CS159–Programming

Report for

“Teach you Tai Chi”

(教你打太极)

Group name: Serious_Mr.Hawking

Members: 严虞炜，霍博，金子翔

Teacher: BaoYang

目录

1——金子翔
2——严虞炜
3、4——霍博

1 背景

1.1 选题原因

疫情期间，体育课从线下转移至线上，看视频、跟着视频练习动作成了体育课的日常。而跟着视频练习动作时，没有了老师的线下指导、纠正，同学们难以自我判断自己的动作是否标准，是否正确，困惑、不自信由此产生：我的动作到底有没有做错？我的动作到底标不标准？尽管，老师可以通过观看学生拍摄并上传的视频给予学生一些指导意见，但这个过程存在诸多问题：

①纠正过程工作量巨大，包括下载视频、观看分析视频、编写文字反馈。

②纠正效率不高，难以将多维的、量化的动作纠正意见以简单的文字的形式反馈给学生。

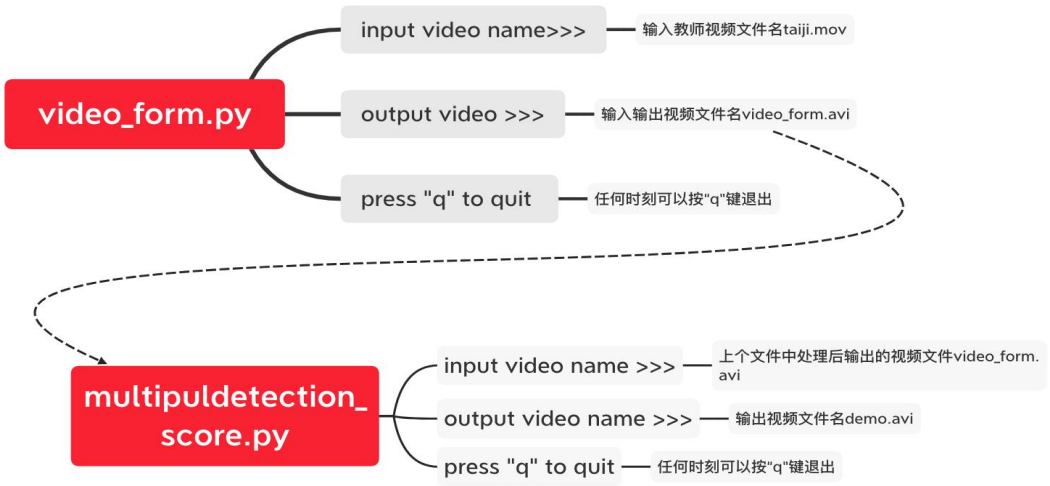
③纠正没有实时性，不能及时纠正学生错误。

因此我们希望找到一种比较便捷的方式来对自己的动作进行评判，从而能够帮助同学们提高锻炼效果。

1.2 项目介绍

项目基于多人识别的姿势评估。

项目分为两个部分，第一部分实现视频的获取，第二部分实现多人骨骼提取和对比评估。在第一部分中，将自己要练习动作的教师视频导入，并跟着视频完成相应动作，形成将二者结合的一个视频文件。然后第二部分文件将该视频中人物进行多人识别与骨骼提取，对识别的骨骼进行对比，最后对动作相似度进行评分。



(项目流程示意图)

1.3 Library 介绍

1.3.1 Openpose 网络架构

Openpose 是自下而上的人体姿势估计方法，即先确定关键点，再将关键点组合成为人的方法。该方法利用置信图获得关键点，利用关键点亲和图确定关键点之间的连接。

OpenPose 网络结构如下所示。

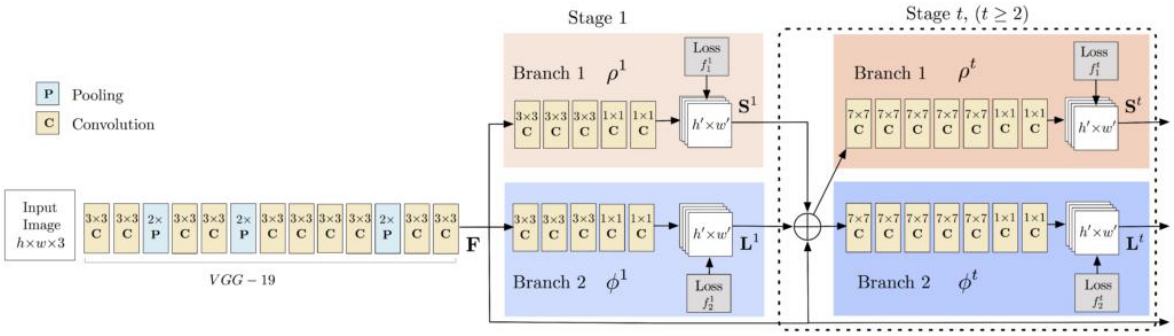


图 1：多人姿势估计模型架构

该模型以大小为 h*w 的彩色图像作为输入，并产生输出的矩阵数组，该矩阵由每个关键点对的关键点的置信度图(Confidence Maps)和零件相似性热图(Part Affinity Maps)组成。上述网络架构包括两个阶段，如下所述：

1.阶段 1：VGGNet 的前 10 层用于为输入图像创建特征图。

2.阶段 2：使用 2 分支多阶段 CNN，其中第一分支预测身体部位位置（例如，肘部，膝盖等）的一组 2D 置信图,称为集合 S。置信图 S 是一种灰度图像，在某些身体部位的可能性很高的位置上具有较高的值。例如，左图的置信度图如下图 2 所示。它在有左肩的所有位置都具有很高的价值。对于 18 点模型，输出的前 19 个矩阵对应于置信图。

图 2：显示给定图像的左肩置信度图（Confidence Maps）

第二个分支预测零件相似性（PAFs）的一组 2D 矢量场，称为 L，该向量场 L 对零件（关键点）之间的关联度进行编码。第 20 至第 57 个矩阵是 PAF 矩阵。在下图中 – 显示了脖子和左肩之间的部分亲和力。请注意，属于同一个人的各个部分之间存在很大的相似性。

图 3：显示给定图像的“颈部-左肩”的零件亲和度图（Part Affinity Maps）

1.3.2 COCO 数据集

COCO 数据库是一个大型图像数据集，根据用途不同，数据集分为目标检测，目标分割（对应的标注信息是“bbox”和“segmentation”），图像语义理解（“captions”），人体关节点（“keypoints”）。在本项目中，我们主要用到 coco 数据库的人体姿态检测使用的关

节点参数。

COCO 输出格式：鼻子－ 0，脖子－ 1，右肩－ 2，右肘－ 3，右腕－ 4，左肩－ 5，左肘－ 6，左腕－ 7，右臀部－ 8，右膝盖－ 9 ，右脚踝－ 10，左臀部－ 11，左膝－ 12，脚踝－ 13 ，右眼－ 14，左眼－ 15，右耳－ 16，左耳－ 17，背景－ 18

COCO KeyPoints

1.3.3 opencv2、numpy

本项目使用 opencv2 配合 numpy 进行图像处理。

2 代码解释

2.1 预处理

（1）关于 Video_form.py 文件的说明

首先导入所需的包和模块文件（事先已安装），有 opencv2，numpy，time

```
import cv2
import time
import numpy as np
```

然后定义函数 getvideoname()和 getoutputname（）用于获取输入视频文件名和输出视频文件名，并进行错误检测。如果输入正确，得到参数 oup 为输出文件名，cap 为 cv2.VideoCapture(vide)的输出。

```
def getvideoname():
    while True:
        try:
            vide = input("input video name >>>")
            cap = cv2.VideoCapture(vide)
```

```
            hasFrame, image1 = cap.read()
            image1 = cv2.resize(image1, (600, 350))
            return cap
        except:
            print("can't find the video,try again!")

def getoutputname():
    while True:
        oup = input("output video name >>>")
        if oup[-4:]=='.avi':
            return oup
        else:
            print("output video name should end with'.avi'")

cap=getvideoname()
oup=getoutputname()
```

打开摄像头：

```
capp=cv2.VideoCapture(0)
```

读取输入视频，对原视频进行缩放，设置 vid_writer 为后续视频写入作准备。
将视频写入的框架宽设置为两倍，因为之后要做一个视频拼接。

```
ret , frame = cap.read()
frame=cv2.resize(frame,(600,350))
vid_writer =cv2.VideoWriter(oup,cv2.VideoWriter_fourcc(*'MJPG'),
10, (frame.shape[1]*2, frame.shape[0]))
```

进行循环读取处理视频帧，输出水平拼接的原视频帧和相机拍摄到的视频帧：

```
while(True):
    ret , frame = cap.read()
    if not ret:
        break#如果不能正常显示，退出循环

    tim=round(time.time() - t,2)#显示拍摄时长，保留两位小数
    #在窗口上显示时间
    cv2.putText(frame, "time:{} sec".format(tim), (10, 50),
cv2.FONT_HERSHEY_COMPLEX, 1,
                (255, 50, 0), 1, lineType=cv2.LINE_AA)
    #进行缩放
    frame=cv2.resize(frame,(600,350))
    hasframe, frame2 = capp.read()
```

```
    frame2=cv2.resize(frame2,(frame.shape[1],frame.shape[0]),
interpolation = cv2.INTER_AREA)
    #水平拼接
    hmerge = np.hstack((frame, frame2))
    #视频显示
    cv2.imshow('teaching and imitate video',hmerge)
    vid_writer.write(hmerge)#视频写入预定的设置格式


    if cv2.waitKey(1) &0xFF ==ord('q'):  #按q键退出
        break
```

最后，释放窗口、视频：

```
# 释放视频，关闭窗口
cap.release()
vid_writer.release()
cv2.destroyAllWindows()
```

（2）关于 multipuldetection_score.py 的预处理说明

导入包，包括 opencv2,time,numpy,sympy

```
import time
import numpy as np
import sympy as sy
```

类似于 video_form.py 中的代码，获取输入输出视频文件名，设置视频输出格式，进行图像放缩，计时并显示时间等等基本上相同部分，略去。

预留 COCO 输出格式。keypointsMapping 是对应顺序关键点的名字的列表，POSE_PAIRS 是关键点之间配对关系的列表，mapIdx 是在 PAFs（Part Affinity Map）中对应配对的索引。colors 是输出颜色预留。

```
keypointsMapping = ['Nose', 'Neck', 'R-Sho', 'R-Elb', 'R-Wr',
'L-Sho', 'L-Elb', 'L-Wr', 'R-Hip', 'R-Knee', 'R-Ank',
                    'L-Hip', 'L-Knee', 'L-Ank', 'R-Eye', 'L-Eye',
'R-Ear', 'L-Ear']
#特征点配对：如R-Sh 和Nose[1, 2]，R-Sh 和R-Elb[2, 3]
POSE_PAIRS = [[1, 2], [1, 5], [2, 3], [3, 4], [5, 6], [6, 7],
              [1, 8], [8, 9], [9, 10], [1, 11], [11, 12], [12, 13],
              [1, 0], [0, 14], [14, 16], [0, 15], [15, 17],
              [2, 17], [5, 16]]
# index of pafs correspoding to the POSE_PAIRS
# e.g for POSE_PAIR(1,2), the PAFs are located at indices (31,32)
of output, Similarly, (1,5) -> (39,40) and so on.
```

```
mapIdx = [[31, 32], [39, 40], [33, 34], [35, 36], [41, 42], [43,
44],
          [19, 20], [21, 22], [23, 24], [25, 26], [27, 28], [29, 30],
          [47, 48], [49, 50], [53, 54], [51, 52], [55, 56],
          [37, 38], [45, 46]]

colors = [[0, 100, 255], [0, 100, 255], [0, 255, 255], [0, 100,
255], [0, 255, 255], [0, 100, 255],
          [0, 255, 0], [255, 200, 100], [255, 0, 255], [0, 255, 0],
[255, 200, 100], [255, 0, 255],
          [0, 0, 255], [255, 0, 0], [200, 200, 0], [255, 0, 0], [200,
200, 0], [0, 0, 0]]
```

2.2 获取关键点 getKeypoints()

首先，加载网络：

```
protoFile = "pose/coco/pose_deploy_linevec.prototxt"
weightsFile = "pose/coco/pose_iter_440000.caffemodel"
net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile)
```

调用 cv2.dnn.blobFromImag 创建输入 Blob，

```
inHeight = 368
inWidth = int((inHeight / frameHeight) * frameWidth)
# 图像预处理 cv2.dnn.blobFromImage
inpBlob = cv2.dnn.blobFromImage(image1, 1.0 / 255, (inWidth,
inHeight), (0, 0, 0), swapRB=False, crop=False)
```

通过网络前进，输出 output：

```
net.setInput(inpBlob)
output = net.forward()
```

获得包含在 output 中的置信图 probMap：

```
for part in range(nPoints):
    probMap = output[0, part, :, :]
    probMap = cv2.resize(probMap, (image1.shape[1],
image1.shape[0]))
```

与原图像 addweighted 后输出的置信图 probmap

获得关键值。对于每个关键点，我们对置信度图应用阈值（此处为 0.1）

```
def getKeypoints(probMap, threshold=0.1):
    # 获取关键点，probMap：置信图，threshold：阈值
    # 高斯平滑
    mapSmooth = cv2.GaussianBlur(probMap, (3, 3), 0, 0)

    mapMask = np.uint8(mapSmooth > threshold)
```

这给出了一个矩阵，在与关键点相对应的区域中包含斑点，如下所示。

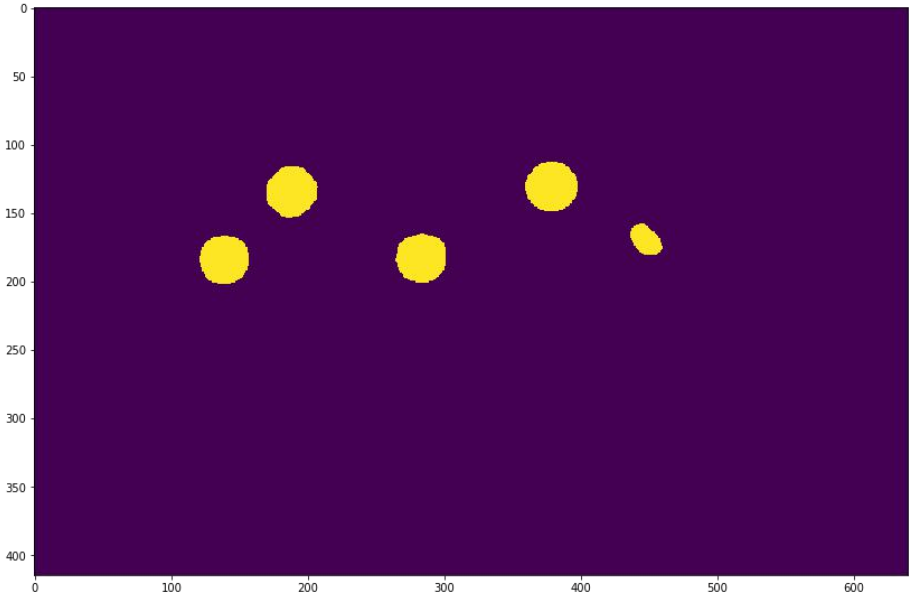


图 5：应用阈值后的置信度图

为了进一步找到关键点的确切位置，我们需要找到每个斑点的最大值。
首先找到与关键点对应的区域的所有轮廓。为此区域创建一个遮罩。通过将 probMap 与此掩码相乘，提取该区域的 probMap。找到该区域的局部最大值。这是针对每个轮廓（关键点区域）完成的。

```
    keypoints = []
    # find the blobs
    contours, _ = cv2.findContours(mapMask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    # for each blob find the maxima
    for cnt in contours:
        blobMask = np.zeros(mapMask.shape)
        blobMask = cv2.fillConvexPoly(blobMask, cnt, 1)
        maskedProbMap = mapSmooth * blobMask
        _, maxVal, _, maxLoc = cv2.minMaxLoc(maskedProbMap)
        keypoints.append(maxLoc + (probMap[maxLoc[1],
maxLoc[0]],))

    return keypoints
```

2.3 查找有效对 getValidPairs()

有效对是连接属于同一个人的两个关键点的身体部位。查找有效对的一种简单方法是找到一个关节与所有可能的其他关节之间的最小距离。例如，在下面给出的图中，我们可以找到标记的鼻子和所有其他脖子之间的距离。最小距离对应该是对应于同一个人的一对。

这种方法可能不适用于所有对。特别是当图像中包含过多的人或部分被遮挡时。例如，对于该对，左肘->左腕–与自己的手腕相比，第三人称的腕部更靠近第二人的肘部。因此，它将不会产生有效的配对。

这是“零件相似性贴图”起作用的地方。它们给出方向以及两个关节对之间的亲和力。因此，该对不仅应具有最小距离，而且其方向还应符合 PAF 热图的方向。

分割连接组成该对的两个点的线。在这条线上找到“ n”个点。检查这些点上的 PAF 是否与连接该对点的线的方向相同。如果方向在一定程度上匹配，则为有效对。

对于每个身体部位对，我们执行以下操作：

1.获取属于一对的关键点。将它们放在单独的列表中（candA 和 candB）。candA 中的每个点都将连接到 candB 中的某个点。下图显示了 Neck–> Right-Shoulder 对的 candA 和 candB 中的点。


```
    for k in range(len(mapIdx)):
        # A->B constitute a limb
        pafA = output[0, mapIdx[k][0], :, :]
        pafB = output[0, mapIdx[k][1], :, :]
        pafA = cv2.resize(pafA, (frameWidth, frameHeight))
        pafB = cv2.resize(pafB, (frameWidth, frameHeight))
# Find the keypoints for the first and second limb
        candA = detected_keypoints[POSE_PAIRS[k][0]]
        candB = detected_keypoints[POSE_PAIRS[k][1]]
```

2.找到连接两个点的单位向量。这给出了连接它们的线的方向。

```
d_ij = np.subtract(candB[j][:2], candA[i][:2])
norm = np.linalg.norm(d_ij)
if norm:
        d_ij = d_ij / norm
else:
        continue
```

3.在连接两个点的直线上创建一个包含 10 个插值点的数组。

```
# Find p(u)
interp_coord = list(zip(np.linspace(candA[i][0], candB[j][0],
num=n_interp_samples),
                    np.linspace(candA[i][1], candB[j][1],
num=n_interp_samples)))
# Find L(p(u))
paf_interp = []
for k in range(len(interp_coord)):
    paf_interp.append([pafA[int(round(interp_coord[k][1]))],
int(round(interp_coord[k][0]))],
                   pafB[int(round(interp_coord[k][1])),
int(round(interp_coord[k][0]))]])
```

4.取这些点上的 PAF 与单位矢量 d_ij 之间的点积

```
paf_scores = np.dot(paf_interp, d_ij)
avg_paf_score = sum(paf_scores) / len(paf_scores)
```

5.如果 70％的点满足标准，则将对视为有效。

```
if (len(np.where(paf_scores > paf_score_th)[0]) /
n_interp_samples) > conf_th:
    if avg_paf_score > maxScore:
        max_j = j
        maxScore = avg_paf_score
        found = 1
```

2.4 组装以人为本的关键点 getPersonwiseKeypoints（）

现在，我们已将所有关键点组合为对，我们可以将共享同一部分候选检测点的对组合为多个人的全身姿势。

我们首先创建一个空列表来存储每个人的关键点。然后我们遍历每对，检查其中的 partA 是否已存在于任何列表中。如果存在，则表示关键点属于此列表，并且该对中的 partB 也应属于此人。因此，将此对的 partB 添加到找到 partA 的列表中。

```
for i in range(len(valid_pairs[k])):
    found = 0
    person_idx = -1
    for j in range(len(personwiseKeypoints)):
        if personwiseKeypoints[j][indexA] == partAs[i]:
            person_idx = j
            found = 1
            break

    if found:
        personwiseKeypoints[person_idx][indexB] = partBs[i]
        personwiseKeypoints[person_idx][-1] +=
keypoints_list[partBs[i].astype(int), 2] + valid_pairs[k][i][2]
```

如果 partA 没有出现在任何列表中，则意味着该对属于不在列表中的新人，因此将创建一个新列表。

```
elif not found and k < 17:
    row = -1 * np.ones(19)
    row[indexA] = partAs[i]
    row[indexB] = partBs[i]
```

2.5 评分

在需要用到的 13 个点中，遍历每个点。通过判断-1 是否在 index 中，得到真实人的个数 personnum。

```
for i in range(13):
    anglelist = []
    wrongmap = []
    personnum=0
    for n in range(len(personwiseKeypoints)):
        index = personwiseKeypoints[n][np.array(POSE_PAIRS[i])]
        if -1 in index:
            continue
        personnum+=1
```

得到配对点的坐标，计算与地面的夹角，加入到创建的 list 中

```
        B = np.int32(keypoints_list[index.astype(int), 0])
        A = np.int32(keypoints_list[index.astype(int), 1])

wrongmap.append([(B[0], A[0]), (B[1], A[1])])
    # 获得与地面夹角
    a = np.array([B[1] - B[0], A[1] - A[0]])
    b = np.array([1, 0])
    cosangle = a.dot(b) / (np.linalg.norm(a) * np.linalg.norm(b))
    angle = sy.acos(cosangle)
    anglelist.append(angle)
```

通过角度方差进行判断是否合格。因为无法确定哪个人是标准的，所以采用方差，即数值偏离的量来做判断。当有 2 个人时，以 15 度偏角的方差量为标准即((15/ 180) ** 2)*2，实际两个人的偏差夹角可以达到 30 度。而当 n 比较大时，以((15/ 180) ** 2)*n 为判断标准，相当于对多人动作进行一致性估计。

将出现错误的骨骼显示为白色。

```
if len(anglelist)==personnum and personnum>1:
    arr_var = np.var(anglelist)
    correctdectect += 1
    # 计算方差
    if arr_var > ((15 / 180) ** 2)*n:  #  #以20 度为误差划分
        wrongs += 1
        for i in range(len(wrongmap)):
            cv2.line(frameClone, wrongmap[i][0],wrongmap[i][1],
[255, 255, 255], 2, cv2.LINE_AA)
```

3.局限与改进

局限：

1．代码运行速度比较慢。项目运行因视频长度，视频中人物个数而异。时间越长，人数越多，时间越长。我们选用的素材是时常为 30 秒的视频，后面的比较过程中需要花将近 30 分钟完成。具体原因可能是因为我们选择了 18 个关键点然后对其中的 13 个数据进行比较。如果改进后效率可提高 30%左右。

2．在进行试用的时候，有时候会因为需要将整个人体都放入电脑屏幕中，导致使用者可能会看不清电脑显示的标准视频。对此我们可以将将视频的窗口调试得尽量大一些。

3. 我们的代码进行识别时是采用的多人识别，当人与人之间靠得太近或者说相互发生遮挡时，关键点的捕捉和选取可能会受到干扰。

4. 我们的代码要求使用者和老师的节奏基本相同，在同学在做的时候可能会达不到这个标准从而影响整个视频的比较过程

5.识别过程中存在关键点跳跃导致骨骼抖动的情况,如何结合前后帧信息消除抖动。

4.总结

在现在上网课的大背景下，体育课线上教授可能不是特别方便。我们的初衷是希望能够在这个方面有所帮助。而用代码解决这个问题的过程中，我们通过寻找相关的数据库(主要是 OpenCV)，对其进行一定的使用，以及自己编写代码的这个过程，加深了自己对代码的熟练度，提升了自己对编程的兴趣。我们在今后的学习中，也会进一步体会 python 的魅力。