Report
Ovchynnikov Kostiantyn

Task 1. Paper review

Paper

Title: LayoutNet: Reconstructing the 3D Room Layout from a Single RGB Image
Authors: Chuhang Zou, Alex Colburn, Qi Shan, Derek Hoiem
Link: https://arxiv.org/pdf/1803.08999.pdf
Tags: 3d, reconstruction, CNN, image, layout
Year: 2018

Summary

What:

- They propose an algorithm to predict room layout from a single image
- Method allows to operate directly on the panoramic image, rather than decomposing into perspective images as do recent works.
- Method compares well in speed and accuracy to other existing work on panoramas, achieves among the best accuracy for perspective images, and can handle both cuboid-shaped and more general Manhattan layouts.

How:

- Method is similar to RoomNet [https://arxiv.org/pdf/1703.06241.pdf], and differs by
    - can be applied both on perspective and panoramic images
- Firstly, they reproject 360 image to the 2D equirectangular projection. And, by using Line Segment Detector they detected candidate Manhattan line segments, which provides additional input features that improve the final performance of the Net.
- The input on they NeuralNet (LayoutNet) has 6 channels (3 - RGB panorama + 3 - Manhattan line feature map).
- LayoutNet uses Encoder-Decoder strategy, but they don't use separate encoders for image and Manhattan lines.
- Their decoder consists of the two branches.
    - First branch, layout boundary map predictor, results with 3-channel probability prediction of wall-wall, ceiling-wall and wall-floor boundary. It contains 7 layers of nearest neighbor up-sampling operation, each followed by a convolution layer with kernel size of 3 × 3. The final layer is a Sigmoid operation. Also, they add skip connections to each convolution layer.

- Second is 2D layout corner map (mC) predictor, follows the same structure as the boundary map predictor and additionally receives skip connections from the top branch for each convolution layer.
- They experiment with fully convolutional layers, instead of up-sampling+conv, but observed worse performance.

- Their regressor at the end of the network, which maps 2D corners and boundaries to 3D layout has 7 conv layers and 3 fc.
- They observed that direct 3D regressor fails due to the fact that small position shifts in 2D can have a large difference in the 3D shape, making the network hard to train.
- Loss consists of
    - Sum of cross-entropy error of the predicted pixel probability of the boundary map and corner map with ground-truth.
    - And Euclidean distance of the regressed 3D cuboid parameters to the ground truth.
- For the augmentation of the data, was used luminance, rotations,left-right flippings.
- For extension, they propose to predict more general Manhattan layouts
    - By thresholding the score of the sixth strongest wall-wall boundary they determine whether to generate four or six walls(L-shaped rooms).

Results:

- At this research, they used three metrics
    - Intersection over union 3D, averaged by images
    - L2 corner distance between predicted and ground truth
    - pixel wise accuracy between the layout and the ground truth
- They used two datasets:
    - PanoContext dataset (500 annotated cuboid layouts)

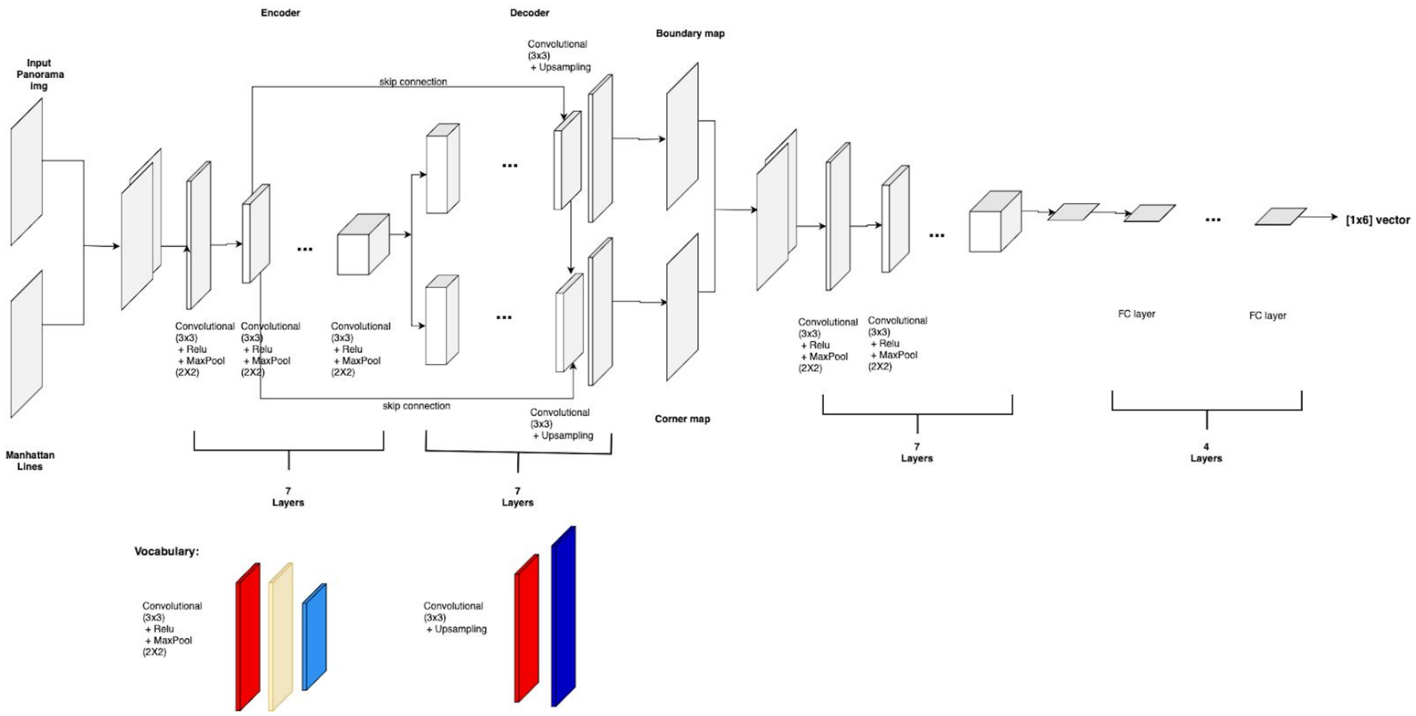
Method	3D IoU (%)	Corner error (%)	Pixel error (%)
PanoContext [33]	67.23	1.60	4.55
ours (corner)	73.16	1.08	4.10
ours (corner+boundary)	73.26	1.07	**3.31**
ours full (corner+boundary+3D)	**74.48**	**1.06**	3.34
ours w/o alignment	69.91	1.44	4.39
ours w/o cuboid constraint	72.56	1.12	3.39
ours w/o layout optimization	73.25	1.08	3.37
ours w/ $L2$ loss	73.55	1.12	3.43
ours full w/ Stnfd. 2D-3D data	75.12	1.02	3.18

- Self-labeled dataset (1413)

Method	3D IoU (%)	Corner error (%)	Pixel error (%)
ours (corner)	72.50	1.27	3.44
ours (corner+boundary)	75.26	1.03	**2.68**
ours full (corner+boundary+3D)	75.39	**1.01**	2.70
ours w/o alignment	68.56	1.56	3.70
ours w/o cuboid constraint	74.13	1.08	2.87
ours w/o layout optimization	74.47	1.07	2.92
ours w/ $L2$ loss	**76.33**	1.04	2.70
ours full w/ PanoContext data	77.51	0.92	2.42

- As a result, they outperform previous solutions by L2 distance.

Task 2. CNN visualization



Task 3. Experiment summary.

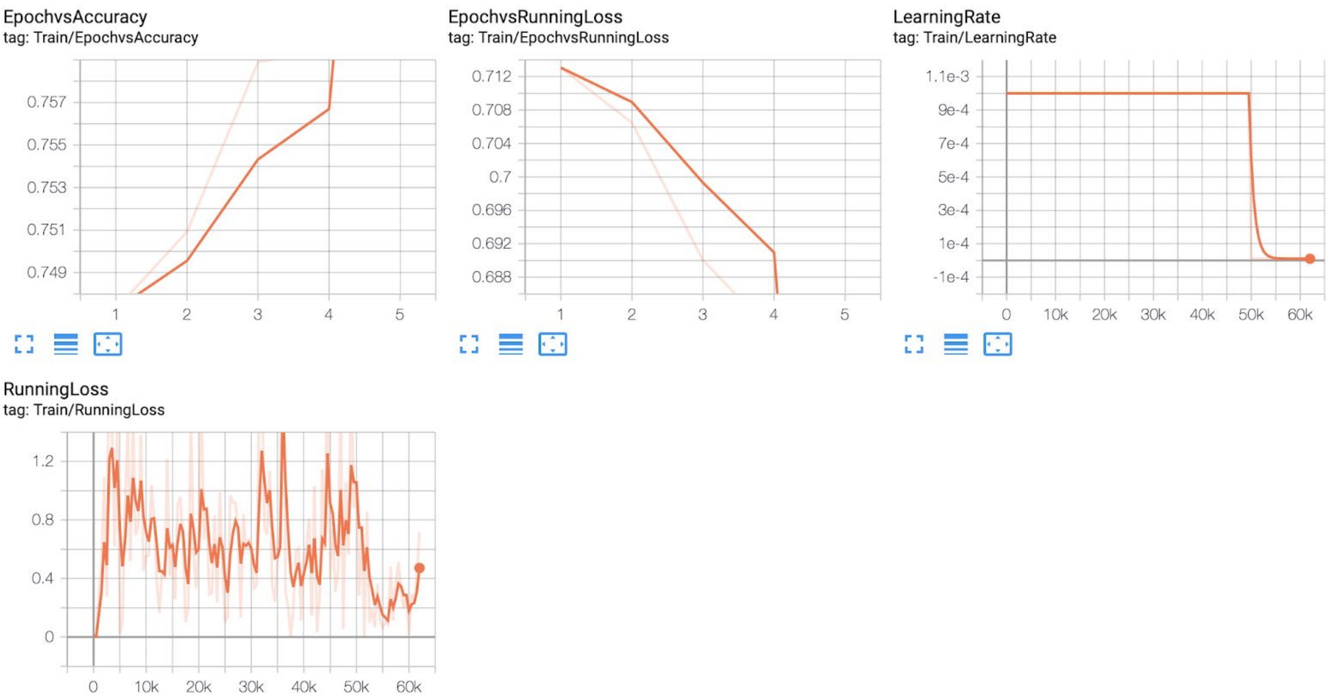
Dataset: cifar10

Out of the box:

architecture:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

plots and metrics:




```
Accuracy of the network on the 10000 test images: 63 %

Accuracy of plane : 72 %
Accuracy of   car : 73 %
Accuracy of  bird : 49 %
Accuracy of   cat : 40 %
Accuracy of  deer : 58 %
Accuracy of   dog : 46 %
Accuracy of  frog : 77 %
Accuracy of horse : 69 %
Accuracy of  ship : 79 %
Accuracy of truck : 67 %
```

What was done:

I was playing with architecture of the Network, and came up with VGG-like architecture.
For the first steps, I added Batch Normalization layer with each Convolution layer, which boosted accuracy on +0.08 (from 0.62 - 0.7).
Then I started stucking new sets of the same layers, and came up with sets of (Conv+BatchNorm+Relu).
Additional layers does not affects accuracy that much, but affected learning time.
Also, changing epochs number from 5 to 7/10 does not affected accuracy.

Running on CPU vs running on GPU:
GPU - 12 min.
CPU - 64 min.

Added *feature_extraction* layers, set of

Conv2d,
BatchNorm2d,
ReLU

And added *mlp_classify* layers.

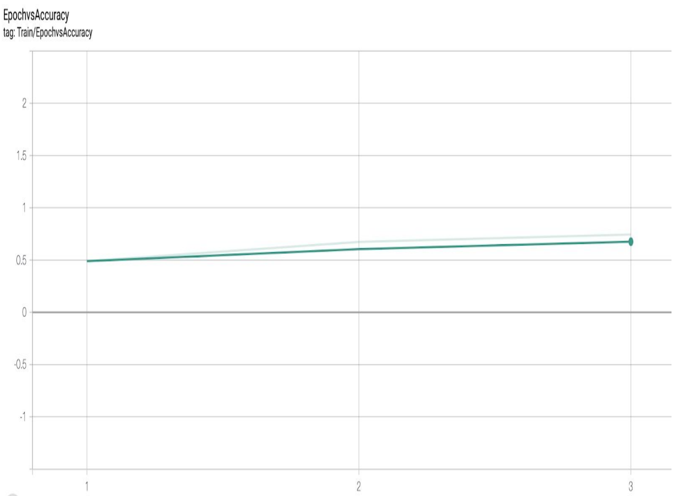
architecture:

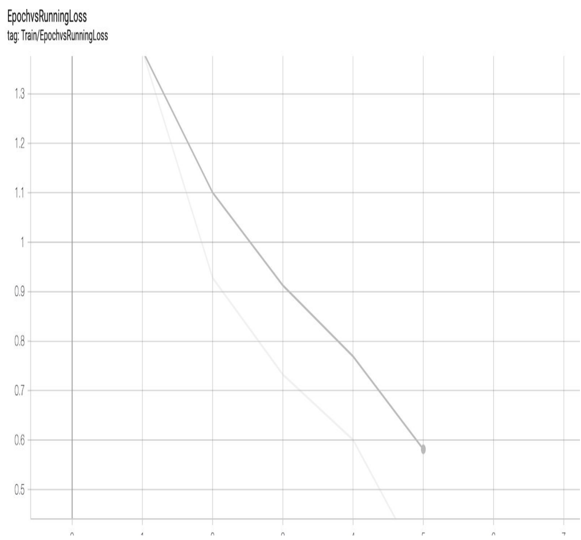
```python
class Net3(nn.Module):

    def __init__(self):
        super(Net3, self).__init__()

        layers = []
        in_channels = 3
        layers_sets = [64, 'pool', 128,
                       'pool', 256, 256,
                       'pool', 512, 512,
                       'pool', 512, 512,
                       'pool']

        for layer_set in layers_sets:

            if layer_set == 'pool':
                layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
            else:
                layers += [nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
                           nn.BatchNorm2d(x),
                           nn.ReLU(inplace=True)]

                in_channels = x

        layers += [nn.AvgPool2d(kernel_size=1, stride=1)]


        self.feature_extraction = nn.Sequential(*layers)

        self.mlp_classify = nn.Sequential(*[
            nn.Linear(512, 120),
            nn.Linear(120, 84),
            nn.Linear(84, 10)
        ])

    def forward(self, x):
        out = self.feature_extraction(x)

        out = out.view(out.size(0), -1)

        out = self.mlp_classify(out)

        return out
```

```
  (feature_extraction): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU(inplace)
    (11): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): ReLU(inplace)
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (16): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (17): ReLU(inplace)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (20): ReLU(inplace)
    (21): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (22): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (23): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (24): ReLU(inplace)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (27): ReLU(inplace)
    (28): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (29): AvgPool2d(kernel_size=1, stride=1, padding=0)
  )
  (mlp_classify): Sequential(
    (0): Linear(in_features=512, out_features=120, bias=True)
    (1): Linear(in_features=120, out_features=84, bias=True)
    (2): Linear(in_features=84, out_features=10, bias=True)
  )
)
```

So, I did VGG-like network.

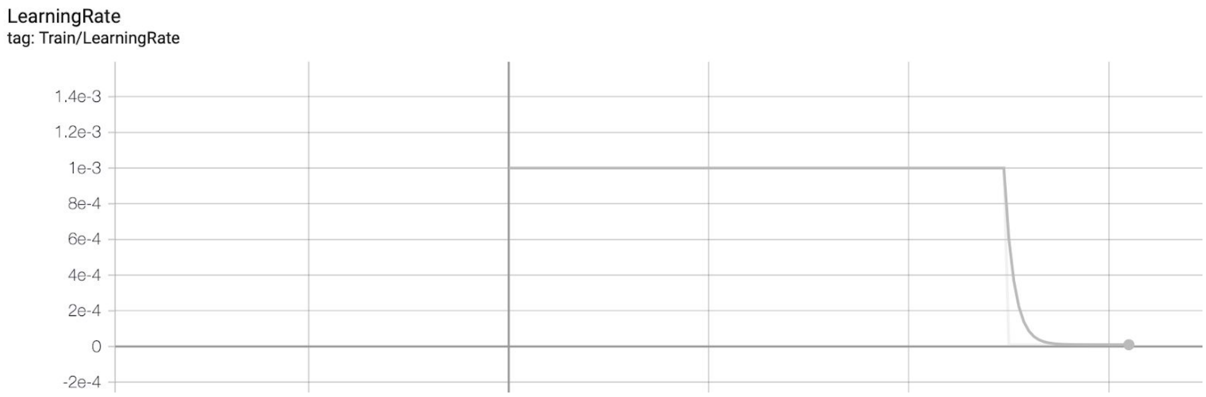
Plots:



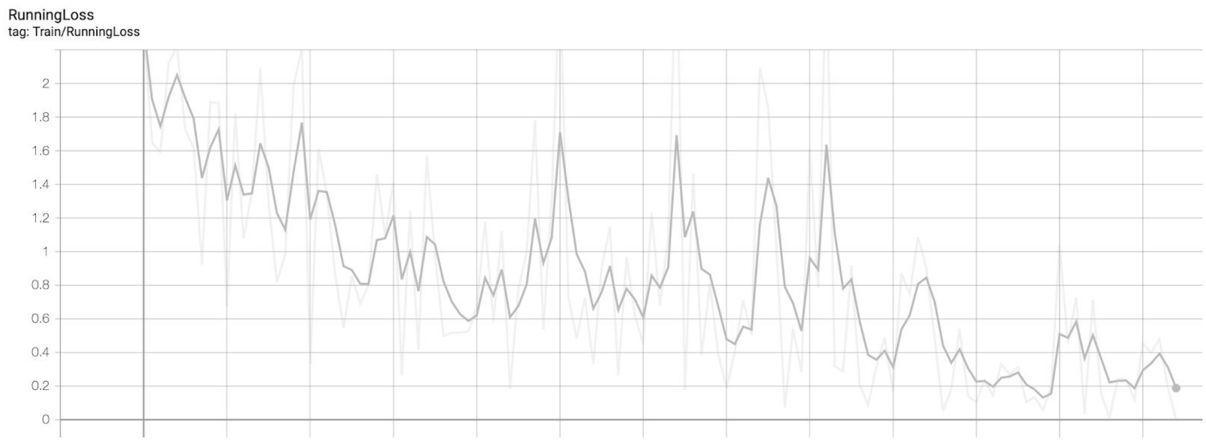


Epoch VS accuracy plot

Epoch VS running loss



Learning rate plot



Loss plot

Result metrics

```
Accuracy of the network on the 10000 test images: 82 %

Accuracy of plane : 86 %
Accuracy of   car : 92 %
Accuracy of  bird : 72 %
Accuracy of   cat : 71 %
Accuracy of  deer : 84 %
Accuracy of   dog : 65 %
Accuracy of  frog : 86 %
Accuracy of horse : 85 %
Accuracy of  ship : 90 %
```

```
Accuracy of truck : 88 %
```

Next steps

As next steps, we can look at different architecture’s elements to add.
Also, we can add dropout, for regularization.
We can try to add skip connection/s after.