之前所述，基于帧的换脸不会考虑帧之间连续的动作，比如眨眼，呼吸等动作，所以我们想到能否用眨眼次数或者眨眼动作的变化来检测deepfakes换脸的痕迹。如果是换脸视频，那么这个视频的眨眼频率应该和脸的所有者的眨眼频率是不同的。

我们使用眼睛纵横比(EAR)来判断眨眼，如图所示， 我们所关注的，就是在睁眼和闭眼的

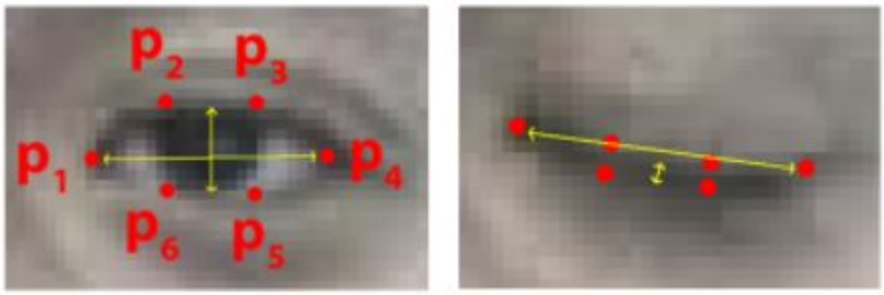


图 9: 人眼对应的六个特征点

时候，这六个特征点的坐标之间的关系，如图中的直线所示，长宽比在眨眼期间是在变化的，所以导出EAR的方程为：

$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2||p_1 - p_4||} \tag{1}$$

分子中计算的是眼睛的特征点在垂直方向上的距离，分母计算的是眼睛的特征点在水平方向上的距离。由于水平点只有一组，而垂直点有两组，所以分母乘上了2，以保证两组特征点的权重相同。我们不难发现，EAR在眼睛睁开时是基本保持不变的，在小范围内会上下浮动，然而，当眼睛闭合时，EAR会迅速下降。这也就是我们进行眨眼检测的原理。

在实现中，我们首先检测特征点，并取出左右眼对应的特征点，分别计算EAR，然后寻找左右眼的轮廓并绘制，若EAR小于阈值，则开始计算连续帧，只有连续帧计数超过EAR_CONS–EC_FRAMES_MIN时，才会计做一次眨眼。对于阈值的选取，可以根据经验设定，或者用两眼EAR的平均值来求出。

经过检测，我们得到如下结果： 根据以上结果，被替换者本身的眨眼频率为0.125，而替换

表 1: 眨眼测试的结果

视频	视频时长/s	左眼眨眼次数	右眼眨眼次数	平均眨眼次数/s
data_dst.mp4	64	9	7	0.125
H64_10000.mp4	64	14	27	0.320
H64_60000.mp4	64	12	16	0.218
SAE_10000.mp4	64	2	4	0.047
data_src.mp4	27	11	14	0.463

者的眨眼频率为0.462，我们换脸得到的结果也主要分布在0.125左右，与换脸者平常的眨眼频率相去甚远，可以说有一定的效果。但同时我们发现眨眼检测同样无法对侧脸进行检测，这种情况下，侧脸时的眨眼同样无法被捕捉，变相降低了眨眼频率，所以我们有必要计算未检测到脸的帧数并在计算眨眼频率时刨除。此外，眨眼频率受到诸多因素的影响，比如生气的时候更同意瞪着眼，我们希望以后可以结合这些情况进行更合理地量化分析。当然，对于某些特定情况，比如用同一个人的视频换脸，或者说只换嘴的情况，眨眼检测可能效果甚微。

4.2 边缘像素检测

在我们看来，换脸，肯定会有换脸的痕迹，所以我们参考了adobe公司研发的人工智能反P图技术，他们提出，每个像素点红黄蓝三原色与周围的像素有一定的关系，加入某一区域的图像与周围没有数字关系或者发生异常，就可以判定这张图片经过修改。

我们的想法是简化算法，对整张图片进行计算，并得到没有问题的像素点的比例，然后对所有帧得到的比例求平均值，若比例越高，则这个视频为未经处理的视频的概率越高，经过对data_dst.mp4和SAE_10000.mp4的计算，我们得到的概率分别为0.7344和0.7367，尽管两者的差距很小，但是如果考虑到视频中人脸占的面积大约在2.5%左右，而两张图片的差别仅仅在于人脸的替换，那么计算可得人脸部分的差距在9.2%左右，已经有了比较大的区别，当然，这可能与我们的迭代次数不足有关，也可能具有偶然性，但不失为一个可行的思路。但是，这个算法的不足之处在于，如果再次拍摄换脸后的视频，那么这种方法对于这种未做处理的视频可能是无效的。

我们还想对用deepfacelab提供的模块或者用人脸检测方法得到的人脸区域的概率值进行比较，但由于时间有限，这一部分的工作还没有完成。以后，我们也可能会尝试使用已有的数据

集，对算法进行优化之后，使用深度学习的方法进一步提高检测效率。

5 总结

感谢老师和助教们给我们提供了这个学习deepfakes的机会。经过一个学期对deepfakes技术的了解和学习，我们了解了这个技术的原理，尝试了不同的模型，学习了不同的调参，也提出了两种有一定效果的检测方法，但是距离吃透这个技术，摸清它的门路，还有很长的路要走。我们发现换脸与检测换脸就像魔高一尺道高一丈，在共同的进步当中。我们希望以后能够继续学习研究这个技术，尝试着在优化我们的方法或者使用别的检测技术的同时，对于deepfakes做出一点属于我们的贡献。

6 参考文献

[1]有没有技术可以判断一张图片是否被PS过？https://www.zhihu.com/question/19720234?sort=created

[2]https://github.com/iperov/DeepFaceLab

[3]眨眼检测 https://blog.csdn.net/hongbin_xu/article/details/79033116

[4]DeepFaceLab不同模型的参数含义 https://www.deepfacelabs.com/read-49-1.html