> 2022-05-03、2022-05-07—2022-05-08、2022-05-18—2022-05-22
>
> 计算机视觉导论 (北京大学 2022 春)
>
> 第三次课程作业笔记 by PkuCuipy

> ⚠️ 请独立完成课程作业!
>
> 报告及代码仅供参考, 欢迎 issue 讨论代码中的问题和改进办法, 提前表示感谢!
>
> 另附我的完成时间如下, 供任务量参考:
>
> 第一题: 2h+
>
> 第二题: 3.5h+
>
> 第三题: 2.5h+
>
> 第四题: 6h+
>
> 第五题: (a) 3h+ (b) 1h+ (c) 6h+

1. Backprojection: Transform a Depth Image to a Point Cloud (15')

- 它这个 `depth_img` 的存储很有意思, 它是利用 png 的三个通道的后两个 (g & b), 然后用「整数÷某个定值 C」的方式编码浮点数, 其中整数用 g×256 + b 来编码, 这样精度是 16 bit 的. 比如 g 是 10, b 是 139, C 取 8000, 则对应浮点数 $25739/8000$.

```
depth_img = cv2.imread('depth.png')
dpt = depth_img[:, :, 2] + depth_img[:, :, 1] * 256
dpt = dpt * 0.00012498664727900177  # ≈ 1/8000
```

- 关于深度图的「深度」是什么? 之前一直理解错了, 之前以为物体到「眼睛」的距离, 但其实是物体到「视平面」的距离, 也即 z 值!
- 这个题本身倒是不难, 因为三维点映射到二维点用的矩阵 K, 那么反过来就是用 K^{-1} 映射.
- 单纯这样映射是还剩一层 ambiguity 的——因为只知道在三维空间的一条线上, 而不知道具体在哪个点.

  这时深度图派上用场, 通过深度图来确定 z, 从而知道在这条线的具体哪个深度上.

---

2. Sample a Point Cloud from a Mesh (35')

> 代码见文件夹 `/mesh_pc`. 不明白为什么要用 `.ipynb`, 反正我是先在 `.py` 写的, 然后再复制到 `.ipynb` for submission.

(a) Uniform Sampling (15')

- 要实现的算法流程是:
  1. 首先按照三角形的面积为权重进行一堆三角形的抽;

     这里涉及三角形面积的计算. 我用海伦公式实现的, 不过现在意识到应该也可以用行列式来算.
  2. 然后在每个三角形再抽一个点.

     算法参考的这篇博客, 也可以直接参考更新后的 Slides.

(b) Farthest Point Sampling (15')

- 这个题不要求并行, (其实确实就不能并行, 因为循环间有依赖!), 但每个循环内部的并行还是可以做到的.
- 一个细节是, 这里 `remains` *理应*是动态减小的, `results` *理应*是动态增加的.

  但考虑到 `np.ndarray` 实现为固定空间占用的数组, 因此每一轮循环都真的去增删是要重新开辟内存并复制的, 代价高昂!

  因此使用如下方法实现动态容量:

  - 实现 `remains` 的动态减少: 把删除的那行与倒数第 i 行交换, 并将最后的那 i 行*视为*无效的.
  - 实现 `results` 的动态增加: 预分配完整的数组, 第 i 轮循环的新增向量存储在第 i 行, 而第 i 行之后的行则目前仍*视为*无效的.

  > 在第 i 轮循环中被「视为无效的」的行不能参与这一轮中距离的计算!

(c) Metrics (5')

- EMD 这里作业允许直接调包. 试图阅读包的源码 (~ 50 行), 不过它是调了一个求解器, 加了一些约束, 我完全没看懂...

  后来看到这篇知乎文章, 里面大概讲了如何把 EMD 化为线性规划问题从而用求解器求解. (但我其实没仔细看hhh)
- 这个包的 EMD 和 Slides 上的大概是不太一样的, 因为求出来的数非常小! (否则理应 EMD 是大于 0.5 倍的 CD 的!)
- 但仍然是可以比较「谁对 sampling 更 sensitive 的」,

  比如我这里考察了 $\frac{\sigma(EMD)}{\mu(EMD)}$ 和 $\frac{\sigma(CD)}{\mu(CD)}$, 发现后者更小.

  这就验证了 Slides 上的说法: CD 更稳定, 而 EMD 则对 sampling 更敏感.
- 后来助教说实现 CD 的时候把 Sum 换成 Mean. 嗯, 这样就和 EMD 的包给出的是同一个量级的东西了. 嗯.

---

3. Marching Cube (25')

> 代码见文件夹 `./marching_cube` .

- 这道题允许用 for 循环, 数据量也不大, 所以写起来自在很多.
- 算法参考这篇博客即可, 也可以参考它的原文.
- 关键代码其实就这几行, 弄清楚原始的作业代码框架的每个变量是什么意思就好:

```
P1 = xyz + edge[:3]     # P for Point: [float; 3]
P2 = xyz + edge[3:]
V1 = grid[tuple(P1)]    # V for Value: float
V2 = grid[tuple(P2)]
P = P1 + ((isovalue - V1) / (V2 - V1)) * (P2 - P1)
```

- Slides 上也提到这种算法本身会面临一些 "ambiguity", 解决办法大概是参考这篇论文.

  但事实上我实现算法的时候完全没有考虑这一点 (博客里也没有考虑这一点), 最后实现出来的效果也非常正常——可能因为这种 "ambiguity" 对于一般的几何体而言比较罕见? 不确定.

---

4. PointNet (35')

> 不知道助教在搞什么, 这一坨代码框架这么多行, 注释都没有, 变量好多都不知道在干啥.
>
> 后来看到了 PointNet 的源码, 大概就是助教偷懒随便删删删得到的hhh

(a) Points Classification and Segmentation (20')

- 踩坑: Softmax 不是接收 `ReLU` 后的值, 而是应该直接接收 `Linear` 的输出!!! 也就是说, 如下注释里的东西不能加!

```
self.mlp = nn.Sequential(
    nn.Linear(in_features=1024, out_features=512),  nn.ReLU(),
    nn.Linear(in_features=512, out_features=256),   nn.ReLU(),
    nn.Linear(in_features=256, out_features=k), # nn.ReLU() 这里不能加这个! 加了性能骤降!
)
return F.log_softmax(self.mlp(x), dim=1)
```

  这也有道理:

  「因为这都最后一层了, 总不能把一些概率直接抹零啊.. 那既然不能抹零, 那不就是恒等映射? 那这个 `ReLU` 有何意义!」

(b) Point Feature Visualization (15')

- 就是说, 「正常」 PointNet 在 forward 的时候, 是对 nPoints×nFeats 的矩阵进行 `dim=0` 的 `MaxPool`, 从而无论有多少个点, 都是得到定长的向量作为 Global-Feature;

  而「这里」(可视化) 的话, 就要求每个点都得有那么个值, 那就改为进行 `dim=1` 的 `MaxPool`, 这样得到 `(batchSize, nPoints)` 的矩阵, 每个点就有一个 value 了.

  最后把这个 value $\in$ R 通过一些规则映射到 R^3 (RGB色彩空间), 用于可视化.
- 但不是很清楚这样做有何意义.. 而且在代码实现上, 就是给之前的 `PointNetFeat` 类和 `FeatNetCls1024D` 打洞, 使得整个代码丑陋不堪... 不理解.

综上, 我认为 PointNet 这道题设计得有点摆烂了..

---

5. Mask RCNN (40')

(a) Prepare the dataset (20')

- 这里所谓 NMS 就是执行一个 `IoU` 的计算, 如果 `IoU` 太大 (比如 > 0.3), 就说明重叠太大了, 那就去掉这个图形.

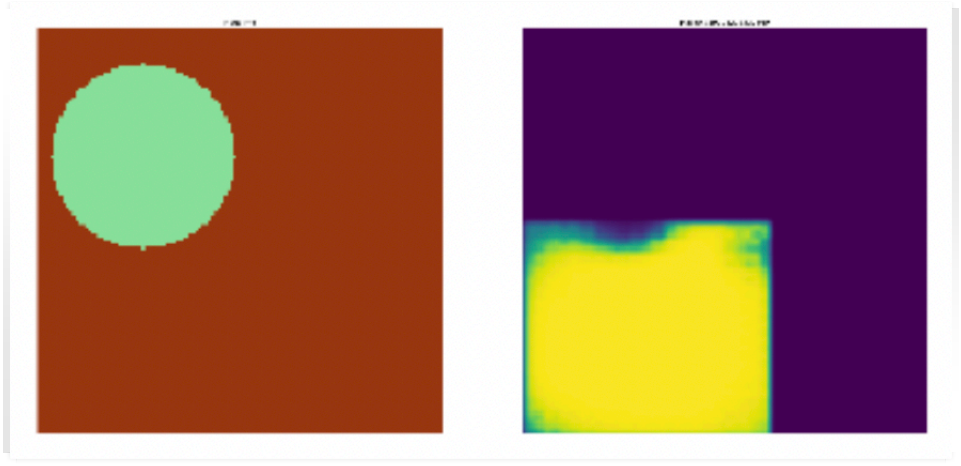
  NMS 可以参考这个源码的 `non_max_suppression` 函数.

  这里计算 IoU 是按照矩形包围盒进行计算的, 所以比较简单, 思路就是先求 intersect (可能是 0), 然后求 union.
- 在 `cv2` 的调用中发现一个神奇的 bug. 和这个人遇到的情况一样.

  大概就是说, `color = np.array([100, 150, 200])` 的情况下, `tuple(color)` 和 `tuple(color.tolist())` 是相等的, 但前者喂给 `cv2` 会报错, 后者则不会. 我猜测这和 python 底层对 int 的存储方式有关? 但理应这对库而言应该是 transparent 的呀..? 也可能是我搞错了..

(b) Train the Mask RCNN (10')

- 说是 Train, 其实是 Fine-tune. 进行 3 个 epoch 说是需要 1h, 但其实几分钟就能跑完.. 不过结果并不好看, 我感觉主要是 box proposal 的性能比较差劲, 那给定错误的 box 后, mask 再准也没用hhh 会预测出那种类似「抗体」的东西, 就是, 圈定了错误的 box 后, 本来是背景的东西预测成前景了. (个人看法: 自己生造的这个数据集, 和 pretrained 模型所用的数据集差别太大!)



《预测出了个 "抗体"》. 大概是模型在「错误的 box 预测」内进行了「正确的 mask 预测」

(c) Evaluate the Mask RCNN (10')

- 心力交瘁, 花了很久时间研究和写.

  我参考的这个博客, 但和若干朋友对答案后发现大家的写法*很不一样*.