记一次坎坷的算法需求实现：轻量级人体姿态估计模型的修炼之路（附MoveNet复现经验）

原创 CV开发者都爱看的 极市平台 2021-09-23 22:00:00 手机阅读

收录于话题
#姿态估计

作者丨Fire

编辑丨极市平台

极市导读

本文记录了作者实现轻量级人体姿态估计模型的全过程，从方案的选取到尝试复现等，详细的叙述了一个项目需求完成的整体思路，并附有谷歌开源的MoveNet的复现经验。本文能给处在CV各个阶段的朋友们带来帮助！ >>加入极市CV技术交流群，走在计算机视觉的最前沿

一、需求背景

这天接到个新需求，需要实时检测自然场景下目标人体的关键点位置。

从算法工程师的角度来拆解下需求：

1、检测人体关键点位置，就是人体姿态估计任务嘛；

2、要实时，那么就是终端部署，服务端那传输延时就不考虑了。对了咱们硬件不大行，所以肯定是要轻量级模型的，分辨率也不能太大，剪枝量化蒸馏三件套也要做好打算；

3、“自然场景下的目标人体”，意思就是场景下可能有多人，但是我们只需要一个目标的关键点，要考虑如何区分（这点后面再展开说）。

二、方案探索

2.1 初见

之前的项目经验主要是人脸相关的分类、检测、分割以及OCR等，虽然没做过人体姿态估计，但是需求分析完，我的第一感觉是“应该很简单”，这份底气来自于之前做过的人脸关键点检测项目，当时的经验是，处理好数据、使用合适的Loss，随便用个剪枝后的轻量级模型都可以达到很高的精度，而且在嵌入式板子上能跑到10ms以下（都不需要PFLD之类的）。而人脸关键点有68点，人体也就17个点，不是简单多了。

于是直接拿出之前人脸关键点的代码，修改亿点点细节（调整模型输出、准备训练数据、修改DataLoader等），用少部分数据先跑跑看看。数据就用COCO和MPII就差不多了，后续可以自己从网上爬取一些来扩充，当然如果有目标场景的数据就更好了。

工欲善其事，必先抓只小白鼠。也就是需要一个评测指标，分类常用Accuracy、F1，检测常用mAP，而人体姿态估计又有所不同，请教了谷哥（Google），**一般是用PCK（PCKh）和OKS+mAP，简单来说，前者就是计算predict和label点的距离再经过head size normalize，后者就是用类似于目标检测中IOU的OKS计算相似度，然后再算AP。**虽然前者在学术界目前已经很少使用（主要是刷榜刷得太高了），但是我们场景只需要单人，加上工程化简单，肯定是选择PCK，且考虑数据不一定都有head的标注，以及我们场景目标大小比较一致，最终决定使用自定义PCK，160分辨率下，距离小于5则算正确，其实PCK也算作一种acc，所以后面用acc指代。

第一次跑完，验证集acc达到了 0.81，感觉还行，于是开始往丹炉疯狂加料（加数据、加Data balance、微调loss、不同关键点loss设置不同weight、怀疑特征提取不够甚至把FastSCNN的backbone都拿来了、各种调参等等），一顿操作下来，终于有了一点提升：val-acc达到了0.9869。

这么高，看来是成了，部署上板测试之前，习惯性在自己电脑上先跑跑，准备好可视化demo一看，傻眼了，除了head之类的比较准，手稍微一动就很容易检测不到。

一腔热血终于被冰冷的现实击溃，看来这个"应该很简单"的任务并不简单...自己挖的坑，跪着也要填完。稍微总结下失败的教训，然后准备老老实实从零开始。

人脸关键点检测直接回归坐标点的效果好，很大原因在于特征分布比较集中。首先是人脸大小比较一致，其次是人脸是刚体，各个关键点相对位置也比较固定，最后是除了侧脸，基本不会遮挡，目标特征也比较清晰。相较之下，人体大小分布就不一致，且各个关节活动范围很大，相对位置不固定，还存在各种位置遮挡和服装遮挡的情况，相比来说任务难了很多。同时由于关键点的分布范围太广，单纯用全连接层去回归，很有可能出现某一部分神经元训练的比较好，另一部分比较差，也就导致泛化能力很差。

2.2 反思

直接回归关键点的思路行不通，那就打开国门睁眼看世界——看看现在的主流方案。这篇2020年的综述就挺不错的：Deep Learning-Based Human Pose Estimation: A Survey。如同华山派著名的气宗和剑宗之争，人体姿态估计也有不同的派别，而且打得更热闹，还分了两个方向：

- **目标学习形式：**直接回归点坐标 VS 回归Heatmap
- **整体检测流程：**Top-Bottom VS Bottom-Up

原来早期的人体姿态估计就是直接回归关键点，看来我也只是在重复前辈的路而已。后来发现性能不好，于是改为回归heatmap的方式来训练。简单来说就是把原图缩小几倍得到特征图，这个特征图有关键点的位置值就是1、背景就是0，因为只设1太稀疏了，所以把1改为一个高斯核，这样也符合实际语义。

确定好学习形式后，再看看检测流程。上面我的方案其实就属于Top-Bottom，也就是先检测人，再裁剪出来对每个人去检测关键点；而Bottom-Up则相反，直接检测出所有的关键点，再依次组合成一个个人。这里和目标检测中的one-stage和two-stage其实有点神似，一般来说，Top-Bottom方式精度高，但是速度慢一点，而Bottom-Up速度快，但是精度差一点。

同时了解了一下目前一些优秀的开源方案（主要侧重一些相对轻量级模型，因此就不包含HRNet之流了）：**AlphaPose、OpenPose、Lightweight OpenPose、BlazePose、Simple Baselines、Fast Human Pose Estimation、Global Context for Convolutional Pose Machines、Simple and Lightweight Human Pose Estimation**等，那就来吧。

2.3 尝试

2.3.1 浅尝辄止

对上面提到的各种潜在可行方案，分别进行测试。

1. 修改上文的我的人脸关键点回归模型（Top-Bottom），把输出改为heatmap，相应调整数据处理和loss，训练完效果一般，val-acc85；

2. AlphaPose、OpenPose都属于比较大的模型，且GPU速度也只是勉强实时，我们是端侧+CPU+低性能板子，估计1秒以上了，所以暂时不考虑，同时发现有个Lightweight OpenPose（Bottom-Up）的项目，直接拿来上板跑了下，360ms，有点希望；
3. BlazePose，这里直接用了TNN的模型，同时发现他们还开源了腾讯光流实验室的一个姿态估计模型（Top-Bottom），比BlazePose要好，于是直接测试了后者，上板200ms，且除了转换后的模型没有什么其他资料；
4. Simple Baselines（Top-Bottom），用预训练模型训练，val-acc在0.95左右还行，就是换成轻量级网络后（mobilenet-v3），精度掉到0.9左右，且可视化很差；
5. Fast Human Pose Estimation（Top-Bottom），相比于沙漏网络，stage砍半，channel砍半，就没其它改变了；使用了蒸馏，loss是两部分，一部分是老师的预测结果，一部分是GT；效果还行，720ms略慢；
6. Global Context for Convolutional Pose Machines（Top-Bottom），提供的预训练模型，可视化效果还行，400ms左右；
7. Simple and Lightweight Human Pose Estimation（Top-Bottom），可视化效果还行，250ms；

2.3.2 集中火力

经过综合考虑，最后选择了Lightweight OpenPose来做优化。

一是它是Bottom-Up的模型，不需要额外训练人体检测器（对应额外的数据处理成本、模型训练成本以及推理耗时）；二是可视化来看，它的精度也是属于表现最好的几个。

要了解Lightweight OpenPose，那么就需要先提下OpenPose，它使用了VGG作为特征提取，加上两个header，分别输出关键点的heatmap和PAF的heatmap，通过多个stage迭代优化（每个stage的输入和输出都是这两个heatmap），最后通过MSE进行优化。关键点的heatmap好理解，PAF是什么呢？这是论文提出的part affinity fields，一般翻译为亲和力向量场，它代表了两个关键点之间的连接信息，个人觉得可以直观理解为关键点之间的骨骼信息，实现的时候是求两个关键点之间的单位向量（代表了方向），然后给对应的heatmap位置赋值。最后把多人检测问题转化为二分图匹配问题，并用匈牙利算法求得相连关键点最佳匹配。

而Lightweight OpenPose作者测试发现原openpose的refinement stage的5个阶段，从第一个refinement stage1之后，正确率没有提升多少，但是运算量就加大了。并且还发现了refinement stage网络的关键点定位(heatmap)和关键点组合(pafs)两部分的两条分支运算，有不少操作是一样的，造成运算冗余。于是主要进行了以下优化：

1. 新的网络设计。只采用initial stage+refinement stage两个阶段网络。（此处，我觉得到refinement stage2阶段的性价比比较高，refinement stage1的正确率还是有点低，尽管运算量不大。但是refinement stage2再加了约19GFLOPs的同时正确率就提高了3%左右，性价比更好。）
2. 更换轻量级backbone。用MobileNet替换,并用空洞卷积优化网络。并且进行了一系列对比实验，发现MoblieNet v1比MobileNet v2效果居然更好。
3. refinement stage中的关键点定位(heatmap)和关键点组合(pafs)部分的网络权值共享，减少操作计算量。最后两个卷积层才分出两个分支分别用来预测关键点的定位(heatmap)和关键点组合(pafs)。

Table 4: Lightweight backbone selection study (the initial and refinement stages have original OpenPose design).

	AP, %	GFLOPs
MobileNet v1 *(cut to conv4_1)*	37.9	23.3
Dilated MobileNet v1 *(cut to conv5_5)*	42.8	27.7
Dilated MobileNet v1 *(cut to conv5_6)*	43.2	31.3
Dilated MobileNet v2 *(cut to conv6_3)*	39.6	27.2

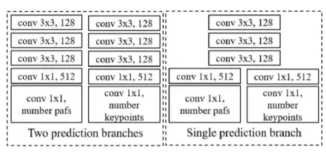


Figure 2: Original two prediction branches and proposed single prediction branch for the initial stage. We also apply this scheme for the refinement stage.

4. 原openpose的refinement stage中都是使用7x7的卷积核。作者却使用了三个连续的1x1, 3x3, 3x3的卷积核来代替7x7,并且最后的3x3是使用了空洞卷积, dilation=2。另外由于用3个卷积核代替原来的一个卷积核,网络层数变得很深,所以作者又加上了一个residual连接。

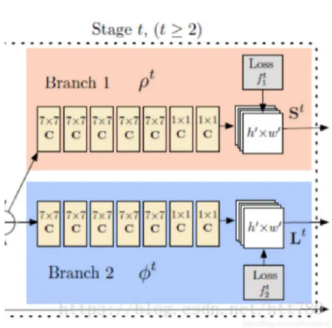


Figure 3: Design of convolutional block for replacement convolutions with 7x7 kernel size in refinement stage.

研究了下Lightweight OpenPose源码，继续开始往丹炉疯狂加料（改backbone、剪枝、加数据、调分辨率、蒸馏、各种调参等等），一顿操作下来，得到了模型：速度80ms，acc0.95（对应的蒸馏teacher acc0.98）

速度勉强能接受，精度还行，但是可视化测试稍微有点不准，可能跟数据也有一定关系。

三、峰回路转

3.1 转机

就在我犹豫要不要继续深入优化Lightweight OpenPose的时候，无意间刷到了这样一篇文章：《实时检测17个人体关键点，谷歌SOTA姿态检测模型，手机端也能运行》。文章介绍了谷歌今年五月开源的一个轻量级人体姿态估计模型MoveNet，在tfjs有对应的api。

众里寻他千百度，蓦然回首，那人却在灯火阑珊处。这不就是上天为我准备的吗？赶紧用网页端测试了下，感觉还行，加上之前有转bodypix的tfjs模型到tf模型的经验，准备直接下模型本地跑跑，幸运的是，这次的MoveNet不仅有tfjs模型，还有tflite模型，这下方便多了。

通过官方博客的只言片语以及Netron可视化网络，对模型有了初步了解，官方宣称是基于CenterNet做的修改，但是修改了很多（The prediction scheme loosely follows CenterNet, with notable changes that improve both speed and accuracy）。具体来说，使用了mobilenetv2+fpn作为backbone，输出四个header，最后经过后处理输出一个最靠近图片中心的人的关键点。由此可知，它也是属于Bottom-Up的流派，同时通过关键点回归的范围进行加权，只输出最靠近中心的一个人的关键点，和我们的场景绝佳匹配，因为就Lightweight OpenPose而言，其余人的PAF信息其实也是多余的。

先测测速度吧，首先尝试上板安卓直接加载tflite模型，报错Didn't find op for builtin opcode 'RESIZE_BILINEAR' version '3'，之前用的org.tensorflow:tensorflow-lite:2.0.0，尝试2.3.0后可以了。速度150ms左右一帧，连续跑在120ms左右。然后尝试一些推理框架（比如Tengine、NCNN、MNN等），这里我习惯使用TNN。结果tflite转tnn和pb转tnn均failed，debug看是不支持ARG_MAX算子，这是后处理里面的，于是自己简单写了个去掉后处理的mobilenetv2+fpn+4个header的模型转tnn，上板测试80ms左右。

考虑到Lightweight OpenPose已经优化了很多，继续压榨空间有限，且上文提到有冗余输出，而这个MoveNet直接就能跑到80多ms，且只输出一个人的关键点，还有谷歌背书，因此痛下决心，准备抛弃Lightweight OpenPose采用MoveNet的方案，尝试复现它。同时我还准备了Plan B：如果复现不了，就直接导出tflite的backbone权重，然后新建一个pytorch模型加载权重，得到模型推理输出后通过C++代码实现后处理。这样的缺点是没法用自己数据训练，无法迭代优化，且直接使用官方提供的预训练模型可视化来看精度也没有达到特别高。

3.2 复现

好吧，那就开始准备撸起袖子复现MoveNet了。

只根据一个模型结构要复现一个模型训练，说难不难，说简单也不简单。模型结构搭建照猫画虎即可，关键是数据如何构造、Loss如何选择没有任何信息，甚至还隐藏了大量细节，比如高斯核的构造、loss权重的设置、优化器等各种超参数设置。道阻且长，行则将至，只能一步一个脚印了。

事先说明，以下所有分析均是主观猜想，只是经过实验验证效果也不错，不代表谷歌官方原始实现就一定是这样，仅供参考。

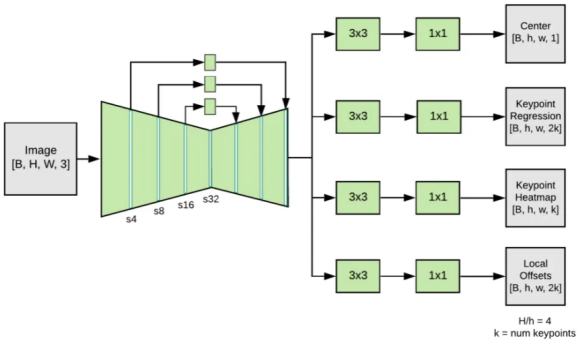
3.2.1 模型结构

要复现一个模型并不难，很多模型有论文、有官方代码或者有其他人复现的代码、哪怕是各种专栏博客的拆解分析，都能帮助很多，遗憾的是，MoveNet都没有，唯一能参考的只有两个东西：谷歌官方博客的介绍，以及TFHub提供的训练好的模型。这也是这篇文章分享的初衷。

观察模型结构，主要分为三部分：backbone、header、后处理。

1.Backbone前面说过了很简单，即mobilenetv2+fpn，为了保证可控性和灵活性，我没有使用现成的一些mobilenetv2的实现，而是自己一层层自己搭起来的（其实也没有太大必要，后续换backbone我也是直接在现成的模型代码上改的）；

2.Header的构建就更简单了，输入backbone的特征图，经过各自的几个卷积层，最后输出各自维度的特征图即可，难点在于要理解每一个header的含义，整体结构参考如下：



经过不断的分析、猜想、测试，现在直接分享得到的结论吧。四个header我们分别命名head_heatmap，head_center，head_reg，head_offset以便说明：

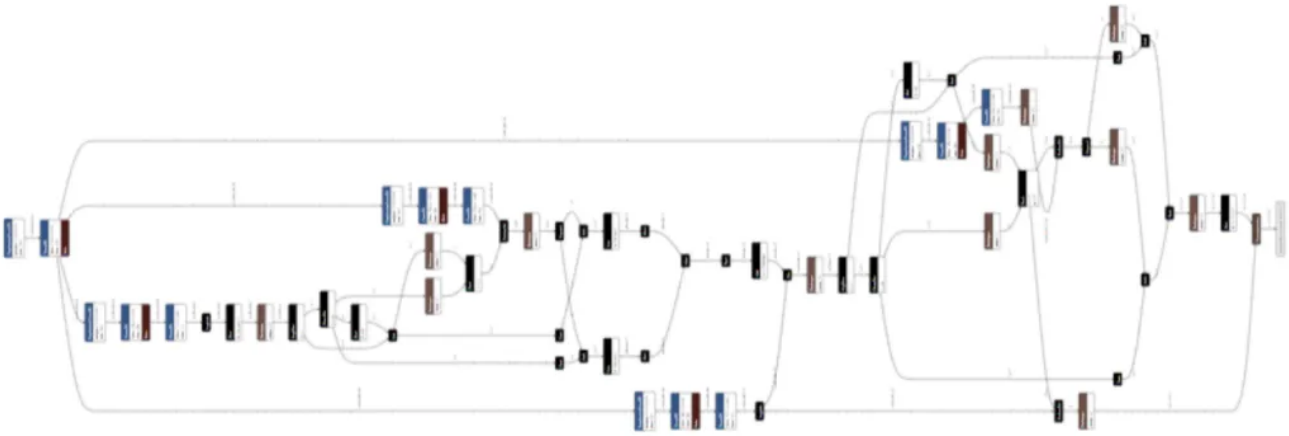
- head_heatmap的维度是[N,K,H,W]，n是batchsize，训练时自己指定，预测时一般为1；K代表关键点数量，比如17；H、W就是对应的特征图了，这里输入是192x192，降采样4倍就是48x48；它所代表的意义就是当前图像上所有人的关键点的heatmap，注意是所有人的；

- head_center的维度是[N,1,H,W]，这里的1代表的是当前图像上所有人的中心点的heatmap，你可以简单理解为关键点，因为只有一个，所以通道为1；它所代表的意义官方博客说是arithmetic mean，即每一个人的所有关键点的算术平均数，但是我实测这样效果并不好，我自己最终是取得所有关键点得最大外接矩形的中心点，当存在一些较远的关键点的时候，可能算术平均数可以很好的训练大部分距离近的点，但是对较远的点效果差点，而我比较关注手腕这种较远的点，按我这么取对每一个点学习起来差不多，这个就仁者见仁智者见智了，以自己场景实验结果为准；

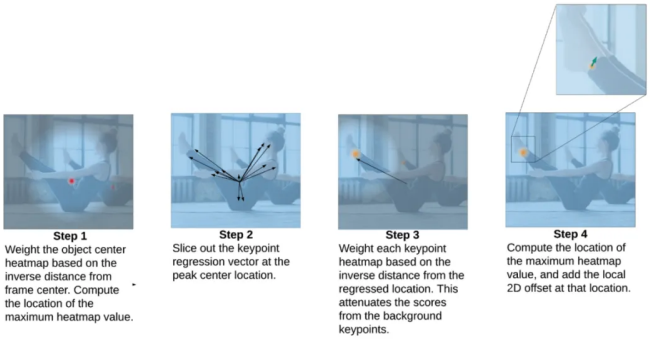
- head_reg的维度是[N,2K,H,W]，K个关键点，坐标用x,y表示，那么就有2K个数据，就是对应这里的2K通道；那么数据如何构造呢？根据模型结构的拆解，就是在每个人的center坐标位置，按2K通道顺序依次赋值x1,y1,x2,y2,...，这里的x、y代表的是同一个人的关键点相对于中心点的偏移值，原始Movenet用的是特征图48尺寸下的绝对偏移值，实测换成相对值（即除以size 48转换到0-1区间）也是可以的，可以稍微加快收敛，不过几乎没有区别；

- head_offset的维度是[N,2K,H,W]，通道意义一样都是对应K个关键点的坐标，只不过上面是回归偏移值，这里是offset，含义是我们模型降采样特征图可能存在量化误差，比如192分辨率下x=0和x=3映射到48分辨率的特征图时坐标都变为了0；同时还有回归的误差，这里一并训练了；

3.后处理相对来说比较麻烦，因为有各种奇奇怪怪的操作，这一步只有结合Netron可视化一步步大胆猜测，结合官方博客认真分析，结构如下，可以看到不是常规的CNN结构。



官方介绍如下，也解开了不少疑团。



也不卖关子了，直接说明后处理流程吧。

首先对于head_heatmap和head_center，需要求一个sigmoid，这里也可以写到网络结构中，主要是保证取值范围，因为后面要乘位置权重。

然后对于head_center，我们乘以一个位置权重矩阵，大小也是48x48，值可以通过Netron导出谷歌的设置，通过Numpy+OpenCV加载可视化可以看出是一个中心点高亮的高斯核。对于我们检测出所有的可能的中心点，通过乘以这样一个中心位置加权矩阵，可以提高靠近中心的人物权重，因为最终我们只取一个最靠近图像中心的。

乘以权重后，就是常见的argmax操作求出最大值点的坐标，这就是最终检测目标的中心点。拿到这个坐标就可以去head_reg的2K个通道对应坐标位置取出2K个值，然后加上中心点坐标，这就得到粗略的关键点位置了。我一开始以为这里是检测的主力输出，但是后续发现，这里的关键点其实很不准的，因为只是粗暴的回归坐标偏移。

它的实际作用是，对于每一个粗略的关键点坐标，咱们再生成一个以这个坐标点为中心的权重矩阵，这里也没用高斯核了，直接0-47等差数列构造即可。构造好一个粗略关键点坐标最小、周边递增的权重矩阵后，再加上一个常数（比如1.8）防止除0，然后就把一开始的head_heatmap除以这个权重矩阵，再分别通过K个通道求argmax，得到精细化的关键点坐标。为什么要这么做呢？**其实思想很简单，就是通过粗定位的关键点范围，然后加权取去找最可能属于当前人物的关键点，因为前面说过了我们是Bottom-Up的方案，head_heatmap检测的是所有人的所有关键点。通过这样一步操作，就可以取出最靠近中心点的人的关键点了。**

最后再根据坐标点去head_offset对应坐标位置取出offset的值加上即可。置信度就是head_heatmap的sigmoid后的值。

3.2.2 数据处理

终于分析完模型结构和后处理流程，但是依旧不能开始训练，因为我们还要喂数据、定目标函数。

数据构造其实弄明白后处理流程后也不难了，一个值得注意的问题是heatmap的高斯核如何构造。一开始我使用的是cv2.GaussianBlur生成的，可视化看过还行，后来debug发现一个问题，单个点没影响，但是当存在多个接近的关键点时，互相的高斯核极大值会受影响，可能变成比1小的值。于是干脆拿了CenterNet和Lightweight OpenPose中的高斯核生成来尝试，前者就是经典的高斯核，后者是按照指数曲线生成的，最终选择了后者。因为MoveNet后处理中，存在一个乘权重然后取最大值的步骤，为了增

加不同点的区分度（比如隔得远的点置信度为1，近的0.9，这时候应该取近的0.9这个点），那么应该增大不同距离点的权重差，但是为了减少同一个点的heatmap范围内变中心点改变（比如同一个高斯核内最大值1，次大0.9，但是0.9更靠近中心，这时候还是应该取1），那么应该减少相邻点间隔。在这两点矛盾的情况下，权重矩阵又是谷歌提供的已知的值，比较合理的一个方案就是选择指数曲线生成的热力图，即增大同一个高斯核中不同点的差值；最后还要注意高斯核的大小半径，可以当作超参数来调整，建议覆盖到原图上分析是否合理，最好根据不同的目标尺寸设置不同的半径大小。

另外一个细节是，如果只给中心点坐标对应的head_reg上那一个点赋值，每次去取回归值的时候，由于只有一个点，学习起来比较困难，因此我是给周围一圈也赋值了，只是要注意赋值坐标大小要对应加减。

3.2.3 损失函数

一般来说，对于这种heatmap形式的数据，比较常用的是L2 loss，比如Lightweight OpenPose，而CenterNet使用的是Focal loss，其实Focal loss也是从L2 loss优化而来，只是根据类别加权处理类别不平衡的问题，根据置信度加权处理难易样本的问题。

最终我采用的是加权版的MSE，相当于平衡了正负样本，因为heatmap+高斯核的方式背景占比很大，但是没有加Focal loss中的gamma，因为实验效果不好。这个加权也很好实现，因为我们的label就是取值0-1的矩阵，直接乘以k然后加1，就变成取值1到k+1的范围，即取值1的位置对应权重k+1，取值0也就是背景对应权重1，这个k可以根据数据情况自己调节，参考取值5-10，参考代码如下：

```
loss = torch.pow((pre-target),2)
weight_mask = target*k+1
#gamma = torch.pow(torch.abs(target-pre), 2)
loss = loss*weight_mask #*gamma
```

最终head_heatmap和head_center采用了加权MSE，head_reg和head_offset采用了L1 Loss，也是参考的CenterNet。

最后需要考虑的就是不同loss的权重问题，前期调试的时候由于量级存在差异（head_reg）是绝对坐标差值，我是设置的head_heatmap:head_center:head_reg:head_offset=1:1:0.1:1，最终优化好之后，直接采用1:1:1:1。

至此，整个MoveNet就已经复现差不多了，跑出来的acc能达到95，再稍微修复了一些bug，优化了一些细节，基本能跑到97了。但是直接跑视频demo可视化还是不是特别满意，因此需要以此为baseline，深度优化。

四、炼丹之道

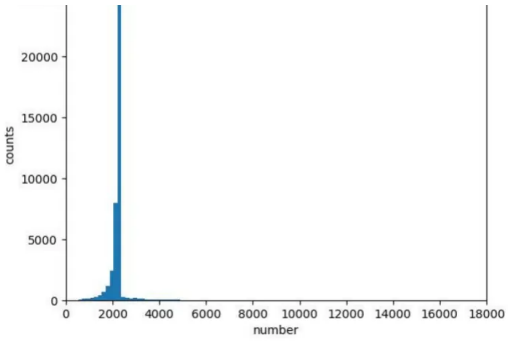
既然有了baseline，训练代码也搭建好了，就是喜闻乐见的炼丹环节了。

其实复现MoveNet的过程中就包含了一些优化，比如loss选择、高斯核生成，因为一开始效果不好，你不知道是复现的不对还是没有优化好。这里数据迭代清洗、一般的数据增强就不多说了，就说一些印象比较深的吧。

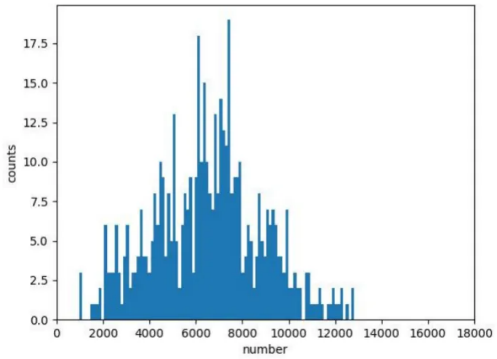
我的建议是先通过可视化分析+实验验证确定好数据增强方案，然后再进行模型结构的调整，最后进行loss优化以及各种超参数调整，这样可以一定程度上避免改了东墙测西墙的重复实验。

首先是数据扩充，谷歌本身除了使用清洗后的COCO数据集，也自己从youtube上爬了一些瑜伽、健身之类的视频然后生成图片和标注，因为原始COCO、MPII之类的图片，还是缺少一些瑜伽之类的特殊的姿态的，这就会造成数据不平衡（类似人脸关键的中的张嘴闭眼），于是除了考虑代码中加入数据平衡，最好的方式也是扩充一些数据，因此我也从B站上下了一些瑜伽、健身、体操、舞蹈等视频标注；

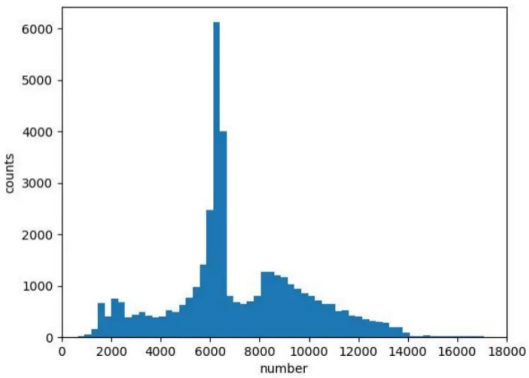
其次是数据增强有两点值得一提，**一是常见的镜像翻转**，分类任务中翻了就翻了，目标检测中要把目标框的横坐标同样翻转，而人体姿态估计更进一步，对关键点横坐标翻转之后，还要修改对应关键点的顺序，比如左手腕的图片水平镜像翻转后，它就不再是左手腕而变成右手腕了，这是一个小坑，不过阅读一些开源代码应该也能看到相应的处理；**第二点就是随机裁剪crop增大目标范围**，主要是初步验证的时候发现泛化效果不好，经过简单数据分析，发现训练集的关键点区域面积集中在2000附近：



再看看目标场景采集的几百张图片的关键点区域面积分布，范围是2000-12000，且峰值在6000-8000：



那么就可以针对性得调整随机crop的概率、crop的范围等，最后得到训练集数据增强后的分布如下，可以看到相比原始分布已经好很多了：



接着就是模型结构的优化，谷歌原始方案居然是使用的MobilenetV2，有点奇怪，据个人经验来说，这几个常见的轻量级模型效果一般排序是MobileNetV3≈ShuffleNetV2 >= MobilenetV2 >= MobilenetV1 ≈ShuffleNetV1，如果谷歌碍于自家产品不使用Shufflnet能理解，那为什么不用MobileNetV3呢？分别加上FPN上手跑跑，原始MoveNet使用的FPN分别融合了四个特征层，而由于网络结构block的差异，MobileNetV3和ShuffleNetV2我均只融合了三个特征层。经过大量实验测试，以及一些经验化剪枝，最后选定的是ShuffleNetV2+FPN，宽度因子为0.75，同时对通道数做了一定的剪枝。

同时把Sigmoid换成了一个近似实现：0.5 * (x/(1+torch.abs(x)))+0.5，训练精度差不多，且速度略快几毫秒，而且还顺便解决了推理工具结果不对齐的问题。

后来等模型训练过程中刷到了一篇文章《姿态估计技巧总结》，尝试了其中一些方案，其中Bone Loss还是有提升的，AID也有轻微提升不过不明显，Automatic Weighted Loss对此模型没有明显效果，尝试了Mish速度下降很多，也就没有训练了。

最后就是后处理的代码实现，因为涉及一些矩阵运算，本来是打算用C++版的OpenCV的Mat运算来做的，后来发现其实涉及的矩阵运算也就是一些element-wise操作，且特征图大小也就48*48，于是干脆用一维数组来做了，一次循环就可以搞定乘权重和记录极大值坐标，时间复杂度O(1)。

至此，整个项目也就接近了尾声，自己挖的坑终于填的七七八八，最终在验证集、测试集上acc都达到了99%（都这么高了，也就没打算蒸馏了），速度在嵌入式CPU上能跑到60+ms。可视化效果也还不错，不过依旧有提升空间，鉴于验证集已经很高了，后续主要的优化方向应该是扩充数据了。

五、总结

总结一下，一开始不熟悉人体姿态估计领域，走了一些弯路。同时在方案选择中也做了不少貌似无用的尝试，不过也正是对Simple Baselines和Lighweight OpenPose的熟悉，才让自己有机会深入了解人体姿态估计的相关流程，从而根据一个MoveNet的模型文件就复现出整个训练流程。最终经过优化，精度接近饱和，速度也从原始Movenet纯backbone跑80ms提升到了包含后处理整个流程达到60多ms。

最后惯例来个展望吧，时间有限，还有很多可以尝试的东西，列出一些供参考：

1. 我的场景首先要保证速度可以接受，其次才是精度，因此这套方案的精度还有很大的提升空间，如果是用高端手机来跑、甚至是GPU终端，换用容量更大的backbone，应该可以达到很高的精度；
2. 尝试一下DSNT、DARK的方案；
3. 尝试下Adversarial Semantic Data Augmentation for Human Pose Estimation论文提出的A数据增强，mpII数据集评测榜达到94.1%，貌似目前最高的；
4. 量化感知训练（QAT），因为PyTorch的QAT貌似不支持导出onnx，也就无法转换到推理框架使用，因此没有做。而训练后量化实际加速很少，掉点明显，分类任务还好，MoveNet有精细化的heatmap乘以权重的操作，误差影响更大所以也不考虑；后续可以考虑使用Tensorflow来做QAT，因为本身谷歌MoveNet就是TF训练的模型；
5. 试试新出的MicroNet，今年的轻量级网络，关注了一段时间最近代码也开源了，论文实验表示甩了Mobilenetv3和ShuffleNetv2一大截，有空可以玩玩；

才疏学浅，有不对的地方欢迎指出，也欢迎友好交流讨论。

参考资料

- [1]Learning Human Pose Estimation Features with Convolutional Networks
- [2]DeepPose Human Pose Estimation via Deep Neural Networks
- [3]Joint Training of a Convolutional Network and aGraphical Model for Human Pose Estimation
- [4]Joint Training of a Convolutional Network and aGraphical Model for Human Pose Estimation
- [5]Convolutional Pose Machine

极市干货

神经网络：视觉神经网络模型优秀开源工作：timm库使用方法和最新代码解读

技术综述：综述：神经网络中 Normalization 的发展历程 | CNN轻量化模型及其设计原则综述

算法技巧（trick）：8点PyTorch提速技巧汇总 | 图像分类算法优化技巧

阅读原文