Computer Vision

3D Object Detection

GroupReport

Supervised by :

Dr. Marwan Torki

Eng. Aya Lotfy

Prepared by:

Moustafa Tohamy	3851
Ziad Ezzat	4492
Moustafa Sherif Ahmed	4347

Outline

- Introduction
- Kitti dataset
- Frustum ConvNet
- Complex Yolo
- Stereo R-CNN

Introduction

In this report we summarize our efforts ,code description and results in 3 papers , Frustum ConvNet ,Complex Yolo and stereo 3d object detection.

Kitti Dataset

- The 3D object detection benchmark consists of 7481 training images ,7518 test and the corresponding point clouds with a total of 80.256 labeled objects.
- precision-recall curves is computed for evaluation.
    - Train data
        - Left Colored Images
        - Right Colored Images
        - Velodyne point clouds(laser information)
            - X/Z plane (bird's eye view)
        - camera calibration matrices
        - training labels

Frustum ConvNet

- ın this approach we predict the 3d location from cloud points
- The preprocessing
    - Getting the proposed frustums from the 2d regions
    - Using the calibration matrix for each image
    - Saving it to use it in the training process
- Description of some parts of codes
    - Kitti object is a class to represent each sample (image , calibration , lidar, label)
    - Drawing utiliti file responsible for drawing the boxes in the velodyne space and on the 2d images
    - Calibration Class is responsible for transferring the coordinate between image , lidar and camera coordinates
    - Training configuration all set in config file
    - AttrDict class is mapper to the configuration
    - Det_base file is the file of base mode(all the details in the presentation and summary)
        - PointNetModule : Single Scale point Net
        - PointNetFeat:Multi Scale point Net ,wrapper of Point Net Module
        - ConvFeatNet: fully convolutional layer
        - PointNetDet:the whole pipeline
    - Box Transform : is responsible for the transformation of the detected boxes
    - Provide Sampler:is a helper class to detect the Frustums
    - Logger : is responsible for writing the log of each epoch
    - Utils is responsible for calculating accuracy and the metrecs
    -
- We didn’t manage to start the training , we just finished the preprocessing part

```
[ ]  1 extract_frustum_data(
     2     os.path.join(BASE_DIR, 'image_sets/val.txt'),
     3     'training',
     4     os.path.join(save_dir, output_prefix + 'val.pickle'),
     5     perturb_box2d=False, augmentX=1,
     6     type_whitelist=type_whitelist)
```

```
-------------  1
-------------  2
-------------  4
-------------  5
-------------  6
-------------  8
-------------  15
-------------  19
-------------  20
-------------  21
-------------  23
-------------  24
-------------  25
-------------  27
-------------  28
-------------  31
```

Complex Yolo 3d

- In this approach we predict the 3d location from cloud points
- There is no preprocessing
- just download the dataset and start the training
- Code description
    - Kitti dataset class: is a representative class of the dataset paths
    - Complex Yolo Class contains the model
    - Region loss responsible of
        - building the targets from the ground truth
        - Define the loss function
    - Eval file responsible of the testing
- We managed to start the training
- Part of our training log
    - 2020-01-19 21:45:55,320 Running epoch = 0
    - 2020-01-19 21:45:56,519 Running batch_idx = 0
    - 2020-01-19 21:45:56,909 nGT 62, recall 0, proposals 29959, loss: x 6.963236, y 7.106055, w 66.112061, h 37.099342, conf 9072.775391, cls 136.173859, Euler 75.637360, total 9401.867188
    - 2020-01-19 21:45:58,053 Running batch_idx = 1
    - 2020-01-19 21:45:58,302 nGT 32, recall 0, proposals 6752, loss: x 3.171172, y 3.378602, w 29.574814, h 38.185181, conf 1819.289551, cls 65.287659, Euler 42.399132, total 2001.286133
    - 2020-01-19 21:45:59,360 Running batch_idx = 2
    - 2020-01-19 21:45:59,659 nGT 60, recall 2, proposals 4096, loss: x 5.749388, y 5.151956, w 36.724430, h 25.961208, conf 1485.957275, cls 107.765564, Euler 52.645340, total 1719.955200
    - 2020-01-19 21:46:00,687 Running batch_idx = 3
    - 2020-01-19 21:46:00,997 nGT 52, recall 2, proposals 5120, loss: x 5.405334, y 5.918251, w 45.645813, h 34.204739, conf 1647.678223, cls 93.719444, Euler 73.305908, total 1905.877686
    - 2020-01-19 21:46:02,006 Running batch_idx = 4
    - 2020-01-19 21:46:02,321 nGT 62, recall 0, proposals 6144, loss: x 5.369157, y 5.582494, w 34.566509, h 38.062122, conf 1924.147095, cls 114.018974, Euler 77.460197, total 2199.206543
    - 2020-01-19 21:46:03,322 Running batch_idx = 5

- 2020-01-19 21:46:03,629 nGT 58, recall 4, proposals 6144, loss: x 4.726812, y 4.265404, w 40.663322, h 64.689880, conf 2022.828247, cls 92.345230, Euler 62.616238, total 2292.135010

https://github.com/HKUST-Aerial-Robotics/Stereo-RCNN/tree/1.0

Paper 2 Code

We recommend using updated **26 May 2019 branch 1** as Pytorch 1.0.0 and Python 3.6 are supported in that branch

We downloaded kitti dataset

Then downloaded github project files and linked training folder to project folder

```
/Stereo-RCNN-1.0/data/kitti/object
```

We updated some codes related to this bug

> https://discuss.pytorch.org/t/runtimeerror-set-sizes-contiguous-is-not-allowed-on-tensor-created-from-data-or-detach-in-pytorch-1-1-0/44208
>
> By replacing for example Img.data.resise with Img.resize in some py files directly to be compatible with colab pytorch version

And edited test files to save output images and neglected velodyne output images evaluation as colab GPU runtime have limited disk space preventing us from downloading velodyne 26GB along with stereo images

All in all

> we downloaded stereo images as the model needs it only in training along with calibration and data label text files and then evaluated our results.
> The model is very slow so we stopped it @ epoch #3 it saves checkpoints every epoch and supports resuming from a checkpoint
> What we did is we downloaded their network weights which as after 12 epochs and resumed training on it for 3 epochs
> And then got some output results using the provided checkpoint

Trainval_net.py

Here is the code that support resuming by assigning checkpoint # in args i/p useful as the network is pretty slow
Default is `stereo_rcnn_12_6477.pth "epoch 12"`

```
  if args.resume:
    load_name = os.path.join(output_dir,
      'stereo_rcnn_{}_{}.pth'.format(args.checkepoch, args.checkpoint))
    log_string('loading checkpoint %s' % (load_name))
    checkpoint = torch.load(load_name)
    args.start_epoch = checkpoint['epoch']
    stereoRCNN.load_state_dict(checkpoint['model'])
    lr = optimizer.param_groups[0]['lr']
    uncert.data = checkpoint['uncert']
    log_string('loaded checkpoint %s' % (load_name))
.
.
All  im_data.resize_ will be replaced to im_.resize_ for code to work
with current colab pytorch version and new pyttorch versions

for step in range(iters_per_epoch):
      data = next(data_iter)
      im_left_data.data.resize_(data[0].size()).copy_(data[0])
      im_right_data.data.resize_(data[1].size()).copy_(data[1])
      im_info.data.resize_(data[2].size()).copy_(data[2])
      gt_boxes_left.data.resize_(data[3].size()).copy_(data[3])
      gt_boxes_right.data.resize_(data[4].size()).copy_(data[4])
      gt_boxes_merge.data.resize_(data[5].size()).copy_(data[5])
      gt_dim_orien.data.resize_(data[6].size()).copy_(data[6])
      gt_kpts.data.resize_(data[7].size()).copy_(data[7])
      num_boxes.data.resize_(data[8].size()).copy_(data[8])

.. saving after each epoch
   save_name = os.path.join(output_dir,
'stereo_rcnn_{}_{}.pth'.format(epoch, step))
    save_checkpoint({
      'epoch': epoch + 1,
      'model': stereoRCNN.state_dict(),
      'optimizer': optimizer.state_dict(),
      'uncert':uncert.data,
```

```
    }, save_name)

    log_string('save model: {}'.format(save_name))
    end = time.time()
    log_string('time %.4f' %(end - start))
```

/content/Stereo-RCNN-1.0/lib/model/dense_align/box_3d.py

.
.
.

Rotation Matrix having cos(ry),sin(ry) ry is under col #13 in train_label text files where ry is a float # from -pi to +pi

```
  self.R_c_o = torch.FloatTensor
([[ m.cos(poses[6]), 0 ,m.sin(poses[6])],
 [ 0,         1 ,     0],
 [-m.sin(poses[6]), 0 ,m.cos(poses[6])]]).type_as(self.T_c_o)
```

.
.
.

/content/Stereo-RCNN-1.0/lib/model/dense_align/dense_align.py

Here we can see using of 2d,and 3d bbox train and camera calibration data to construct Region of Interest for bbox alignment

```
def sample(calib, scale, f_h, f_w, box_left, poses, borders):
  ''' Return sample pixel for the left image in the valid RoI region
    Inputs:
      box_left: rois x 4          2D box from Train labels
      poses: x, y, z, w, h, l, theta (rois x 7)    3D Train Labels
      f_w, f_h: width and height of the rescaled image
      borders: left and right border of the valid RoI (rois x 2)
    Return:
      all_uvz: sample u locations, sample v locations, delta z w.r.t
the object center
           rois x pixels x 3
      all_weight: we sample same number pixels for all object RoI,
            As a result, 0 denotes unused pixels in all_weight
            1 denotes useful pixels in all_weight. (rois x pixels)

  '''
```

test_net.py

Import of pretrained resnet 101 resnet101_caffe.pth
We downloaded it under /Stereo-RCNN-1.0/data/pretrained_model/

And loading of saved checkpoint stereo_rcnn_12_6477.pth
Under Stereo-RCNN-1.0/models_stereo/

.
.
.
.

```
 # initilize the network here.
  stereoRCNN = resnet(imdb.classes, 101, pretrained=False)
  stereoRCNN.create_architecture()

  print("load checkpoint %s" % (load_name))
  checkpoint = torch.load(load_name)
  stereoRCNN.load_state_dict(checkpoint['model'])
  print('load model successfully!')
```

.
.
.
.

We commented Vendolyne projection lines in this file

/content/Stereo-RCNN-1.0/lib/model/stereo_rcnn/stereo_rcnn.py
Path of Stereo RCNN

.
.
.
.bilinear upsampling

```
 def _upsample_add(self, x, y):
        '''Upsample and add two feature maps.
        Args:
          x: (Variable) top feature map to be upsampled.
          y: (Variable) lateral feature map.
        Returns:
          (Variable) added feature map.
        Note in PyTorch, when input size is odd, the upsampled feature
map
        with `F.upsample(..., scale_factor=2, mode='nearest')`
        maybe not equal to the lateral feature map size.
        e.g.
        original input size: [N,_,15,15] ->
        conv2d feature map size: [N,_,8,8] ->
        upsampled feature map size: [N,_,16,16]
        So we choose bilinear upsample which supports arbitrary output
sizes.
        '''
            _,_,H,W = y.size()
                    return F.interpolate(x, size=(H,W), mode='bilinear',
            align_corners=False) + y
```

.
.

```
  def forward(self, im_left_data, im_right_data, im_info,
gt_boxes_left, gt_boxes_right,\
                gt_boxes_merge, gt_dim_orien, gt_kpts, num_boxes):
        batch_size = im_left_data.size(0)

        im_info = im_info.data
        gt_boxes_left = gt_boxes_left.data
        gt_boxes_right = gt_boxes_right.data
        gt_boxes_merge = gt_boxes_merge.data
        gt_dim_orien = gt_dim_orien.data
        gt_kpts = gt_kpts.data
```

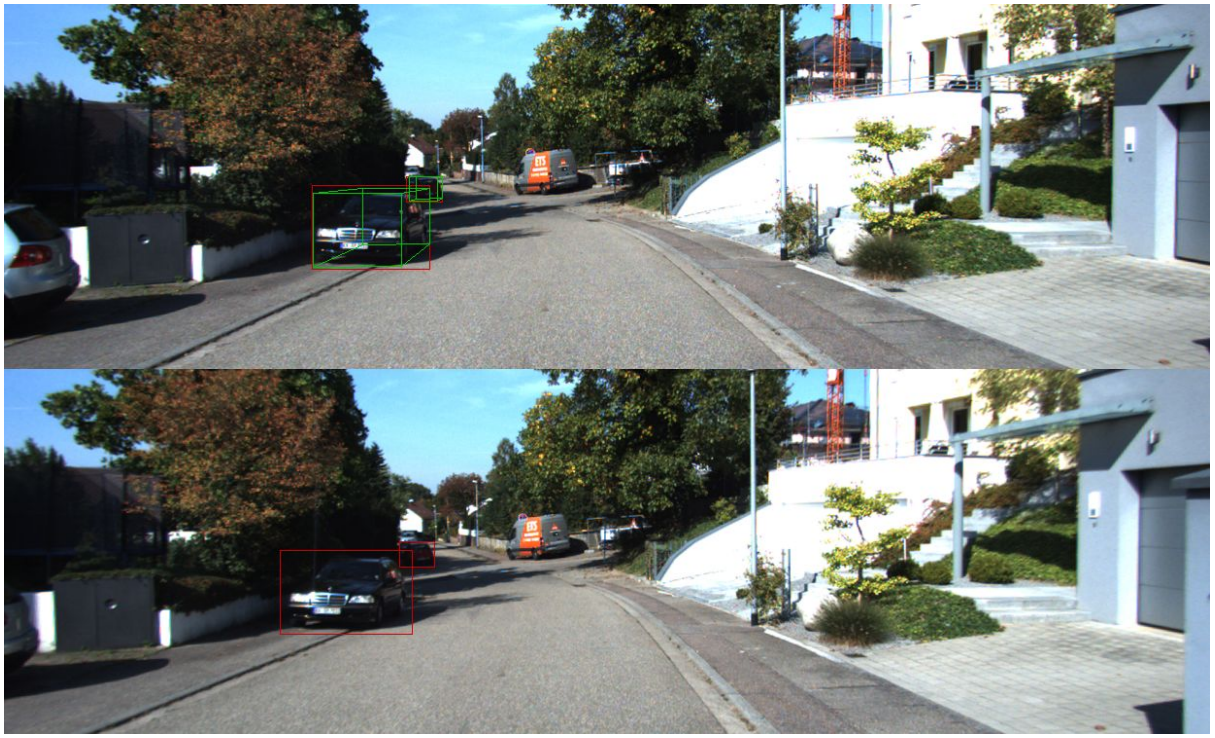
```
        num_boxes = num_boxes.data

        # feed left image data to base model to obtain base feature map
        # Bottom-up
        c1_left = self.RCNN_layer0(im_left_data) # 64 x 1/4
        c2_left = self.RCNN_layer1(c1_left)      # 256 x 1/4
        c3_left = self.RCNN_layer2(c2_left)      # 512 x 1/8
        c4_left = self.RCNN_layer3(c3_left)      # 1024 x 1/16
        c5_left = self.RCNN_layer4(c4_left)      # 2048 x 1/32
        # Top-down
        p5_left = self.RCNN_toplayer(c5_left)    # 256 x 1/32
        p4_left = self._upsample_add(p5_left,
self.RCNN_latlayer1(c4_left))
        p4_left = self.RCNN_smooth1(p4_left)     # 256 x 1/16
        p3_left = self._upsample_add(p4_left,
self.RCNN_latlayer2(c3_left))
        p3_left = self.RCNN_smooth2(p3_left)     # 256 x 1/8
        p2_left = self._upsample_add(p3_left,
self.RCNN_latlayer3(c2_left))
        p2_left = self.RCNN_smooth3(p2_left)     # 256 x 1/4
        p6_left = self.maxpool2d(p5_left)        # 256 x 1/64

        # feed right image data to base model to obtain base feature
map
        # Bottom-up
        c1_right = self.RCNN_layer0(im_right_data)
        c2_right = self.RCNN_layer1(c1_right)
        c3_right = self.RCNN_layer2(c2_right)
        c4_right = self.RCNN_layer3(c3_right)
        c5_right = self.RCNN_layer4(c4_right)
        # Top-down
        p5_right = self.RCNN_toplayer(c5_right)
        p4_right = self._upsample_add(p5_right,
self.RCNN_latlayer1(c4_right))
        p4_right = self.RCNN_smooth1(p4_right)
        p3_right = self._upsample_add(p4_right,
self.RCNN_latlayer2(c3_right))
        p3_right = self.RCNN_smooth2(p3_right)
        p2_right = self._upsample_add(p3_right,
self.RCNN_latlayer3(c2_right))
```

```
        p2_right = self.RCNN_smooth3(p2_right)
        p6_right = self.maxpool2d(p5_right)

        rpn_feature_maps_left = [p2_left, p3_left, p4_left, p5_left,
p6_left]
        mrcnn_feature_maps_left = [p2_left, p3_left, p4_left, p5_left]

        rpn_feature_maps_right = [p2_right, p3_right, p4_right,
p5_right, p6_right]
        mrcnn_feature_maps_right = [p2_right, p3_right, p4_right,
p5_right]
```

Applying RCNN_Region Propasal Network giving stereo rpn_feature_map(both left and right)

```
        rois_left, rois_right, rpn_loss_cls, rpn_loss_bbox_left_right =
\
            self.RCNN_rpn(rpn_feature_maps_left,
rpn_feature_maps_right, \
            im_info, gt_boxes_left, gt_boxes_right, gt_boxes_merge,
num_boxes)
```

results



Stereo RCNN Training Log file

[epoch 1][iter   0/6478] loss: 8.1726, lr: 1.00e-03
fg/bg=(5/507), time cost: 1.408158
rpn_cls: 0.6934, rpn_box_left_right: 0.0039, rcnn_cls: 0.6881, rcnn_box_left_right 0.0152,dim_orien 0.0202, kpts 3.7930
uncert: -0.9991, -1.0010, -0.9991, -1.0010, -1.0009, -0.9907
[epoch 1][iter   1/6478] loss: 5.5532, lr: 1.00e-03
fg/bg=(6/506), time cost: 1.198747
rpn_cls: 0.6951, rpn_box_left_right: 0.0006, rcnn_cls: 0.6856, rcnn_box_left_right 0.0196,dim_orien 0.0234, kpts 2.8504
uncert: -0.9974, -1.0029, -0.9975, -1.0028, -1.0027, -0.9756
[epoch 1][iter   2/6478] loss: 8.1649, lr: 1.00e-03
fg/bg=(6/506), time cost: 1.200525
rpn_cls: 0.6940, rpn_box_left_right: 0.0026, rcnn_cls: 0.6801, rcnn_box_left_right 0.0250,dim_orien 0.0226, kpts 3.8755
uncert: -0.9950, -1.0056, -0.9952, -1.0053, -1.0053, -0.9528
[epoch 1][iter   3/6478] loss: 8.2379, lr: 1.00e-03
fg/bg=(11/501), time cost: 1.186558

.
.

.
.
.
.
.

[epoch 2][iter 6235/6478] loss: -17.2293, lr: 1.00e-03
fg/bg=(36/476), time cost: 1.223495
rpn_cls: 0.0000, rpn_box_left_right: 0.0006, rcnn_cls: 0.0305, rcnn_box_left_right 0.0614,dim_orien 0.0404, kpts 2.1387
uncert: -6.2822, -7.3278, -3.5555, -3.2145, -3.6251, 0.5078
[epoch 2][iter 6236/6478] loss: -17.4608, lr: 1.00e-03
fg/bg=(29/483), time cost: 1.212602
rpn_cls: 0.0028, rpn_box_left_right: 0.0004, rcnn_cls: 0.0394, rcnn_box_left_right 0.0485,dim_orien 0.0101, kpts 1.5109
uncert: -6.2871, -7.3291, -3.5548, -3.2133, -3.6256, 0.5070
[epoch 2][iter 6237/6478] loss: -21.7086, lr: 1.00e-03
fg/bg=(7/505), time cost: 1.202961
rpn_cls: 0.0000, rpn_box_left_right: 0.0001, rcnn_cls: 0.0003, rcnn_box_left_right 0.0070,dim_orien 0.0050, kpts 2.1158
uncert: -6.2925, -7.3312, -3.5552, -3.2130, -3.6268, 0.5065
[epoch 2][iter 6238/6478] loss: -22.1938, lr: 1.00e-03
fg/bg=(9/503), time cost: 1.210482
rpn_cls: 0.0000, rpn_box_left_right: 0.0001, rcnn_cls: 0.0063, rcnn_box_left_right 0.0037,dim_orien 0.0007, kpts 1.3911

References

Frustum ConvNet Repo:
https://github.com/zhixinwang/frustum-convnet
Complex Yolo Repo:
https://github.com/AI-liu/Complex-YOLO