基于 TensorFlow 深度神经网络的人脸 68 关键点检测

摘要

从网络上获取数据集，编写并借鉴 python 代码进行数据预处理，生成可导入训练模型的数据格式。利用 TensorFlow 搭建深度神经网络并训练，得到 ckpt 文件，最后获取模型进行特征点的检测。

关键词—数据预处理、深度神经网络、人脸 68 关键点

1. 引言

人脸关键点定位旨在检测人脸上的一组预定义关键点，其吸引了业界和研究团体的广大兴趣。在 GitHub 搜索 facial landmark 会发现许多的研究团体已经做了很多很实际也很有趣的开发。然而，许多开发者仍旧是直接调用 Dlib 的库来实现该功能，经实践发现 Dlib 有很大的缺陷，在一些头部有较大俯仰角或是光照比较极端的情况下，其基本上标记不出结果。从这个角度说，人脸关键点定位在不受控制环境下仍是个未得到很好解决的问题[1]。阅读论文我发现，解决人脸关键点定位的最直接方法是将其视为基于图像的回归问题，即输入是图片，输出是坐标[1]。因此我可以依靠深度神经网络来解决这个问题。我首先从网络上收集数据集，然后进行数据的预处理，以让它可以方便的输入网络。之后我依据论文搭建深度神经网络，在服务器上进行训练，最终得到模型。得到模型时候，我将其提取出来，进行应用。

2. 获取数据集

经过在网络上的调研，我发现了 300-W、LFPW、HELEN、AFW、IBUG 五个数据集[2]。将其整合到一个文件夹 dataset 下，如 Figure.1 所示，经检测共有 4236 张图片，与之对应有 4236 个 pts 文件。在网上查找后得知[3]，pts 文件存储着数据集中标注好的 68 关键点坐标。

```
Total files: 8472
jpg: 2802; pts: 4236; png: 1434; Done!
```

Figure.1：文件统计结果

3. 数据预处理

在网上获取的数据集并不能直接用于训练，我们需要对其进行一些预处理来让数据集适合神经网络的训练[4]。网上下载的数据集是否可信是一个问题，我们需要进行一下验证。验证的方法也很简单，就是随机抽取一些图片与其对应的 pts 文件，将 pts 文件中存储的坐标读出来并画在图片上，来验证特征点坐标与图片的一致性[5]。经验证，如 Figure.2 和 Figure.3 所示，抽样样本特征点坐标与图片匹配，故认为数据集无误。

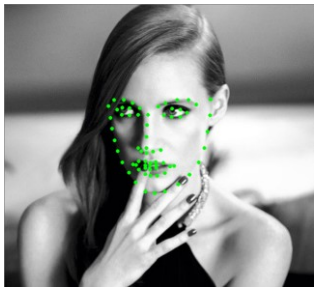

Figure.2

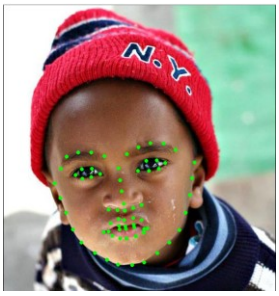

Figure.3

3.1. 人脸区域获取

输入神经网络的图片是固定大小的 128*128*3[1]，而输进网络里面进行训练的图片不是数据集里面的图片，而仅仅是图片中包含人脸的那块区域[6]。这样，神经网络的作用很明确，即找到所给人脸的特征点，而不用先进行一步确定人脸位置的操作。

很显然，要完成这个想法需要进行一下数据的预处理，即将数据集里的图片中人脸识别并提取出来。由于图片大小发生变化，相应的坐标值也要发生变化。

首先是人脸区域的获取。提取出来的人脸区域应当满足一定的条件[6]：

1、由于输入网络的图片大小是 128*128*3,，即人脸区域应为正方形。

2、尺寸大小合适

3、包含全部的 68 关键点

我从网上查找发现，opencv 开源了人脸检测器的代码，可以在其基础上进行实现。利用 opencv 人脸检测器提取出来的人脸不能够直接用于训练。如 Figure.4 和 Figure.5 所示，其会存在特征点超出其提取出的人脸区域以及提取出本来不存在人脸的区域的情况，并且所得的框均为长方形。

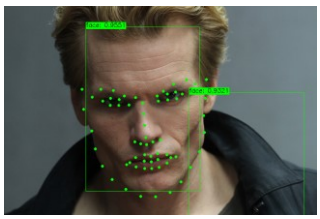

Figure.4：发生误检测

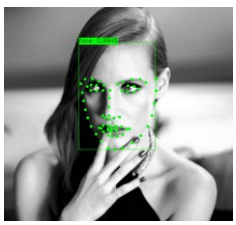

Figure.5：特征点超出范围

于是，需要对 opencv 人脸检测器得到的框进行处理。处理的过程如 Figure.6 所示，即先判断获取的 Boxes 是否包含特征点，如果不包含的话就舍弃掉。如果包含的话，就对该框进行操作，使其变为一个正方形，并且为了尽可能的使框均匀的框住特征点，我们把它下移一个常数 C 的距离，在这里我们令

$$C = height - width$$ [6]

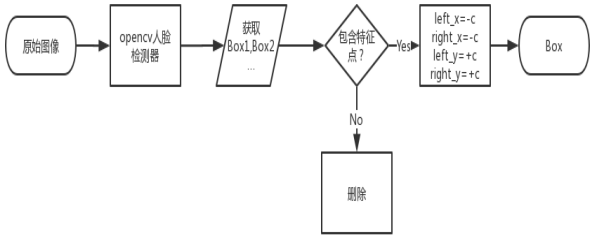

Figure.6：对 opencv 检测到的人脸区域的操作

经过上述过程之后，框变成了正方形，且只剩下一个框，包含所有的关键点，效果如 Figure.7 和 Figure.8 所示：

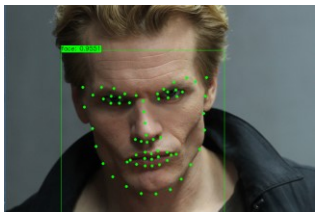

Figure.7：Figure.4 处理后

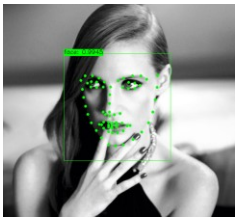

Figure.8：Figure.5 处理后

2.2. 数据类型转换

从网上了解到，TFRecord 文件是一种 TensorFlow 原生支持的数据格式[7]。TendorFlow 官方文档给出了三种数据来源的方式，其中，当数据量比较大的时候，其推荐使用 Standard TensorFlow format 来进行存储训练与验证数据，该格式的后缀名称为 tfrecord。因此，我们需要将我们的数据转换成 TFRecord 文件。

我们需要保存到 TFRecord 文件中的信息包括：图像、图像格式、图像尺寸、图像文件名、特征点坐标[8]。

经过上一部分的工作，我们只需要把框里的图片提取出来，并且新命名存储起来就可以了。这里要注意的是，特征点的坐标要发生相应的变化。而变化也很简单，就是用原坐标值减去所得框左上角顶点的横坐标与纵坐标即可。中间会有两种文件格式作为中间的数据格式，其皆为数据交换模式，即 Json 文件及 csv 文件[9][10]。整个处理流程参考博客[8]，整理如 Figure 9 所示：

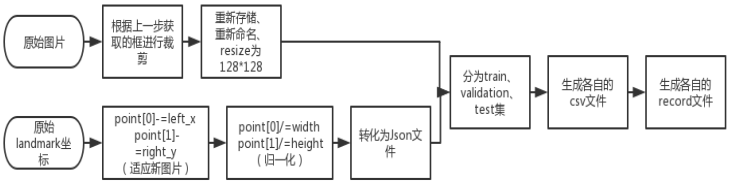

Figure 9：数据处理流程

至此，我们完成了数据预处理的操作，获取的 record 文件可以直接输进神经网络。

4. 搭建深度神经网络模型

论文[1]里的网络结构如 Figure.10 所示

input 128 × 128 × 3
conv 3 × 3 × 32
pool 2
conv 3 × 3 × 64
conv 3 × 3 × 64
pool 2
conv 3 × 3 × 64
conv 3 × 3 × 64
pool 2
conv 3 × 3 × 128
conv 3 × 3 × 128
pool 2
conv 3 × 3 × 256
full 1024
full 2n

Figure.10[5]：n 为 landmark 的数目，即为 68

按照这个结构，利用 TensorFlow 框架搭建网络模型，设置一些超参数：batch_size=32, num_epochs=50，steps=11350，采用 Adam 优化算法进行优化。我们用 L2 Loss 函数，其为平方差的和：

$$\mathcal{L} = \sum_{i=0}^{N-1} \|Y_i - f(X_i)\|_2^2.$$

5. 训练神经网络，获得模型

在服务器上进行训练，最终结果如 Figure. 11 所示：

```
INFO:tensorflow:loss = 0.00028890767, step = 10450 (10.119 sec)
INFO:tensorflow:global_step/sec: 10.6495
INFO:tensorflow:loss = 0.00023031363, step = 10550 (9.389 sec)
INFO:tensorflow:global_step/sec: 10.7368
INFO:tensorflow:loss = 0.00027565582, step = 10650 (9.314 sec)
INFO:tensorflow:global_step/sec: 10.4094
INFO:tensorflow:loss = 0.00018618794, step = 10750 (9.607 sec)
INFO:tensorflow:global_step/sec: 10.759
INFO:tensorflow:loss = 0.00018063933, step = 10850 (9.296 sec)
INFO:tensorflow:global_step/sec: 10.5542
INFO:tensorflow:loss = 0.00021134583, step = 10950 (9.474 sec)
INFO:tensorflow:global_step/sec: 10.6369
INFO:tensorflow:loss = 0.0001909533, step = 11050 (9.402 sec)
INFO:tensorflow:global_step/sec: 10.5273
INFO:tensorflow:loss = 0.00016118107, step = 11150 (9.498 sec)
INFO:tensorflow:global_step/sec: 10.7416
INFO:tensorflow:loss = 0.00016846014, step = 11250 (9.309 sec)
INFO:tensorflow:Saving checkpoints for 11350 into model/model.ckpt.
INFO:tensorflow:Loss for final step: 0.00024562486.
```

Figure. 11

可见经过 11350 steps 之后，其训练集的 loss 为 0.00024562486。

在测试集上检测的结果如 Figure. 12 所示，loss 为 0.001724738。

```
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 11350: model/model.ckpt-11350
{'MSE': 0.041560255, 'loss': 0.001724738, 'global_step': 11350}
```

Figure. 12

我们随机抽取测试集的图片打印出来，发现效果还不错。如 Figure. 13 和 Figure. 14：

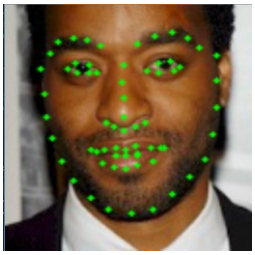

Figure. 13

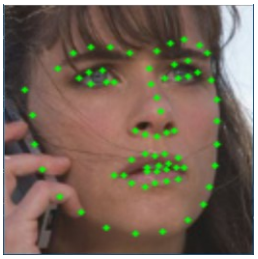

Figure. 14

6. 模型的重构与使用

最开始计划导出. pb 文件出来用于推演[11]，但是最终发现利用 ckpt 的文件足以重构模型。下面是我在网上找的一张图片，Figure. 15 是 Dlib 的输出，Figure. 16 是经 ckpt 重构模型的输出，很显然后者效果要比前者好。

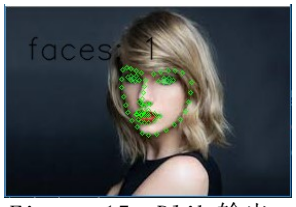

Figure. 15：Dlib 输出

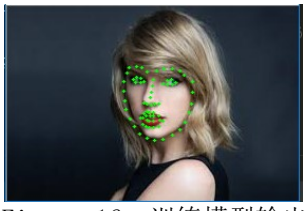

Figure. 16：训练模型输出

7. 总结

本次实验，我完成了从数据集的搜集到预处理，再到网络的搭建、训练以及模型的提取、重构、使用整一套流程，最终实现了优于现有 Dlib 库的 68 关键点检测。在完成的过程中大量的参考了*尹国冰的博客*，以及一些论文。

8. 参考文献

[1] Haoqiang Fan, Erjin Zhou, Approaching human level facial landmark localization by deep learning, Image and Vision Computing, Volume 47, 2016, Pages 27-35, ISSN 0262-8856, https://doi.org/10.1016/j.imavis.2015.11.004.

[2] https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/

[3] https://blog.csdn.net/sjtu_edu_cn/article/details/51830738

[4] https://yinguobing.com/facial-landmark-localization-by-deep-learning-data-and-algorithm/

[5] https://yinguobing.com/facial-landmark-localization-by-deep-learning-data-collate/

[6] https://yinguobing.com/facial-landmark-localization-by-deep-learning-preprocessing/#fn1

[7] https://yinguobing.com/tfrecord-in-tensorflow/

[8] https://yinguobing.com/facial-landmark-localization-by-deep-learning-tfrecord/

[9] https://www.cnblogs.com/bigberg/p/6430095.html

[10] https://blog.csdn.net/abcpanpeng/article/details/1718149

[11] https://yinguobing.com/facial-landmark-localization-by-deep-learning-save-model-application/