



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

Openpose 实验实习报告

课程名称： 生产实习

任课老师： 吴子朝老师

姓名： 赵茜茜

学号： 18220211

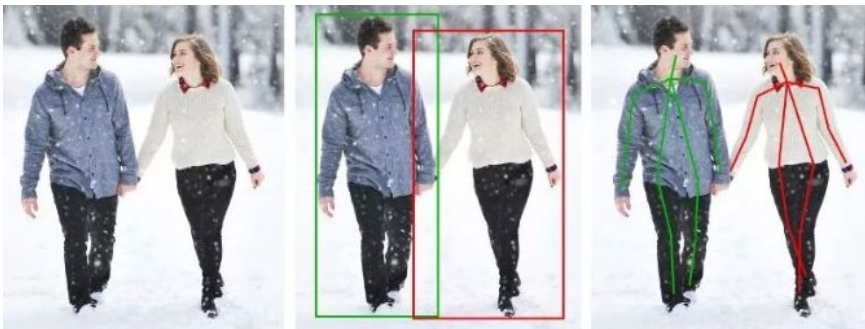
专业： 数字媒体技术

Openpose 姿态检测

1. 背景

在当前姿态识别的领域，估计图像中的多人姿势有许多挑战需要攻克，例如，在图像识别中，不同场景下未知的人数、任意的姿势或比例，人们的接触与重叠等等都将是至关重要需要解决的问题。在当下的团队中，一种较为常见的方法是使用人员检测器并为每个检测执行单人姿势估计，自上而下的方法听起来非常直观和简单。但是，这种方法存在一些隐藏的缺陷，例如，当人员检测器失败时没有资源可以恢复，运行时间与人数成正比，并且，

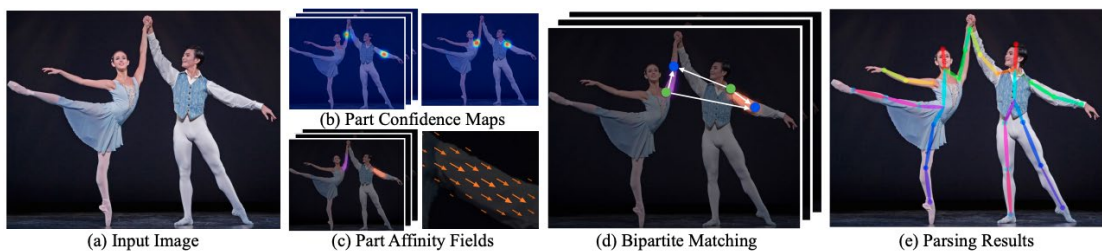
即使人员检测器失败，也会执行姿势估计。



图：自上而下的方法

2. 概述

当下也有其他一些团队尝试了自下而上的方法，但仍然面临一些问题，例如，在最终的解析时需要昂贵的全局推理，并且没有保证效率上的收益，未利用全局上下文先验信息，即图片中其他人的身体关键点信息，并且将关键点对应到不同的人物个体，算法复杂度太高。而本论文提出了一种有效检测图像中多人姿势的方法。我们将其称为部件亲和域，用以学习将身体部位与图像中的个体相关联。该架构对全局上下文进行编码，基于检测出的关节点和关节联通区域，用贪心算法进行自下而上地解析，可以将这些关节点快速对应到不同人物个体，在实现实时性能的同时保持高精度。该架构旨在通过同一顺序预测过程的两个分支共同学习零件位置及其关联。



特点（创新点）

PAF，这是在关键点之间建立的一个向量场，描述一个 limb 的方向。有了 heatmap 和

PAF，再使用二分图最大权匹配算法来对关键点进行组装，从而得到人体骨架。

优点

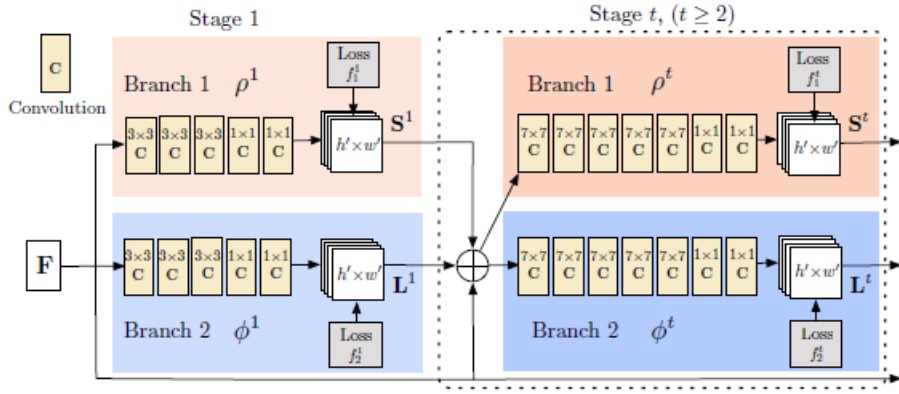
计算自下而上的检测和关联编码，并能够为后续的解析过程提供足够的全局上下文，同时做到小部分计算成本和高质量结果

3. 方法

架构

采用了两分支多级 CNN 的架构，顶部米色显示的分支用于预测置信图 confidence maps (S)，另外一个分支，即底部蓝色显示的分支用于预测 Par Affinity Fields (L)，也对应着 heatmap 与 vectormap。

其中 $S = (S_1, S_2, \dots, S_j)$ ，表示 heatmap，j 表示要检测的关节数； $L = (L_1, L_2, \dots, L_C)$ ，表示 vectormap，C 表示要检测的关节对数。



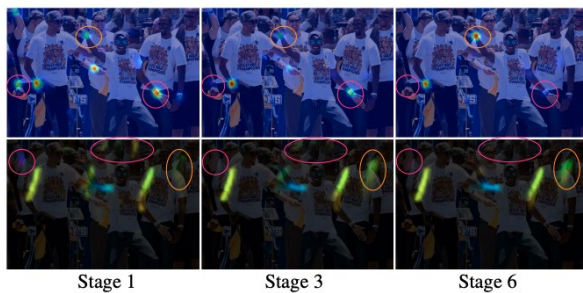
网络的 Stage1 接收的输入是特征 F，然后经过 Branch1 和 Branch2 网络的处理后分别得到 S_1 和 L_1 。从 Stage2 开始，阶段 t 网络的输入包括三部分： S_{t-1}, L_{t-1}, F 。每个阶段网络的输入为：

$$S^t = \rho^t(F, S^{t-1}, L^{t-1}), \forall t \geq 2$$

$$L^t = \phi^t(F, S^{t-1}, L^{t-1}), \forall t \geq 2$$

其中 ρ^t 和 ϕ^t 是 t 阶段用于推理的 CNN。

下图显示了多阶段网络的积极优势，我们观察到在前几个阶段左右身体部位之间存在一些混淆。但随着阶段的进展，网络变得更擅长做出这些区分。



损失函数

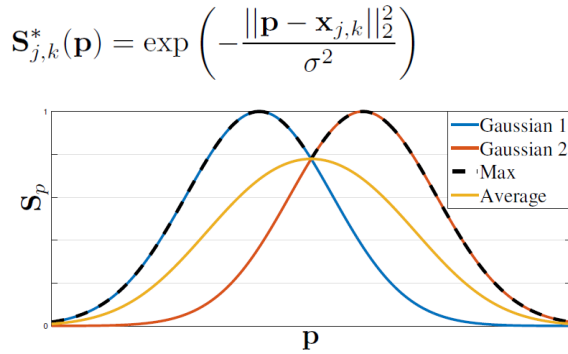
项目使用了两个损失函数，分别对应于每个 stage 的两个 branch:

$$f_S^t = \sum_{j=1}^J \sum_{\mathbf{p}} \mathbf{W}(\mathbf{p}) \cdot \|\mathbf{S}_j^t(\mathbf{p}) - \mathbf{S}_j^*(\mathbf{p})\|_2^2$$
$$f_L^t = \sum_{c=1}^C \sum_{\mathbf{p}} \mathbf{W}(\mathbf{p}) \cdot \|\mathbf{L}_c^t(\mathbf{p}) - \mathbf{L}_c^*(\mathbf{p})\|_2^2$$

\mathbf{S}_j^* 是真实部分置信度图，而是 \mathbf{L}_c^* 真实身体关节亲和向量场。 \mathbf{W} 是一个二进制掩码，当位置 \mathbf{p} 的标签缺失的时候， $\mathbf{W}(\mathbf{p})=0$ ，否则就为 1，掩码用于避免在训练期间惩罚真正的正预测，显然，对于未被标记的人物关节点 $\mathbf{W}(\mathbf{p})=0$ ，而被标记的人物关节点和非关节点 $\mathbf{W}(\mathbf{p})=1$ ，所以未被标记的人物关节点不会影响模型的学习过程，整个 CNN 网络架构的总体目标函数为：

$$f = \sum_{t=1} T(f_S^t + f_L^t)$$

为了评估上述方程式，论文根据标注了 2D 的关键点的图生成真实置信图。每一个置信图表示身体的一个特定的部位在图像上某点发生的可能性，如果图像只有一个人每个置信图理论上只有一个峰值当图像中有多人时，对应每一个人 k 都对应一个可见的身体部位 j 的峰值：



网络在位置 \mathbf{P} 的预测值对应的置信值计算是如上图所示是取最大值：

$$\mathbf{S}_j^*(\mathbf{p}) = \max_k \mathbf{S}_{j,k}^*(\mathbf{p})$$

PAFs

Part Affinity Fields (PAFs)，一组二维矢量场，用于对图像域上肢体的位置和方向进行编码。



论文以右手的小臂部分的肢体图像为例子。使用 $x_{j_1, k}$ 和 $x_{j_2, k}$ 来表示来自某个人 k 肢体 c 的部位 j_1 和部位 j_2 的真实位置。如果一个点 p 在这个肢体 c 上，那么 $L_{c,k}^*(p)$ 的值就是一个从 j_1 指向 j_2 的单位向量，而对于所有其他的点，这个向量都是 0。

在论文中，定义了真实部分亲和向量场：

$$L_{c,k}^*(p) = \begin{cases} v & \text{if } p \text{ on limb } c, k \\ 0 & \text{otherwise.} \end{cases}$$

其中 v 是肢体方向的单位向量，其中点 p ：

$$0 \leq v \cdot (p - x_{j_1, k}) \leq l_{c,k} \text{ and } |v_{\perp} \cdot (p - x_{j_1, k})| \leq \sigma_l$$

其中 σ_l 是肢体宽， $l_{c,k} = \|x_{j_2, k} - x_{j_1, k}\|_2$ 是肢体长度， v_{\perp} 是垂直于 v 的一个向量。

真实部分亲和场是将上面提出的对于每个人 k 计算的值再进行平均化：

$$L_c^*(p) = \frac{1}{n_c(p)} \sum_k L_{c,k}^*(p)$$

其中 $n_c(p)$ 是在某个点 p 上对于所有 k 个人的非零向量个数。

测试期间，论文通过沿着连接候选部分位置的线段，计算相应 PAF 上的线性积分来测量候选部件检测之间的关联：

$$E = \int_{u=0}^{u=1} L_c(p(u)) \cdot \frac{d_{j_2} - d_{j_1}}{\|d_{j_2} - d_{j_1}\|_2} du$$

其中 $p(u)$ 是通过均匀采样 u 的方式插值两个候选身体部位 d_{j_1} 和 d_{j_2} 的位置，用这来求这两个关节点的相似度：

$$p(u) = (1 - u)d_{j_1} + ud_{j_2}$$

使用 PAFs 进行多人解析

论文对检测置信图执行非极大抑制，来获得一组离散的身体部位候选点。由于图中有多人，对于每种关键点，有若干候选边，因此有很多可能的肢体，易产生误报。

为解决这个问题，论文通过计算 PAF 上的线积分评估每一个候选肢体。找到最优肢体是一个 NP 问题，在本文中使用一个贪心松弛策略（匈牙利算法），使得每次都能产生高质量的匹配。

算法目标：找到所有最佳关联，记为 Z 。考虑一对关键点 j_1, j_2 (例如脖子，右臀) 组成肢体 c ，找到最佳匹配：

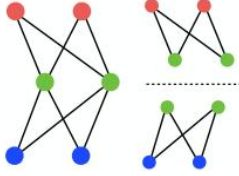
$$\begin{aligned}
\max_{\mathcal{Z}_c} E_c &= \max_{\mathcal{Z}_c} \sum_{m \in \mathcal{D}_{j_1}} \sum_{n \in \mathcal{D}_{j_2}} E_{mn} \cdot z_{j_1 j_2}^{mn} \\
\text{s.t.} \quad &\forall m \in \mathcal{D}_{j_1}, \sum_{n \in \mathcal{D}_{j_2}} z_{j_1 j_2}^{mn} \leq 1 \\
&\forall n \in \mathcal{D}_{j_2}, \sum_{m \in \mathcal{D}_{j_1}} z_{j_1 j_2}^{mn} \leq 1
\end{aligned}$$

其中 E_c 是来自肢体类型 c 的匹配的总权重， \mathcal{Z}_c 是肢体类型 c 的 \mathcal{Z} 的子集， E_{mn} 是等式中定义的部分 $\mathbf{d}_{j_1}^m$ 和 $\mathbf{d}_{j_2}^n$ 之间的部分的亲和力。

论文再采用两个松弛进行优化：

首先，选用最少的边来或得一个人的骨架，而不是使用完整的图。

然后，将匹配问题分解成一组二部匹配的子问题，独立地确定点的匹配。



通过这两个松弛，优化就可以简单地分解为：

$$\max_{\mathcal{Z}} E = \sum_{c=1}^C \max_{\mathcal{Z}_c} E_c$$

4. 结果

论文在两个基准上评估：1. MPII human multi-person dataset 2. COCO 2016 keypoints challenge dataset . 并包含了不同场景、拥挤、遮挡、接触、比例变化。

MPII Multi-Person

参考论文 (Deepcut: Joint subset partition and labeling for multi person pose estimation. In CVPR, 2016. 1) 的工具包来测量基于 PCKh 阈值的所有身体部位的 mAP。准确率高，运行极快。在全部测试集上，使用 3 种尺度 (x0.7, x1, x1.3)，进一步将性能提高到 78.6%。

Method	Hea	Sho	Elb	Wri	Hip	Kne	Ank	mAP	s/image
Subset of 288 images as in [22]									
Deepcut [22]	73.4	71.8	57.9	39.9	56.7	44.0	32.0	54.1	57995
Iqbal et al. [12]	70.0	65.2	56.4	46.1	52.7	47.9	44.5	54.7	10
DeeperCut [11]	87.9	84.0	71.9	63.9	68.8	63.8	58.1	71.2	230
Ours	93.7	91.4	81.4	72.5	77.7	73.0	68.1	79.7	0.005
Full testing set									
DeeperCut [11]	78.4	72.5	60.2	51.0	57.2	52.0	45.4	59.5	485
Iqbal et al. [12]	58.4	53.9	44.5	35.0	42.2	36.7	31.1	43.1	10
Ours (one scale)	89.0	84.9	74.9	64.2	71.0	65.6	58.1	72.5	0.005
Ours	91.2	87.6	77.7	66.8	75.4	68.9	61.7	75.6	0.005

下表展示了验证集上不同结构的比较，说明贪心策略可行，效率提高很多，且效果更好（收敛更快）。

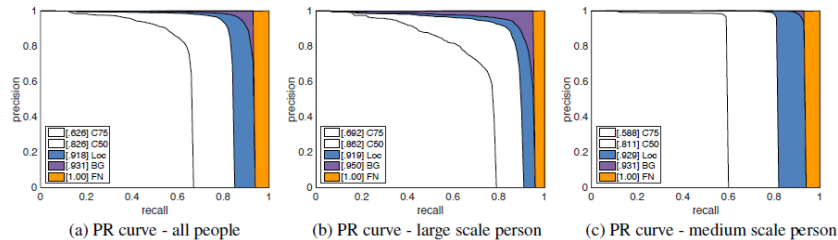
Method	Hea	Sho	Elb	Wri	Hip	Kne	Ank	mAP	s/image
Fig. 6b	91.8	90.8	80.6	69.5	78.9	71.4	63.8	78.3	362
Fig. 6c	92.2	90.8	80.2	69.2	78.5	70.7	62.6	77.6	43
Fig. 6d	92.0	90.7	80.0	69.4	78.4	70.1	62.3	77.4	0.005
Fig. 6d (sep)	92.4	90.4	80.9	70.8	79.5	73.1	66.5	79.1	0.005

COCO key points challenge

论文的方法在小规模人群(APM)上的准确性低于自顶向下的方法。原因是我们的方法必须在一次拍摄中处理图像中所有人所跨越的更大的尺度范围。相比之下，自上而下的方法可以将每个检测到的区域的补丁重新缩放到更大的尺寸，从而在更小的尺度上遭受更少的退化。

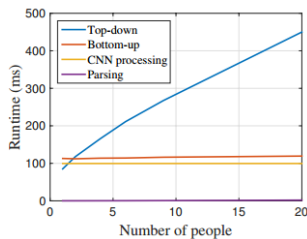
Team	AP	AP ⁵⁰	AP ⁷⁵	AP ^M	AP ^L
Test-challenge					
Ours	60.5	83.4	66.4	55.1	68.1
G-RMI [19]	59.8	81.0	65.1	56.7	66.7
DL-61	53.3	75.1	48.5	55.5	54.8
R4D	49.7	74.3	54.5	45.6	55.6
Test-dev					
Ours	61.8	84.9	67.5	57.1	68.2
G-RMI [19]	60.5	82.2	66.2	57.6	66.6
DL-61	54.4	75.3	50.9	58.3	54.3
R4D	51.4	75.0	55.9	47.4	56.7

下图也显示了论文的方法在验证集上的错误细分。大多数误报来自不精确的定位，而不是背景混淆。这表明在捕捉空间依赖性方面比在识别身体部位外观方面有更多的改进空间。



运行时分析

通过测试比较自顶向下的方法：人体检测+CPM，与本论文中的方法，本方法更有优势，运行时间由两个主要部分组成：(1) 处理时间，其运行时间复杂度为 $O(1)$ ，随着人数的变化而保持不变；(2) 多人解析时间，运行时复杂度为 $O(n^2)$ ，其中 n 代表人数。



工程源码

项目目录与源码文件

```
| appveyor.yml      #规范文件目录
| CMakeLists.txt    #Cmake 列表文件
| LICENSE           #证书
| README.md         #github 主页说明文本
├─3rdparty          #第三方
|   | Versions.txt
|   └─caffe         #深度学习框架——快速特征嵌入的卷积结构
|   └─pybind11      #绑定 c++ 与 python 的轻量级库
|   └─windows       #windows 支持
|       | getCaffe.bat
|       | getCaffe3rdparty.bat
|       | getFreeglut.bat
|       | getOpenCV.bat
|       | getSpinnaker.bat
|       └─caffe
|       └─caffe3rdparty
|       └─freeglut
|       └─opencv    #计算机视觉和机器学习软件库
|       └─unzip
|       └─wget      # http 下载工具
├─build             #Cmake 生成文件
|   | ALL_BUILD.vcxproj
|   | ALL_BUILD.vcxproj.filters
|   | cmake_install.cmake
|   | OpenPose.sln  #主项目
|   └─bin           #编译文件存放
|   └─CMakeFiles
|   └─examples      # Openpose 官方给予的相关实例
|       | cmake_install.cmake
|       └─calibration #Openpose 的校准工具 用于计算并保存输入图像的内在参数
|       └─CMakeFiles
|       └─deprecated
|       └─openpose   #demo——展示了 openpose 库的所有功能。
|           | cmake_install.cmake
|           | OpenPoseDemo.vcxproj
|           | OpenPoseDemo.vcxproj.filters
|           | OpenPoseDemo.vcxproj.user
|           └─CMakeFiles
|               | generate.stamp
|               | generate.stamp.depend
|               └─Debug
```



```

| | | |——OpenPoseDemo.dir    #运行文件  debug&release
| | | | |——Debug
| | | | |——Release
| | | |——Release
| | |——tutorial_api_cpp    #实例教程  C++ API
| | | | 01_body_from_image_default.vcxproj    #从图像中处理并识别姿势
| | | | 01_body_from_image_default.vcxproj.filters
| | | | 01_body_from_image_default.vcxproj.user
| | | | 02_whole_body_from_image_default.vcxproj    #从图像中处理并识别姿势（全身）
| | | | 02_whole_body_from_image_default.vcxproj.filters
| | | | 02_whole_body_from_image_default.vcxproj.user
| | | | 03_keypoints_from_image.vcxproj    #从图像中处理并显示关键点
| | | | 03_keypoints_from_image.vcxproj.filters
| | | | 03_keypoints_from_image.vcxproj.user
| | | | 04_keypoints_from_images.vcxproj
| | | | 04_keypoints_from_images.vcxproj.filters
| | | | 04_keypoints_from_images.vcxproj.user
| | | | 05_keypoints_from_images_multi_gpu.vcxproj    #从图像中处理并显示关键点（多 gpu）
| | | | 05_keypoints_from_images_multi_gpu.vcxproj.filters
| | | | 05_keypoints_from_images_multi_gpu.vcxproj.user
| | | | 06_face_from_image.vcxproj    # 面部识别 image
| | | | 06_face_from_image.vcxproj.filters
| | | | 06_face_from_image.vcxproj.user
| | | | 07_hand_from_image.vcxproj    #手势识别 image，输出手部关键点
| | | | 07_hand_from_image.vcxproj.filters
| | | | 07_hand_from_image.vcxproj.user
| | | | 08_heatmaps_from_image.vcxproj    #图像热度图处理显示
| | | | 08_heatmaps_from_image.vcxproj.filters
| | | | 08_heatmaps_from_image.vcxproj.user
| | | | 09_keypoints_from_heatmaps.vcxproj    #从图像热度图处理运行算法显示
关键点
| | | | 09_keypoints_from_heatmaps.vcxproj.filters
| | | | 09_keypoints_from_heatmaps.vcxproj.user
| | | | 10_asynchronous_custom_input.vcxproj    #异步自定义输入处理与识别
| | | | 10_asynchronous_custom_input.vcxproj.filters
| | | | 10_asynchronous_custom_input.vcxproj.user
| | | | 11_asynchronous_custom_input_multi_camera.vcxproj    #异步多摄像头输入
| | | | 11_asynchronous_custom_input_multi_camera.vcxproj.filters
| | | | 11_asynchronous_custom_input_multi_camera.vcxproj.user
| | | | 12_asynchronous_custom_output.vcxproj    # 异步自定义输出
| | | | 12_asynchronous_custom_output.vcxproj.filters
| | | | 12_asynchronous_custom_output.vcxproj.user
| | | | 13_asynchronous_custom_input_output_and_datum.vcxproj    #异步自定义输入输出数据

```

```

|   |   |   |   13_asynchronous_custom_input_output_and_datum.vcxproj.filters
|   |   |   |   13_asynchronous_custom_input_output_and_datum.vcxproj.user
|   |   |   |   14_synchronous_custom_input.vcxproj    #同步自定义输入
|   |   |   |   14_synchronous_custom_input.vcxproj.filters
|   |   |   |   14_synchronous_custom_input.vcxproj.user
|   |   |   |   15_synchronous_custom_preprocessing.vcxproj  #同步自定义预处理
|   |   |   |   15_synchronous_custom_preprocessing.vcxproj.filters
|   |   |   |   15_synchronous_custom_preprocessing.vcxproj.user
|   |   |   |   16_synchronous_custom_postprocessing.vcxproj #同步自定义后期处理
|   |   |   |   16_synchronous_custom_postprocessing.vcxproj.filters
|   |   |   |   16_synchronous_custom_postprocessing.vcxproj.user
|   |   |   |   17_synchronous_custom_output.vcxproj          #同步自定义输出
|   |   |   |   17_synchronous_custom_output.vcxproj.filters
|   |   |   |   17_synchronous_custom_output.vcxproj.user
|   |   |   |   18_synchronous_custom_all_and_datum.vcxproj    #同步自定义所有模式集成
|   |   |   |   18_synchronous_custom_all_and_datum.vcxproj.filters
|   |   |   |   18_synchronous_custom_all_and_datum.vcxproj.user
|   |   |   |   cmake_install.cmake
|   |   |   |   └─CMakeFiles
|   |   |   |   └─Debug
|   |   |   |   └─Release    #运行生成文件
|   |   |   └─tutorial_api_python
|   |   |   └─user_code
|   └─src                #源码
|   |   |   cmake_install.cmake
|   |   |   └─CMakeFiles
|   |   └─openpose        #openpose 库源码
|   └─x64
└─cmake
└─doc                # 说明 mrakdown 文件
└─examples
|   |   CMakeLists.txt
|   └─calibration
|   └─deprecated
|   └─media    #demo 实例中输入的 image & video
|   └─openpose #包含所有 openpose 库功能的 demo 源文件
|   └─tests
|   └─tutorial_api_cpp    #openpose 官方示例的 demo 源文件 (C++)
|   └─tutorial_api_python #openpose 官方示例的 demo 源文件 (python)
|   └─user_code          #供用户生成原型
└─include    #openpose 库头文件
|   └─openpose
|   └─openpose_private
└─models    #openpose 模型文件

```

```
| | getModels.bat #windows 获取模型文件
| | getModels.sh
| |─cameraParameters
| |─face
| |─hand
| |─pose
└─python
  | | CMakeLists.txt
  |─openpose
└─scripts #相关脚本
  |─Cl
  |─osx
  |─tests
  |─ubuntu
  |─ubuntu_deprecated
└─src #openpose 库源文件
  | CMakeLists.txt
  └─openpose
    └─3d
      └─calibration
      └─core
      └─face
      └─filestream
      └─gpu
      └─gui
      └─hand
      └─net
      └─pose
      └─producer
      └─thread
      └─tracking
      └─unity
      └─utilities
      └─wrapper
```

实验过程

1. 实验平台与工具

Window 10 Version 21H1 64bit;

Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 12 核心

NVIDIA GeForce GTX 1050 4G

Microsoft Visual Studio 2019 Community ; Cmake

2. 实验步骤

1、 安装Microsoft Visual Studio 2019 Community

<https://visualstudio.microsoft.com/zh-hans/downloads/>

2、 VS IDE安装必须模块



3、 确认CUDA核心与显卡支持的版本号

项目

NVIDIA GeForce GTX 1050

细节

驱动程序版本：471.11

驱动器类型：DCH

Direct3D 功能... 12_1

CUDA 核心：640

图形时钟：1354 MHz

内存数据速率：7.01 Gbps

显示


组件

文件名

文件版本


产品名称

3D 设置

nvGameS.dll


30.0.14.7111

NVIDIA 3D Settings Server

nvGameSR.dll


30.0.14.7111

NVIDIA 3D Settings Server

NVCUDA64.DLL

30.0.14.7111

NVIDIA CUDA 11.4.56 driver

PhysX

09.19.0218

NVIDIA PhysX

4、 英伟达官网下载对应版本Cuda并安装

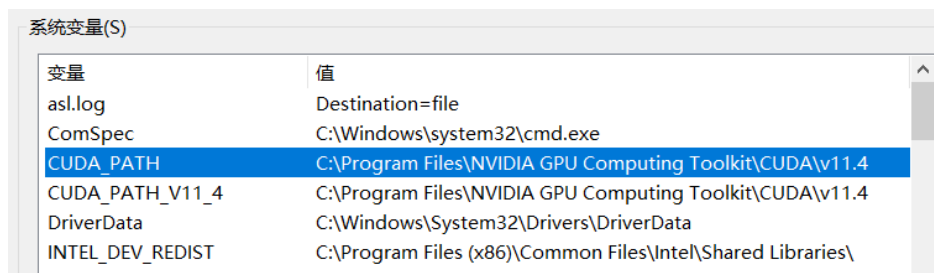
https://developer.download.nvidia.com/compute/cuda/11.4.0/local_installers/cuda_11.4.0_471.11_win10.exe

5、 安装对应Cuda版本的CuDnn

<https://developer.nvidia.com/rdp/cudnn-download>

解压压缩包，将 bin、include、lib 目录复制到 Cuda 的 bin、include、lib 目录

6、添加环境变量



7、验证 cuda & cudnn 是否正确安装, cmd 输入 `nvcc -V` 用于验证 cuda 是否正确安装; 打开路径 `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.4\extras`, 运行 NVIDIA 提供的 `deviceQuery.exe` 和 `bandwidthTest.exe` 来查看 GPU 的状态

8、从 github 中克隆 openpose 源码至本地仓库

<https://github.com/CMU-Perceptual-Computing-Lab/openpose>

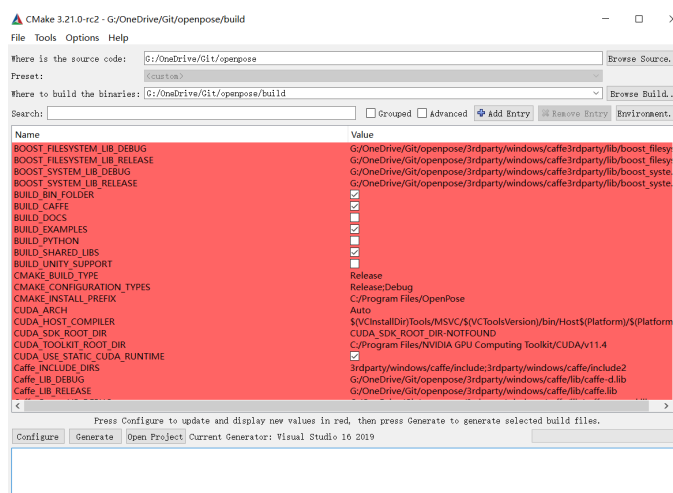
本人使用 GitHub desktop 克隆至本地

9、打开 .. openpose\3rdparty 目录, 执行以下批处理程序, 确保第三方库支持。

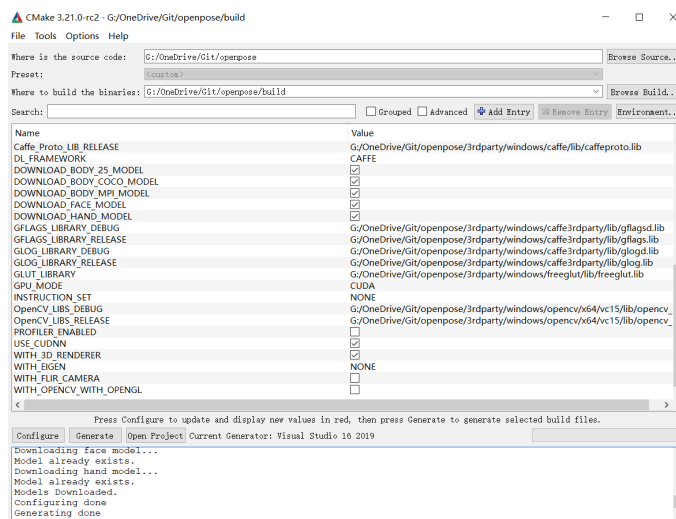
getCaffe.bat	✓	2021/7/5 15:43
getCaffe3rdparty.bat	✓	2021/7/5 15:43
getFreeglut.bat	✓	2021/7/5 15:43
getOpenCV.bat	✓	2021/7/5 15:43
getSpinnaker.bat	✓	2021/7/5 15:43

10、打开 .. openpose\models 目录, 执行 getModels.bat 程序, 获取模型文件。

11、打开 cmake, 编译。如下, 上面为本地 openpose 源码目录, 下面为 build 文件夹作为生成文件的存放地点, 选择 configure 确认 visual studio 版本, 设置完成后 generate。



12、 编译后，显示如下，则成功



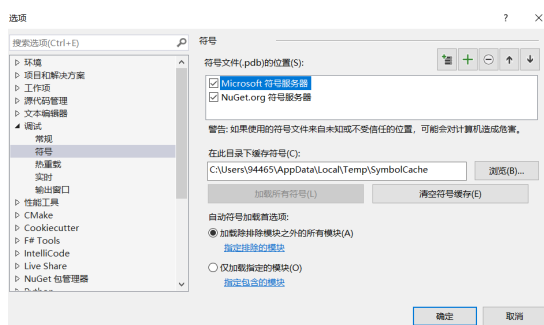
13、 选择 open project，进入 visual studio 2019，选择 ALL_BUILD 右键-生成

14、 把 openposeDemo 设置为启动项，并 Release 下运行，生成 OpenPose library 库。

问题一：运行报错，显示未加载



解决：点击“调试”-“选项”，勾选“常规”下的“启用源服务器支持”，勾选“符号”下的“Microsoft 符号服务器”，确定后重新运行。



问题二：运行时报错

F0707 10:49:43.372203 17272 syncedmem.cpp:71] Check failed: error == cudaSuccess (2 vs. 0) out of memory

*** Check failure stack trace: ***,

解决：报错原因在于使用了默认 BODY_25 模型,该模型要求的显存至少为 2.5G（安装 CuDnn 之后），因此转换模式为 COCO --model_pose COCO，或者减小图像输入尺寸或减少追踪人数,使用 --net_resolution 320x176 减小输入图像尺寸或--number_people_max 1 减少追踪人数，这些命令参数可以在 visual studio 项目属性下设置编辑。此外，在源文件中添加代码，有相同功能，类似如下：

```
int main(int argc, char *argv[])
{
    // Parsing command line flags

    FLAGS_number_people_max = 1;

    FLAGS_model_pose = "COCO";

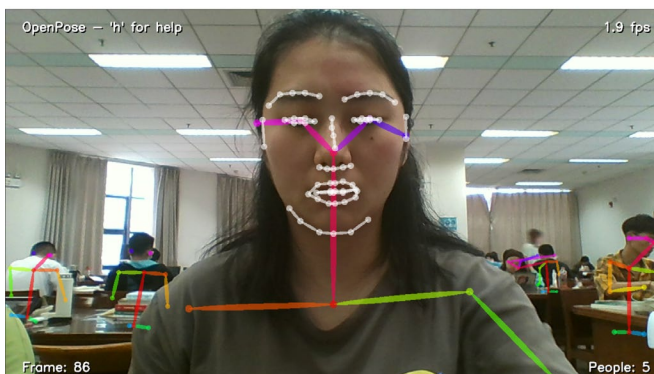
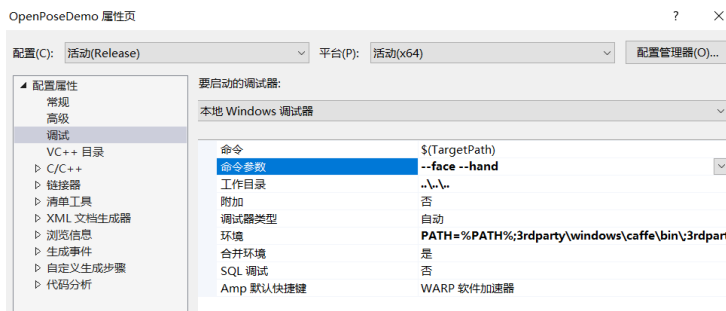
    FLAGS_net_resolution = "320x176";

    gflags::ParseCommandLineFlags(&argc, &argv, true);

    // Running tutorialApiCpp

    return tutorialApiCpp();
}
```

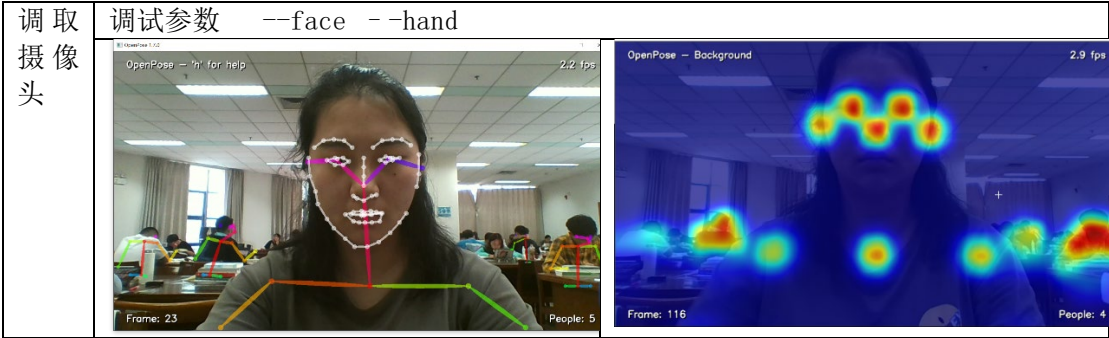
15、 设置在 Example/OpenPoseDemo 中设置启动参数 --face --hand，运行，将默认调用摄像头。

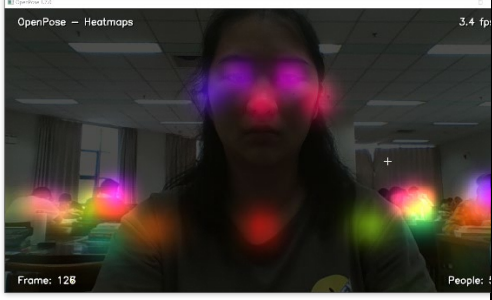
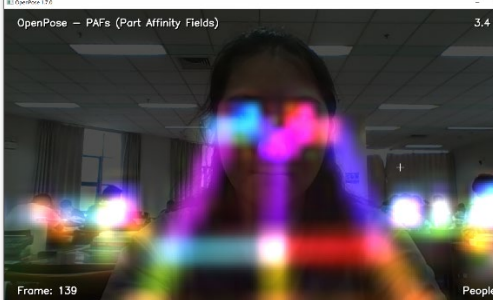
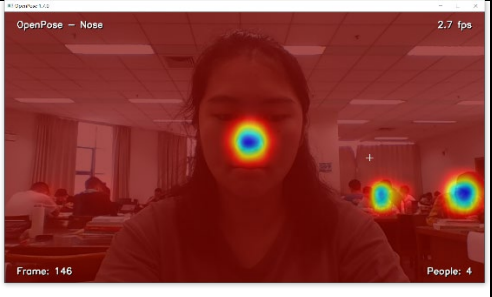
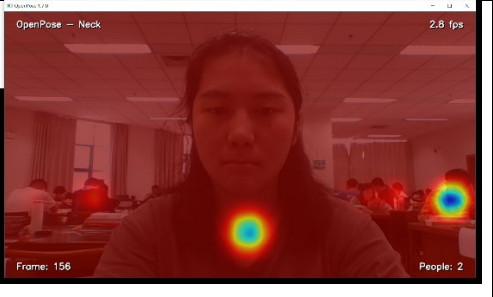
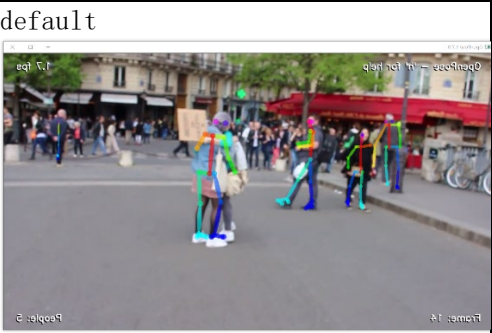
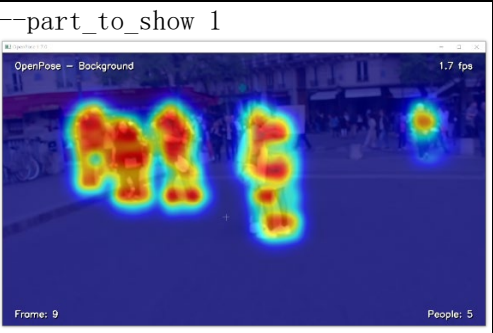
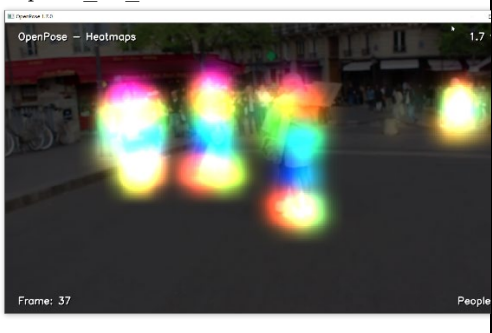





16、依次设置 C++API 目录下的各种工程为启动项目，实验调试启动参数来测试 open pose 的功能与实现。

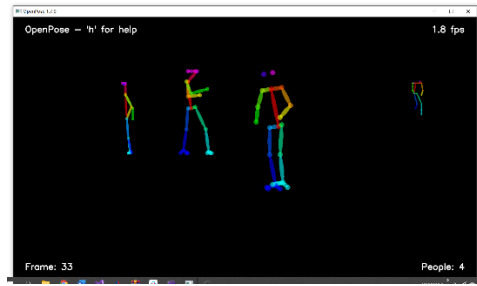
- face: 开启 Face 关键点检测.
 - hand: 开启 Hand 关键点检测
 - video input.mp4: 读取 Video.
 - image_dir path_to_images/: 运行图像路径内的图片.
 - ip_camera 在 streamed IP camera 上运行.
 - write_video path.avi: 将处理后的图片保存为 Video.
 - write_images folder_path: 将处理后的图片保存到指定路径.
 - write_keypoint path/: 在指定路径输出包含人体姿态数据的 JSON, XML 或 YML 文件.
 - process_real_time: 对于视频，可能在实时运行时，跳过某些视频帧.
 - disable_blending: 如果 --disable_blending=True，则在纯色背景上渲染估计结果(如 keypoints skeletons 和 heatmaps)，而不显示原始图像. Related: part_to_show, alpha_pose, and alpha_pose.
 - part_to_show: 可视化的预测通道(Prediction channel).
 - display 0: 不打开可视化显示窗口. 对于服务器部署和 OpenPose 加速很帮助.
 - num_gpu 2 --num_gpu_start 1: 多 GPUs 时，设置开始的 GPU id. 默认使用所有可用的 GPUs.
 - model_pose MPI: 采用的模型 Model，影响 Keypoints 的数量、运行速度和精度.
- 具体调试参数可通过看 flags.hpp 文件中的说明，可在源文件中通过添加 GLAGS_为头的函数，进行参数修改，作用于全局。

3. 实验结果



输入 video o			
			
	调试参数	<code>--video examples\media\video.avi --face --hand --write_json output_json_folder/</code>	
	default		<code>--part_to_show 1</code> 
	<code>--part_to_show 2</code>		<code>--part_to_show 3</code> 
	<code>--part_to_show 4</code>		<code>--part_to_show 9</code> 

```
--disable_blending=True
```



输入
image



```
G:\OneDrive\Givertopense-build\G4Release\O3_keypoints_from_image.exe
Starting OpenPose demo...
Configuring OpenPose...
Starting thread(s)...
Auto-detecting all available GPUs... Detected 1 GPU(s), using 1 of the
body keypoints: Array{T::tostring():
[6] 981928 152 783838 0 350525
[7] 356888 203 023315 6 675885
[8] 375839 193 225667 6 668282
[7] 603498 280 886689 0 871789
[8] 540741 205 326309 0 950468
[6] 853237 219 943436 0 539141
[2] 380702 342 066010 0 904770
[2] 904938 339 478516 8 882377
[12] 724945 422 460234 0 665996
0.00000 0.00000 0.00000
0.00000 0.00000 0.00000
0.00000 0.00000 0.00000
0.00000 0.00000 0.00000
0.00000 0.00000 0.00000
0.00000 0.00000 0.00000
0.00000 0.00000 0.00000
0.00000 0.00000 0.00000
0.00000 0.00000 0.00000
0.00000 0.00000 0.00000
Face keypoints: Array{T::tostring():
Left hand keypoints: Array{T::tostring():
```

[illegible][illegible]

```

[WARN] global C:\build\wincap-build-rind6\c65\opencv\modules\core\src\misc\
mix_operations.cpp (1034): cv-Mat_1d::Address::assign OpenCV's processing of multi-
dimensional arrays might be changed in the future: http://github.com/opencv/opencv/issue/56173
NOTE: In addition with the user flags, this demo has auto-selected the following flag set:
      -hmatmats add_parts -hmatmats add_bkg -hmatmats add_PAFs
OpenPose demo successfully finished. Total time: 2.768159 seconds.

```

异步自定义输入输出处理

