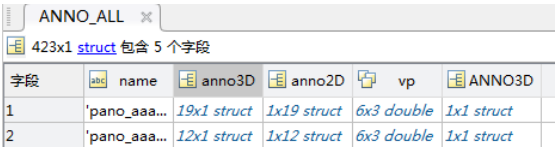
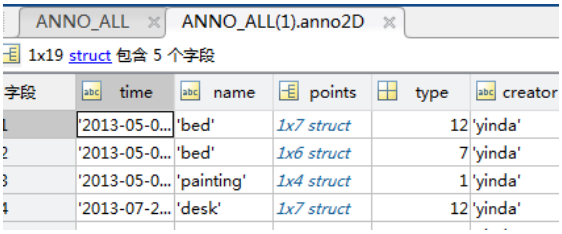
panoContext_data:
有 4 个文件夹，bedroom，bedroom_exp 和 living_room，liviing_room_exp
_exp 是原 PanoContext 算法里用于计算 room alignement 的，不用管。

bedroom:
423 张全景图，其中有对应.json 的 393 张；提供一个 IMGILST.mat 标示全景图是否有.json 对应，提供一个 ANNO_ALL.mat，里面有对 object 的标注，2D 投影和 3D 点坐标都有，另外还有 vp 的标注。但是 3D 点(有两组坐标)和 vp 的标注有点看不懂。。
后来发现，ANNO_ALL.mat 里还有部分图的 room layout 标注


ANNO_ALL

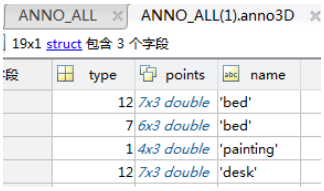
423x1 struct 包含 5 个字段

字段	name	anno3D	anno2D	vp	ANNO3D
1	'pano_aaa...	19x1 struct	1x19 struct	6x3 double	1x1 struct
2	'pano_aaa...	12x1 struct	1x12 struct	6x3 double	1x1 struct


ANNO_ALL | ANNO_ALL(1).anno2D

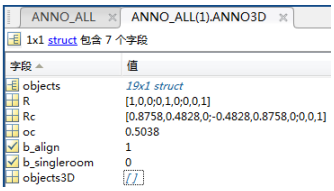
1x19 struct 包含 5 个字段

字段	time	name	points	type	creator
1	'2013-05-0...	'bed'	1x7 struct	12	'yinda'
2	'2013-05-0...	'bed'	1x6 struct	7	'yinda'
3	'2013-05-0...	'painting'	1x4 struct	1	'yinda'
4	'2013-07-2...	'desk'	1x7 struct	12	'yinda'


ANNO_ALL | ANNO_ALL(1).anno3D

19x1 struct 包含 3 个字段

段	type	points	name
	12	7x3 double	'bed'
	7	6x3 double	'bed'
	1	4x3 double	'painting'
	12	7x3 double	'desk'


ANNO_ALL | ANNO_ALL(1).ANNO3D

1x1 struct 包含 7 个字段

字段	值
objects	19x1 struct
R	[1,0,0;0,1,0;0,0,1]
Rc	[0.8758,0.4828,0;-0.4828,0.8758,0;0,0,1]
oc	0.5038
b_align	1
b_singleroom	0
objects3D	[]

但是拿到 LayoutNet 提供的数据后，发现 PanoContext 的 2D 投影标注不对，更像是经过平移的标注？

Livingroom:
283 张全景图，其中有对应.json 的 283 张；ANNO_ALL.mat 中还多包涵一个 anno_valid，每张图标注 0/1，求和是 213. 还不明白是啥。ANNO_ALL 变量还多一组 3D 坐标(看不懂是多 1 组还是多 4 组，也不知道是干啥的)

.json 文件是标注文件，其中包含物体 3D bounding box 顶点投影到 2D 图像上的坐标(根据顶点的可见情况，分成 9 类)。--不重要

TODO LIST

1. ~~搞到 Stanford 2D-3D 数据集(layout)的图像和 gt~~

2. ~~2.~~ 搞明白 PanoContext 数据集(layout&object)的 layout 和 object 的 3D 数据怎么用，~~或者找 LayoutNet 要(估计只能要到 layout 的)~~

   此外，PanoContext 的 object 的 3D 框标注点不全，分了 9 种情况，需要考虑如何应对。

3. 已知 3D 框在全景图上的投影点 8 个，水平方向该怎么连?
   ->按照 panocontext 提供几何关系的那个文档应该可以算

4. 关于 3D Geometry for Panorama 的文档：

如果要想从 2D 全景图重建 3D 的表达，似乎需要知道 2D 全景图中 ground plane 的语义分割结果？

Note:

Panocontext 提供的映射关系是从 0 开始标的；
我自己生成的数据从 1 开始标的；
有 4 张 90° 透视图拼全景的 code，效果还不错；

Discuss

PanoContext 的 layout 数据找到了，toolbox 得到的那个 layout 是怎样得到的？　　用的标注；
两个红色的 vp 点是啥？　　无限远处的 vp；
从全景拆分成透视图，只拆水平视角是否能够完整的恢复原来的全景？还是说会像 stanford2D-3D 那种形式？　　不能，只能恢复水平视角部分，类似 stanford2D-3D

从 2D 全景图重建 3D 的表达，ground plane 的语义分割结果？ --人标的，但是不知道被遮挡的地面怎么标？

7.13.2018

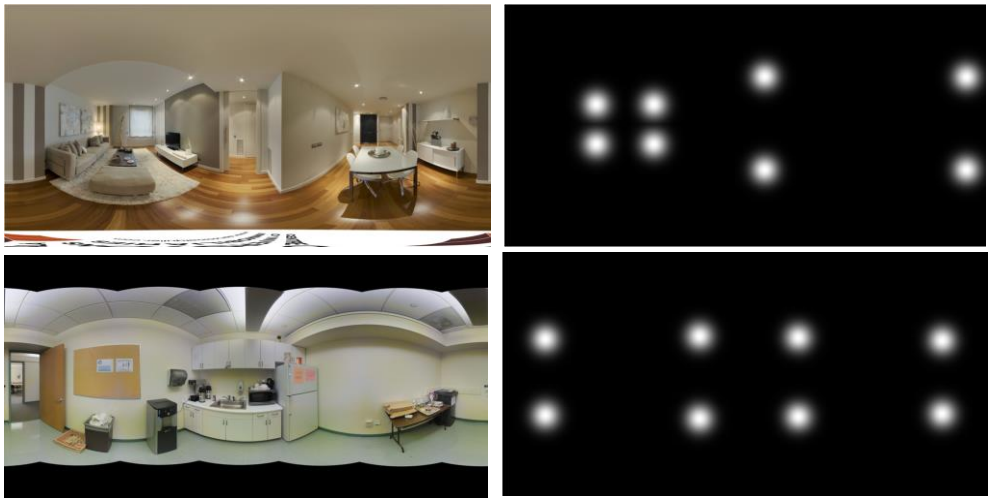
把我从 LayoutNet 分好的全景数据交给谢超宇，让他拆分成 Perspective images 和对应的 gt；

数据说明：

两个文件夹/PanoContext 和/Stanford2D-3D 分别是 2 个数据集的数据，

PanoContext 包含 3 组(train,val,test)6 个文件夹，img 里是全景图，cor 里是关键点的 ground truth，另外 3 个.txt 不用管；

Stanford2D-3D 包含 6 组 12 个文件夹，同样包含全景图和关键点的 ground truth；



拆分需求：

1.把全景图和相应的 cor 标注进行拆分，拆分结果保持一致，拆分出来的图片保持原来的目录结构进行储存，即可以按照我原来的文件目录划分规则、新建文件夹然后保存；

(panocontext 和 stanford 2D-3D 两个数据集处理结果分开放，img 和 cor 分开放，文件夹命名参考我的命名方式)

2.拆分的结果保存格式和我的.mat 文件也保持一致，即都是 double 型，数据取值范围也一致，存成.mat 型；

3.拆分方式：

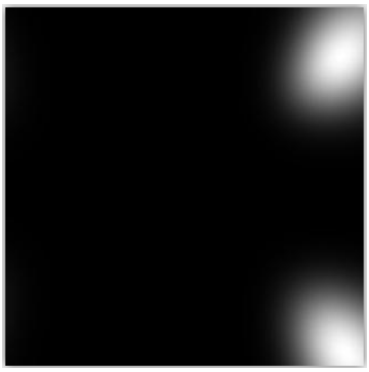
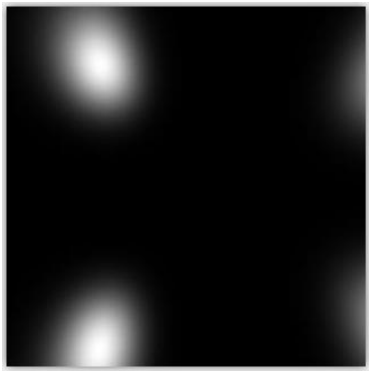
panocontext：仅水平方向拆一组，然后水平方向+竖直方向拆一组，分开放两个文件夹。水平方向拆 N 张：原则是同一墙角点出现在 3 张以上的透视图中；

stanford2D-3D：只能拆水平方向，就按水平方向拆，看图应该是只能固定拆 6 张？试试看拆出来好看就行；

7.16.2018

拿到拆分后的 perspective images 两组，分别 FOV 是 90-90 和 60-60，其中 stanford2D-3D 只

有一组；
发现问题，拆分后的 perspective images 的 gt 存在畸变，而且畸变方式和与其在全景图上的 location 有关，这不适合网络学习，于是重新预处理数据，生成新的 gt(正圆)；
同时还存在一些关键点不在该视角下，但是其高斯有一部分在该视角的情况；

后期网络预测出来的 perspective image 上的高斯分布，按照与原图一致的映射规则映射回去，期望生成多个高斯分布，可对其求平均以提高 keypoints 的预测正确率；

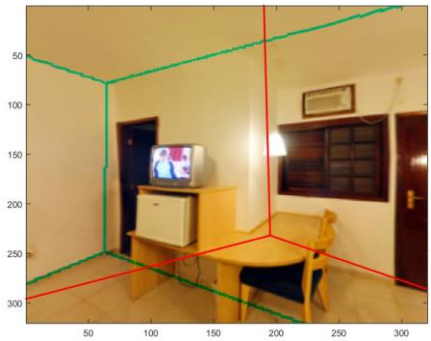
另外，由全景分出来的透视图，仅有中心部分的关键点，没有 semantic boundaries 和图像边界的交点，室内拓扑结构和原来我们用的不一样。

考虑两套方案：

1. 在全景图上根据 8 个点，进行连线，得到 boundary map，然后拆分，利用 boundary 与图像边界交点作为相应关键点；
2. 重新设计 room layout estimation 的网络，使之只预测三面墙交汇的关键点，不预测落在图像边界的关键点(还可以考虑不在视野内但是高斯 gt 部分在图内的情况，如上右图)。

先做方案 1：我先写个脚本把 8 个点的坐标全抠出来然后给谢超宇。 Done
反馈结果：8 个点的储存顺序需要知道

关于方案 2 的思考：虽然预测的东西变少了，任务变简单了。但这相当于做的任务完全变了，因为数据形式整个就变了，原来的 11 类拓扑有 5 类无效，按理说可以借鉴 roomnet 的方法对房间拓扑进行分类，按照不同的拓扑类别计算 loss(算了下 label 应该是 12 通道)。同时也可以忽略拓扑类别，每张图只标三墙交汇点的高斯(即 label 为单通道)，每张图仅预测这样一组三墙交汇的点。不论选取什么样的实现方式，预计把问题简化应该会使学到的三墙交汇点更加准确，但是工程量会变大，数据处理和网络修改都有点大。
另外，方案二还有不受拓扑分类正确率约束的优点。



7.17.2018

规范下昨天说的 8 个点的储存顺序，按照坐标的空间关系进行了重排。 Done

把重排的结果交给谢超宇让他帮我用 panocontext 的 toolbox 提取 boundary map

并进行拆分(60-60 和 90-90)：


....

发现存在如图所示的问题，这个是数据问题，没法改，但是这种情况不是多数。

另外，拆出来的 boundary map 不符合要求，和背景像素区分度不够(没置 1)。

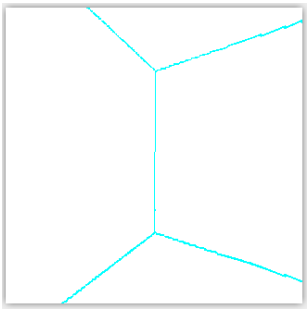
061	0.3081	0.3129	0.3175	0.3189
111	0.3131	0.3178	0.3248	0.3331
124	0.3175	0.3237	0.3289	0.3357
199	0.4050	0.3305	0.3333	0.3365
501	0.8294	0.7128	0.5962	0.4798
344	0.7531	0.8720	0.9914	0.8901
370	0.3407	0.4592	0.5802	0.7009
363	0.3367	0.3353	0.3366	0.3376
298	0.3343	0.3369	0.3410	0.3444
207	0.3264	0.3315	0.3382	0.3457
159	0.3195	0.3245	0.3326	0.3409

让谢超宇重新拆...

开始写文档，写了下 introduction 和 method 的总起部分；

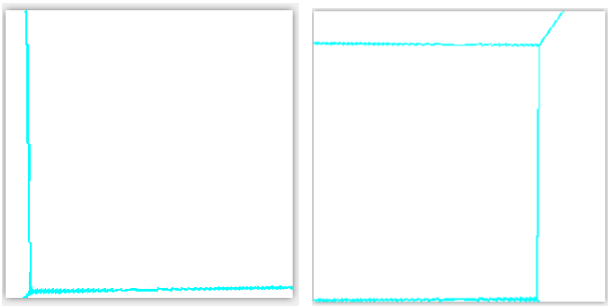
7.18.2018

谢超宇帮我把数据做了重新拆分，并生成对应的二值图像。



接下来需要由这类图像得到我们常用关键点形式的关键点坐标。

考虑使用 hough 变化检测直线然后找与边界的交点，但是发现有许多语义边界直线非常短的情况，可能直线检测不准：



然后考虑找连通域，用连通域在图像边界的交互处作为边界关键点，但是存在因为全景图像拆分而出现语义边界断裂的情况：

另外因为拆分带来的变形，使得语义边界不是单像素宽。
开始着手写数据处理的脚本，把以上的 boundary maps 作为输入，然后生成符合 roomnet 需求的形式。

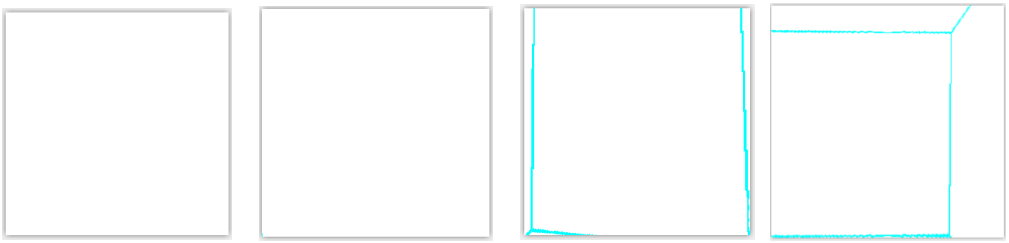

7.19.2018
继续写数据处理的脚本，思路是这样的：

1. 先 boundary maps 做膨胀[3,3]，去除可能存在的断裂情况，然后检测连通域数目；

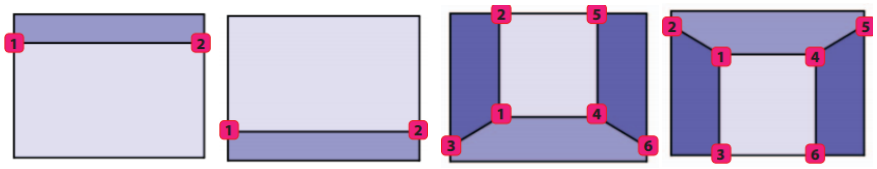
2. 同时用原 boundary maps 检测在图像边缘上的点的位置及数目，并且按照 4 条边存，方法是先求个前向差分，然后找[+1,-1]的对数，取匹配的[1,-1]的中心坐标作为边缘关键点的位置；

3. 根据连通域数目和关键点的数目及位置，判断拓扑类别以及符合 RoomNet 数据形式的点序；

其中加了许多判断，先剔除不合理的拓扑，或者数据质量较差的情况。



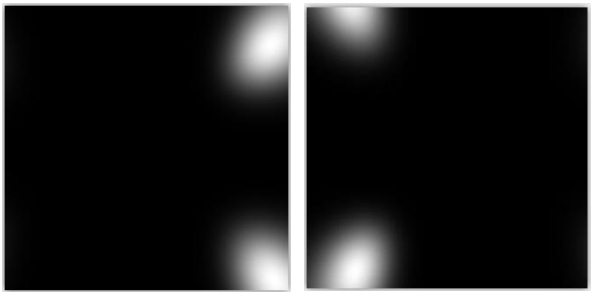
写到一半发现有个之前没考虑到的问题：



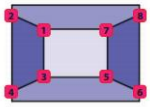
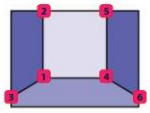
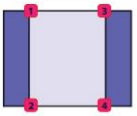
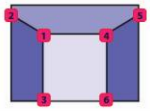
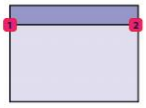
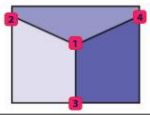
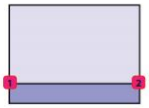
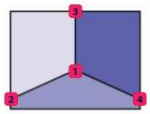
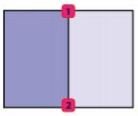
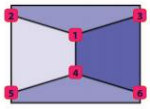
对以上 4 个拓扑，2 种情况，只知道边界关键点的位置和连通域数目，还是存在歧义。
后来考虑到相机在拍摄时的姿态是特定的，对前两张情况，按 90-90 拆分时，如果关键点在图像左右边界中点以上，则视作天花板/中墙，在边界中点以下则视作中墙/地板；按 60-60 拆分时，水平视角与 90-90 的规则相同，往上看的视角则默认为天花板/中墙，往下看的视角默认为中墙/地板。

对后两种情况，大致翻了下数据集，没看到…估计应该很少，在 4 连通域 4 边界关键点的情况下加了个判断，等遇到了再说吧，~~应该可以根据 4 个边界关键点的位置关系判断。~~

在写到第三步的时候，需要图像内部关键点的坐标。发现从之前谢超宇给的以下这种数据里不方便得到，因为经过全景图像拆分后，原来图像的最大值已经不是固定值了。



所以让谢超宇帮我把全景图的 8 个关键点坐标投到 perspective images 上计算下坐标，并且标注上/下关键点。

Type	Room Layout	Example Image	Example Layout	Type	Room Layout	Example Image	Example Layout
0				6			
1				7			
2				8			
3				9			
4				10			
5							

之前只考虑用连通域和边界关键点来判断拓扑类别以及关键点点序，其实考虑的不够成熟，目前打算加入图像内关键点的上下类别以及位置，具体判别方法归纳如下：

0 类： 5 个连通域，4 个边界关键点，可确定此类拓扑。
关键点点序：内部关键点点序已知，边界 4 个关键点可以通过相互的位置关系确定点序。

1 类，2 类，5 类： 4 个连通域，4 个边界关键点可确认属于这三类拓扑，这三类拓扑类内可通过内部关键点的标签区分。
关键点点序：内部关键点点序已知，边界 4 个关键点可以通过相互的位置关系确定点序。

3 类，4 类： 3 个连通域，3 个边界关键点可以确认属于这两类拓扑，类内可通过内部关键的标签区分。
关键点点序：3 类-内部关键点以下的点为第三关键点，然后剩下的左右分别为 2,4 点。
4 类-内部关键点以上的点为第三关键点，然后剩下的左右分别为 2,4 点。

6 类，7 类： 3 个连通域，4 个边界关键点可以确认属于这两类拓扑，由于全景图中的墙-墙边界都是和地面垂直的，所以 7 类的 4 个边界关键点一定在上下两条边，且 6 类一定不存在这种情况。
关键点点序：6 类-通过相互的位置关系确定点序
7 类-根据拓扑类别，以及边界关键点所在边即可快速判定。

8 类，9 类，10 类： 2 个连通域，2 个边界关键点；第 10 类两个关键点一定在上下边界，8,9 类不存在这种情况；8,9 类可通过第 7 页的规则区分。
关键点点序：直接根据拓扑类别，关键点两两的上下、左右关系确定。

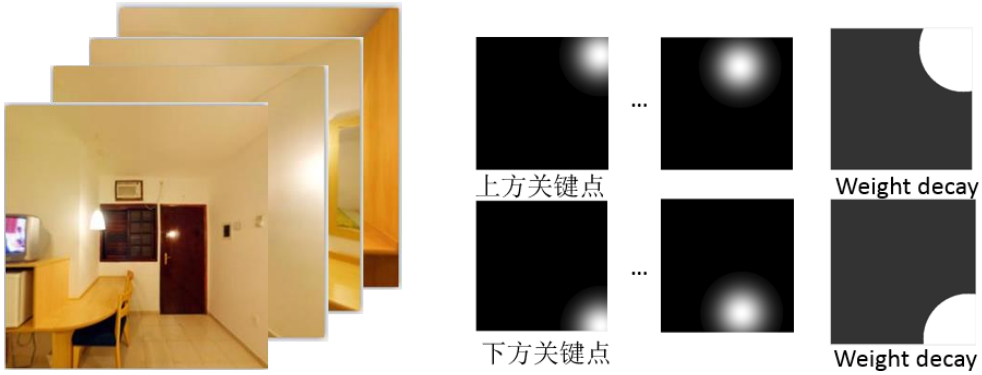
另外，发现如果把语义边界图按三类边界的方式，判断点序的规则可以进一步简化。先按原思路写吧，重新再做一组数据也麻烦，如果原思路太复杂再考虑引入这种表达。

7.26.2018

上周决定使用方案二，于是按照方案二完成网络的搭建，以及数据的生成，以下归纳最终的数据形式及生成过程：

输入：全景图及对应的 ground truth

输出：众多视角的透视图以及对应的 ground truth 和 weight decay mask



对每张透视图，配对一个双通道的关键点 ground truth，每个通道对应上/下关键点的高斯，高斯的标准差对 320x320 的图像来说取 40，在距离关键点 2.5σ以外的区域(函数值小于 0.0439）视为背景置零。同时还配对双通道的 weight decay mask，背景区域值为 0.2，关键点区域值为 1，用于训练。

对 PanoContext 和 Stanford 2D-3D 都可以使用以上方法进行数据生成，此外按照不同的视角范围(60-60 和 90-90)也可以按照以上方法生成。方便进行不同数据条件下的实验。

新的网络结构旨在服务于上述表达形式。
搭建好的网络结构，见文档&PPT。

```
python /ghome/dengrf/SecRoomNet/main.py --
testing=/gdata/dengrf/RoomNet/log/log10.2B/model.ckpt-39999 --
log_dir=/gdata/dengrf/RoomNet/log/logt/ --
test_dir=/gdata/dengrf/RoomNet/test/panoContext/temp_60.txt
```

实验设计：
可以考虑直接裸训 Sec-roomnet,以及在 LSUN 预训练的模型上 finetune，做一组对比；
原来的网络存在过拟合的问题，考虑可以尝试添加正则化；
尝试不同数据集的组合进行网络的训练；

startdocker -D /gdata/dengrf/SecLayout -u "-it" -c /bin/bash bit:5000/husy_py2_tf1.4

CUDA_VISIBLE_DEVICES=4,5,6,7 python /ghome/dengrf/SecRoomNet/main.py --log_dir=/gdata/dengrf/SecLayout/log/log1/ --image_dir=/gdata/dengrf/SecLayout/PanoContext/90/train/train_list.txt --val_dir=/gdata/dengrf/SecLayout/PanoContext/90/test/test_list.txt --batch_size=5

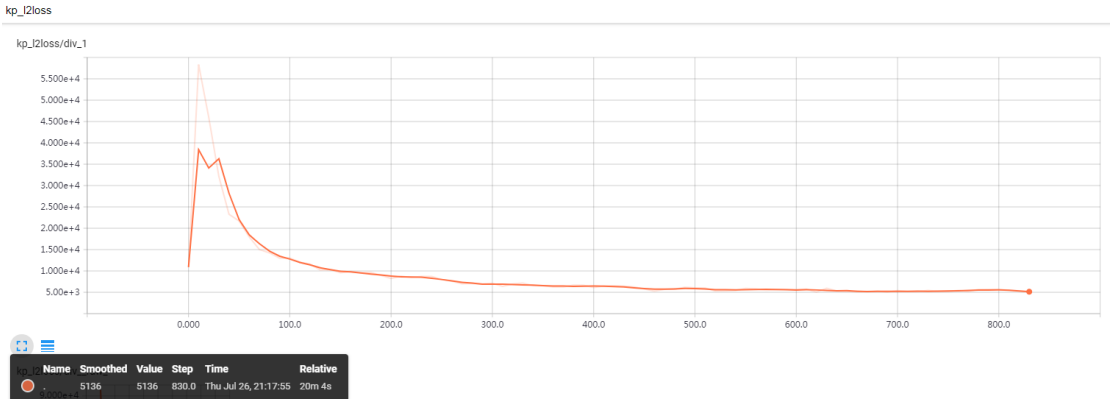
CUDA_VISIBLE_DEVICES=4,5,6,7 python /ghome/dengrf/SecRoomNet/main.py --testing=/gdata/dengrf/SecLayout/log/log4/model.ckpt-39999 --log_dir=/gdata/dengrf/SecLayout/log/logt/ --test_dir=/gdata/dengrf/SecLayout/PanoContext/90/test/test_list.txt

效果展示：
可以拍不同的房间，挑一下角度比较一致的，然后把不同房间的照片拼成一个新的房间；

训练老是报错：

```
Traceback (most recent call last):
  File "/ghome/dengrf/SecRoomNet/main.py", line 52, in <module>
    tf.app.run()
  File "/home/anaconda2/lib/python2.7/site-packages/tensorflow/python/platform/app.py", line 48, in run
    _sys.exit(main(_sys.argv[:1] + flags_passthrough))
  File "/ghome/dengrf/SecRoomNet/main.py", line 49, in main
    model.training(FLAGS, is_finetune=False)
  File "/ghome/dengrf/SecRoomNet/model.py", line 893, in training
    image_batch, kp_batch, weight_batch = Roomloader(shuffled_img, shuffled_kp, shuffled_W, batch_size, step)
  File "/ghome/dengrf/SecRoomNet/Inputs.py", line 59, in Roomloader
    KPmat = scio.loadmat(kp_names[i])
  File "/home/anaconda2/lib/python2.7/site-packages/scipy/io/matlab/mio.py", line 136, in loadmat
    matfile_dict = MR.get_variables(variable_names)
  File "/home/anaconda2/lib/python2.7/site-packages/scipy/io/matlab/mio5.py", line 292, in get_variables
    res = self.read_var_array(hdr, process)
  File "/home/anaconda2/lib/python2.7/site-packages/scipy/io/matlab/mio5.py", line 252, in read_var_array
    return self._matrix_reader.array_from_header(header, process)
  File "mio5_utils.pyx", line 673, in scipy.io.matlab.mio5_utils.VarReader5.array_from_header
  File "mio5_utils.pyx", line 738, in scipy.io.matlab.mio5_utils.VarReader5.array_from_header
ValueError: Did not fully consume compressed contents of an miCOMPRESSED element. This can indicate that the .mat file is corrupted.
```

训练 830 次(接近一个 epoch)的曲线长这样：



最后发现确实是数据问题，剔除出问题的一张就可以了，但是问题出在哪儿没有深究。

实验结果

目前的实验设计：

一、PanoContext 数据集，按照 90° 视角拆分，一张全景图只拆水平视角得到 12 张透视图，可得训练集 4965 张透视图，测试集 636 张透视图，暂时不用验证集数据；

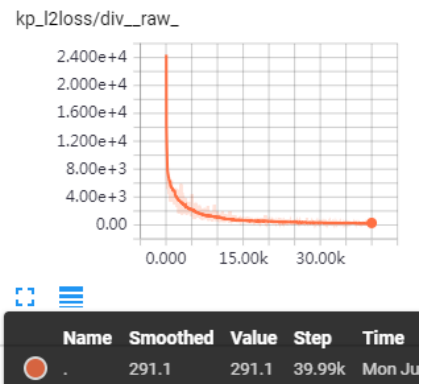
二、PanoContext 数据集，按照 60° 视角拆分，一张全景图可拆分成 24 张透视图，可得训练集 9935 张透视图，测试集 1272 张透视图，暂时不用验证集数据；

三、Stanford2D-3D 数据集，这个数据集只能按水平视角拆分，所以按 90° 拆分，一张全景图得到 12 张透视图，可得训练集 5240 张透视图，测试集 1356 张透视图；(running)

四、考虑如果不做定量比较，训练数据量应该越大越好。所以把 PC 和 ST 两个数据集按 90° 拆分得到的训练数据综合在一起，并且把 PC 数据集的验证集数据也拿来作为训练数据，可扩充得到 10755 张训练透视图。

调试网络超参数均在“实验设计一”的配置下完成。设计一、二已有实验结果，三、四实验还在跑，此外考虑到实验中出现的过拟合现象，按设计四多配置一组实验，在 loss 中增加一个正则化项。

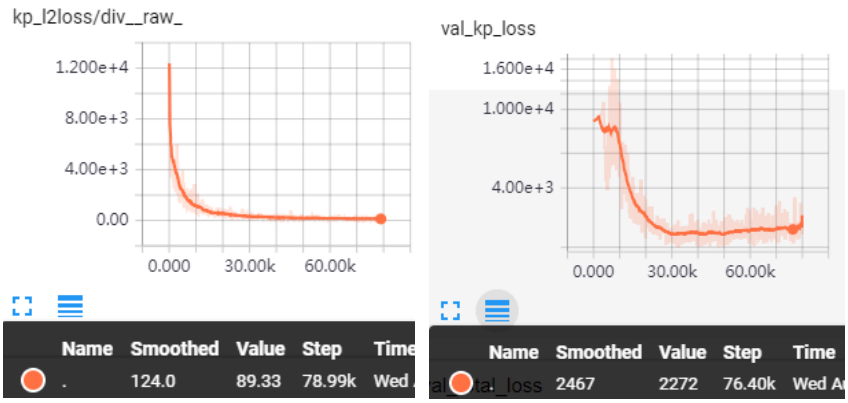
1. Sec4, log4, 90°, lr=0.00001, batch_size=5 (best setting)



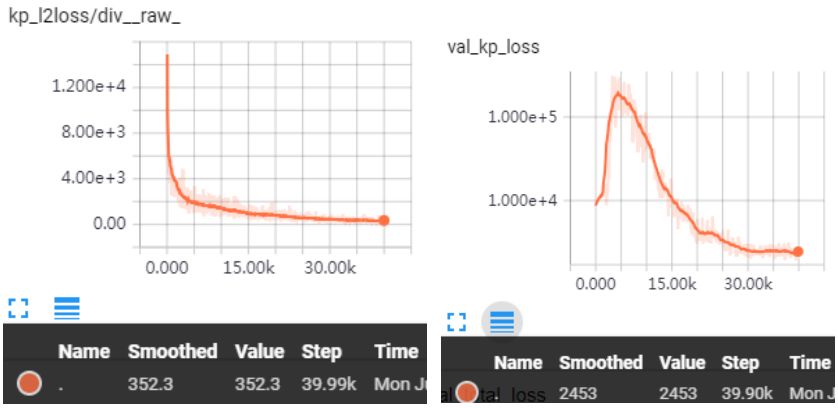
Training loss curve

validation loss curve (on testset)

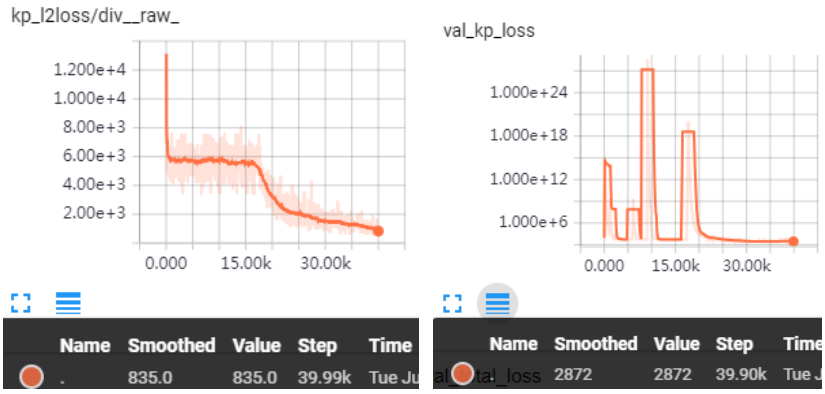
迭代到 8000 次：



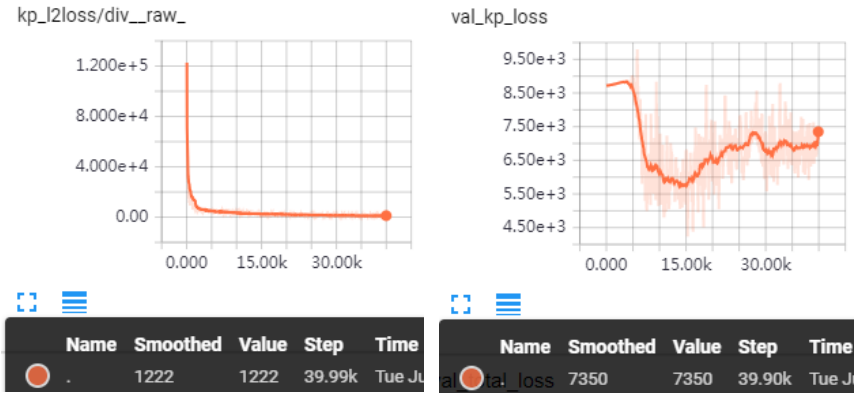
2. Sec5, log5, 90°, lr=0.0001, batch_size=5



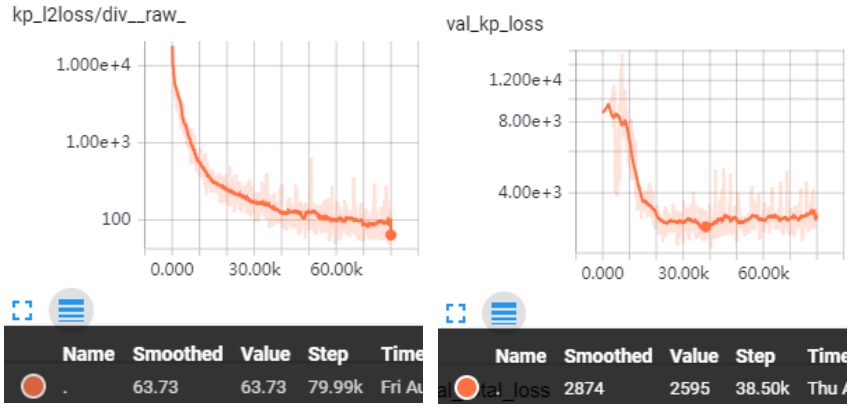
3. Sec6, log6, 90°, lr=0.001, batch_size=5



4. Sec7, log7, 90°, lr=0.000001, batch_size=5

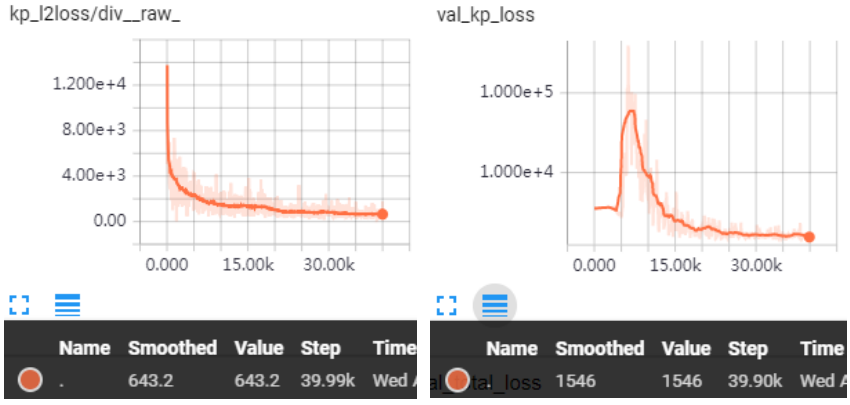


5. Sec9, log9,90°, lr=0.00001,batch_size=10

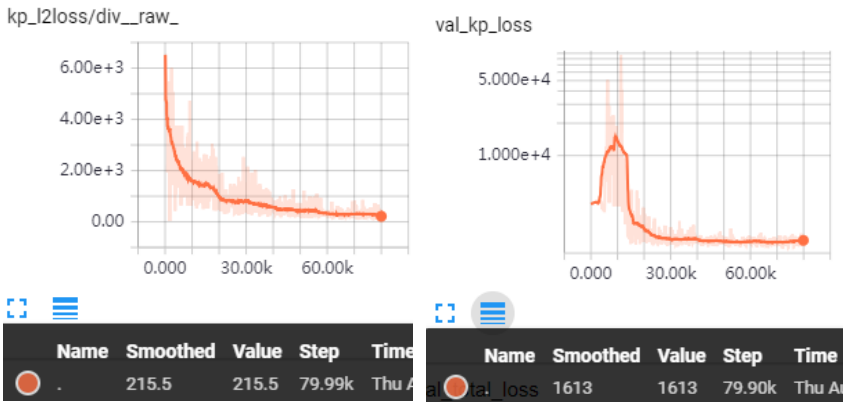


更大的 batch_size 基本没有影响(从验证 Loss 看相较于 batch_size=5 反而差一点点)，随着迭代次数增大，过拟合更明显。

6. Sec8, log8, 60°,lr=0.00001, batch_size=5



迭代到 80000 次：



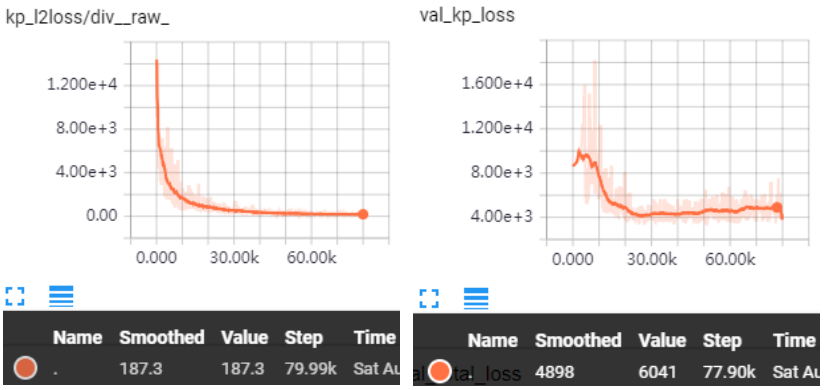
60°从 trainin loss 曲线上看，应该比 90° 相对难训，但是验证 loss 更小这个不知道是什么情况，可能是收敛到了局部最优？

所以迭代 40000 次应该就够了

实验设计三：

7. SecST2, logST2, Stanford 2D-3D, batch_size=5, lr=0.00001

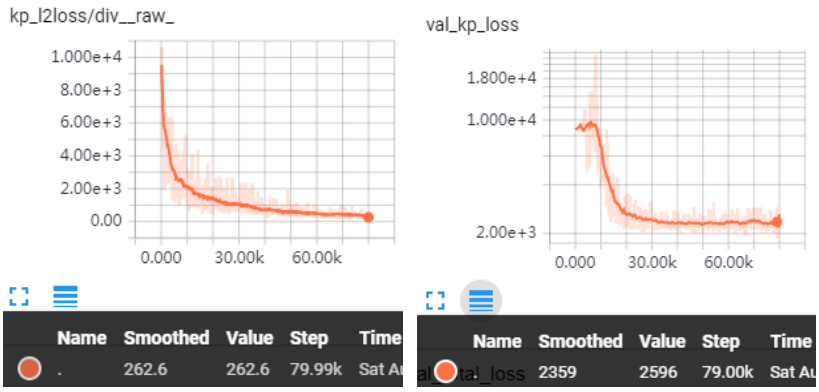
迭代到 80000 次：



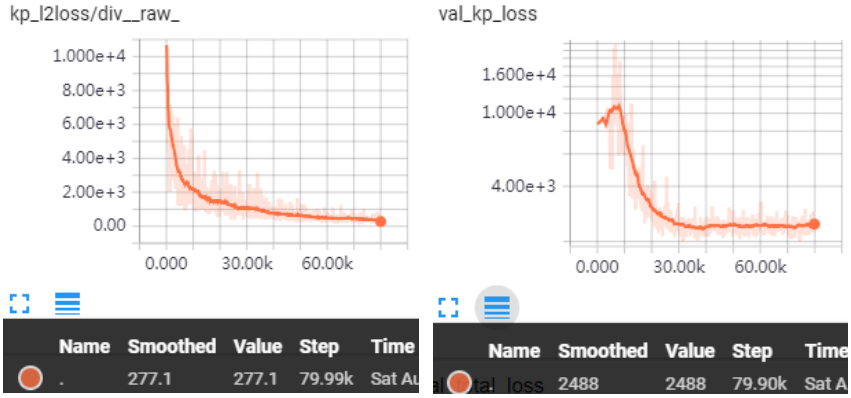
考虑过拟合问题明显，尝试添加一个正则化项，实验正在进行中。

实验设计四：

8. SecALL, logALL, PC+ST(90), batch_size=5, lr=0.00001



9. SecALL_norm, logALL_norm, PC+ST(90), batch_size=5, lr=0.00001 (仅添加 L2 正则化项)



没效果，可能是正则化项的权重太小了 wd=0.0005

10. SecALL_norm, logALL_norm, PC+ST(90), batch_size=5, lr=0.00001 (L2 正则化项 wd=0.005)

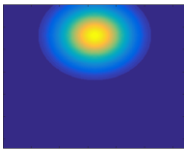
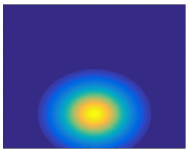
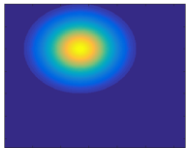
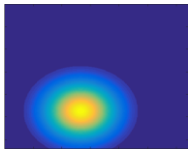
用 Sec4,log4 训好的模型(90°)，加上其滑动平均，在测试集进行整体测试：
average keypoints l2 loss is 3264.65
不用滑动平均时：
average keypoints l2 loss is 2289.12

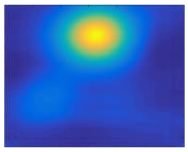
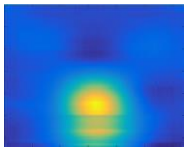
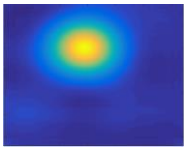
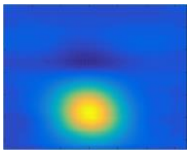
用 Sec8,log8 训好的模型(60°)，加上其滑动平均，在测试集进行整体测试：
average keypoints l2 loss is 1655.09
不用滑动平均时：
average keypoints l2 loss is 1463.42

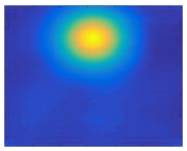
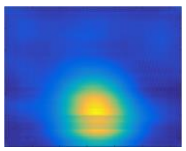
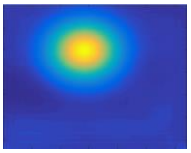
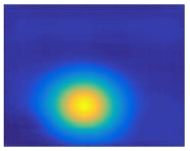
以上这个结果因为 l2 loss 和实际最终结果非直接相关，所以加/不加滑动平均哪种更好还未确定(按照实现 roomnet 的经验，加了滑动平均效果会好)，为了提高效率，暂时不在透视图上定量评估了，干脆把系统搭完，在全景图上做定量评估。

视觉上看，两种方案效果都挺好的，找的比较准。加滑动平均时，probability map 数值大概在 0~0.5x 的样子，即最大值点概率在 0.5 多一点。不加时，probability map 看起来更合理，在 0~1.1x 的样子，最大值点概率在 1 左右。

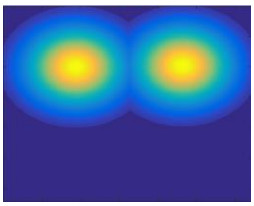
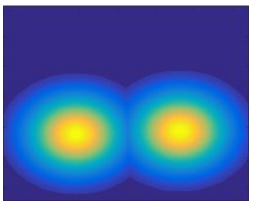
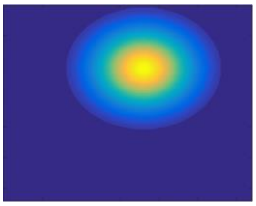
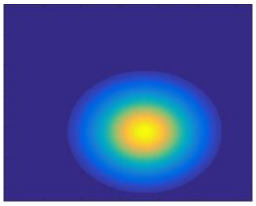
原图：  

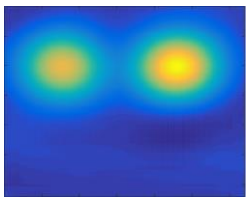
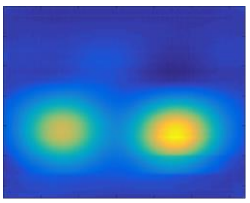
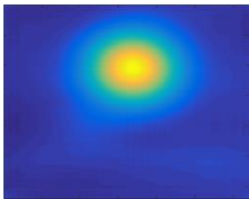
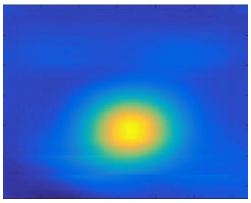
Gt ：    

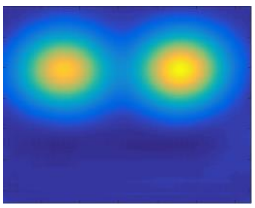
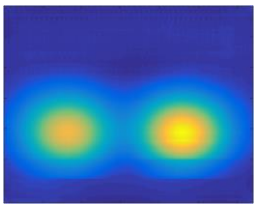
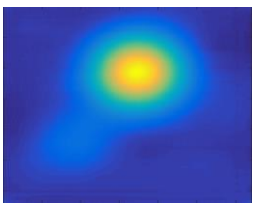
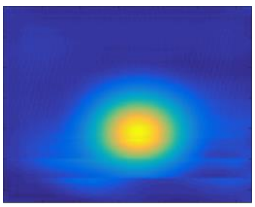
不加：    

加 ：    

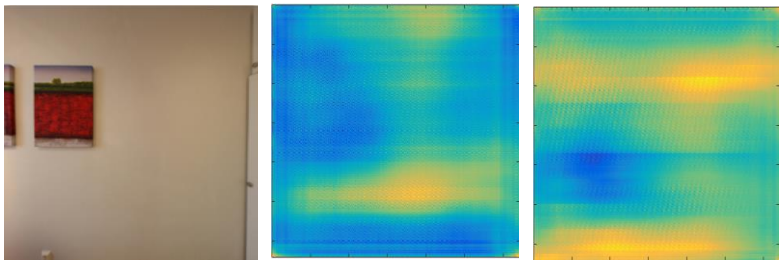
原图：  

Gt ：    

不加：    

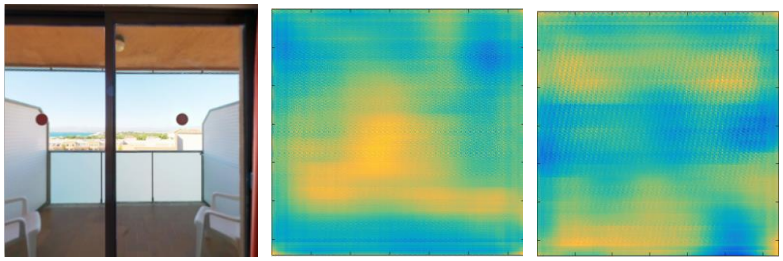
加 ：    

原图　　不加



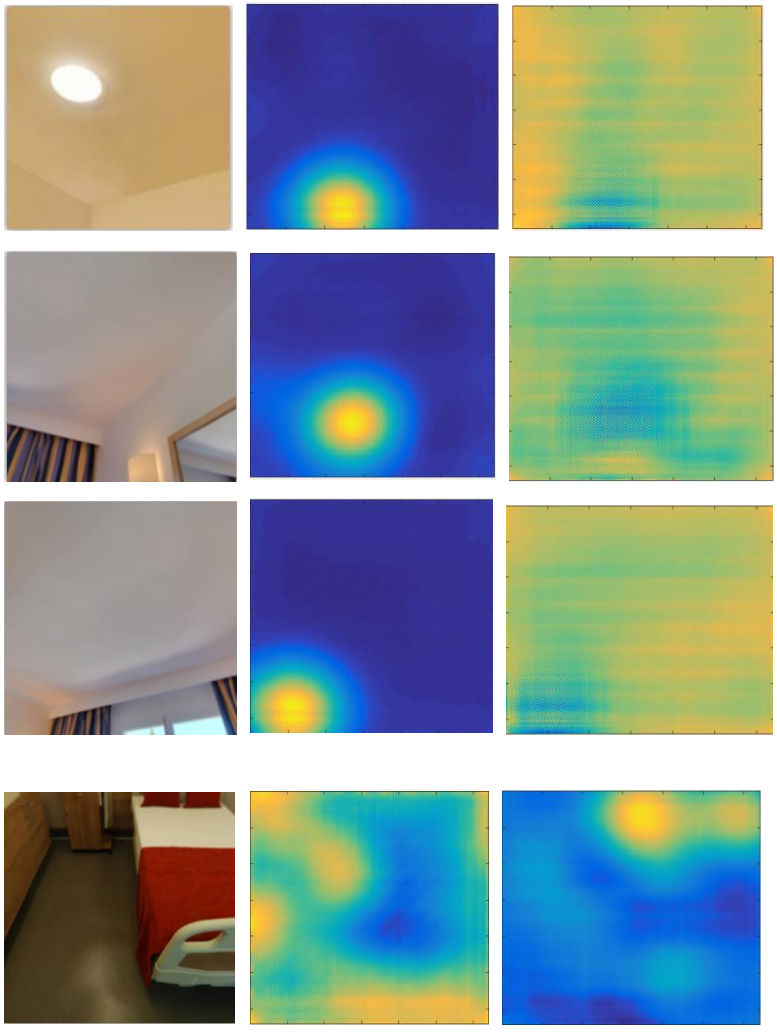
最大值 0.0578, 0.0832

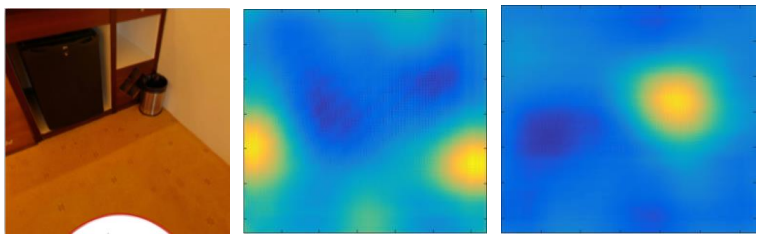
加滑动平均效果也差不多，比较平均。



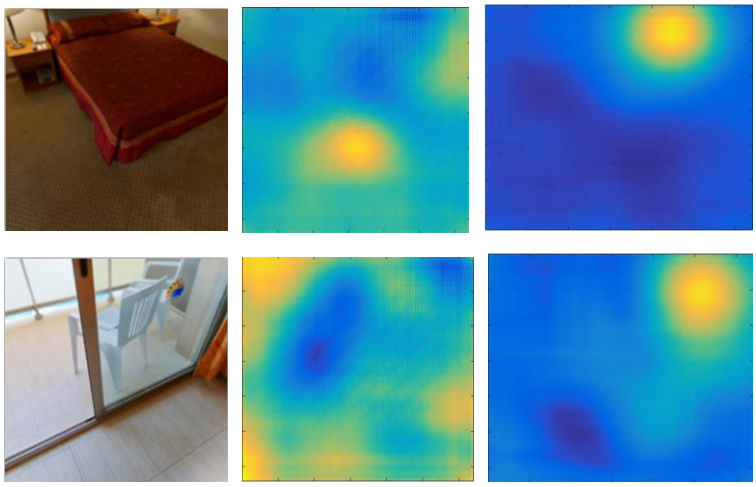
最大值 0.0506, 0.0614

60°视角图的部分结果：





(上部关键点可通过视角类别滤除)



结论：综上，从视觉上看，实验结果符合预期，视角 90° 比视角 60° 的透视图容易做。

最终在全景图上定量评估算法预测的性能，有以下三个指标：

Method	3D IoU (%)	Corner error (%)	Pixel error (%)
PanoContext [33]	67.23	1.60	4.55
ours (corner)	73.16	1.08	4.10
ours (corner+boundary)	73.26	1.07	**3.31**
ours full (corner+boundary+3D)	**74.48**	**1.06**	3.34

这是 PanoContext 上，先前两种算法的结果。
根据论文及部分已有的代码，完成了以上三个指标的评估脚本。

按最初谢用阈值化的后处理方法，得到的结果：
加滑动平均(46/53)： 3D IoU: 56.43% Corner error: 1.77% Pixel error: 6.69%
不加滑动平均(47/53)：3D IoU: 58.13% Corner error: 2.34% Pixel error: 6.38%

LOG 的后处理方法：
加滑动平均(53/53): 3D IoU: 56.72% Corner error: 2.73% Pixel error: 9.40%
不加滑动平均(53/53): 3D IoU: 53.14% Corner error: 3.76% Pixel error: 11.75%

加权去除‘边界’后 LOG 后处理：
加滑动平均(53/53): 3D IoU: 57.47% Corner error: 2.53% Pixel error: 8.73%
不加滑动平均(53/53): 3D IoU: 54.85% Corner error: 3.15 % Pixel error: 10.82%

用 LayoutNet 的后处理方法：

加滑动平均(53/53): 3D IoU: 55.03% Corner error: 2.44% Pixel error: 6.77%

不加滑动平均(53/53): 3D IoU: 57.96% Corner error: 2.54% Pixel error:7.25 %

先算 LOG 响应，然后用 LayoutNet 后处理：

加滑动平均(53/53): 3D IoU: 59.58% Corner error: 2.20% Pixel error: 6.78%

不加滑动平均(53/53): 3D IoU: 58.76% Corner error: 2.48% Pixel error: 7.54%

定性：3,11,28,32,34,40 （左边是 LOG 方法，右边是 layoutnet 后处理方法）

Notes

分上下两类关键点，在实验结果中发现，上部关键点比下部关键点找的准，这是很合理的，因为下部关键点更容易被遮挡。分开学可以避免这两类关键点的混淆，尤其防止下部关键点污染上部关键点的结果。

注意用不同数据集训练时，记得修改 TEST_ITER 参数，因为每个验证(测试)集的样本不一定都为 5 的整数。

一个训练数据扩充方案：能不能把全景图循环右移 N 度，然后再拆分？

能不能不限制输入图像尺寸必须统一为某个特定视角？比如输入既有 60° 又有 90° ？从网

络端来说，只用把这两种视角的图片放一起训就行了，不知道方不方便拼接？

最后可以在测试时，做一组水平 flip 的图，然后测完求平均。