DTV HW2- Detection

Introduction

In this assignment, it's objective is to modify a neural network to do detection. Dataset used is Pascal VOC 2007. The data has been split into 50% for training/validation and 50% for testing. The distributions of images and objects by class are approximately equal across the training/validation and test sets. In total there are 9,963 images, containing 24,640 annotated objects.

Experiment Setup

The neural network used in this assignment is faster-RCNN, using ResNet as its CNN layer. ResNet in this case has 101 layer implementation, edited from torchvision source code and adding additional module to try and improve its performance.

```
class Bottleneck(nn.Module):
  expansion = 4

  def __init__(self, inplanes, planes, stride=1, downsample=None):
    super(Bottleneck, self).__init__()
    self.conv1 = nn.Conv2d(inplanes, planes, kernel_size=1, stride=stride, bias=False) # change
    self.bn1 = nn.BatchNorm2d(planes)
    self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, # change
                           padding=1, bias=False)
    self.bn2 = nn.BatchNorm2d(planes)
    self.conv3 = nn.Conv2d(planes, planes * 4, kernel_size=1, bias=False)
    self.bn3 = nn.BatchNorm2d(planes * 4)
    self.relu = nn.ReLU(inplace=True)

    self.globalAvgPool = nn.AdaptiveAvgPool2d(1)
    self.fc1 = nn.Linear(in_features=planes*4, out_features=round(planes/4))
    self.fc2 = nn.Linear(in_features=round(planes/4), out_features=planes*4)
    self.sigmoid = nn.Sigmoid()

    self.downsample = downsample
    self.stride = stride
```

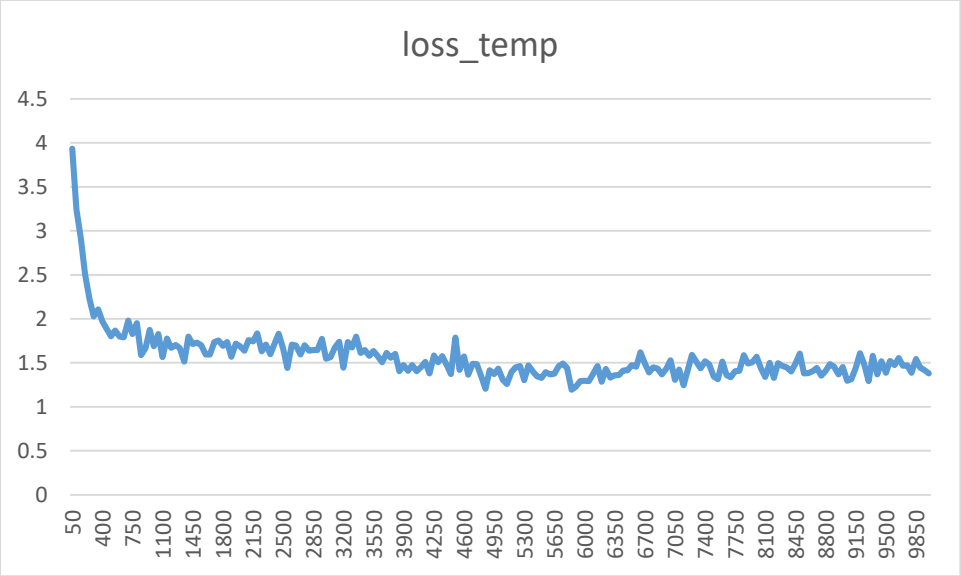
```
def forward(self, x):
  residual = x

  out = self.conv1(x)
  out = self.bn1(out)
  out = self.relu(out)

  out = self.conv2(out)
  out = self.bn2(out)
  out = self.relu(out)

  out = self.conv3(out)
  out = self.bn3(out)

  if self.downsample is not None:
    residual = self.downsample(x)

  original_out = out
  out = self.globalAvgPool(out)
  out = out.view(out.size(0), -1)
  out = self.fc1(out)
  out = self.relu(out)
  out = self.fc2(out)
  out = self.sigmoid(out)
  out = out.view(out.size(0), out.size(1), 1, 1)
  out = out * original_out

  out += residual
  out = self.relu(out)
  return out
```

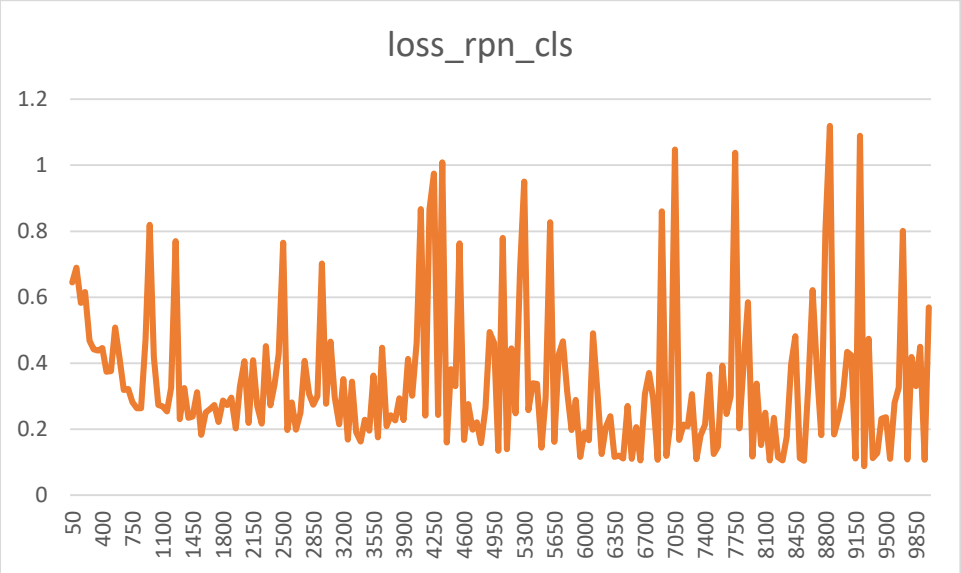
Result

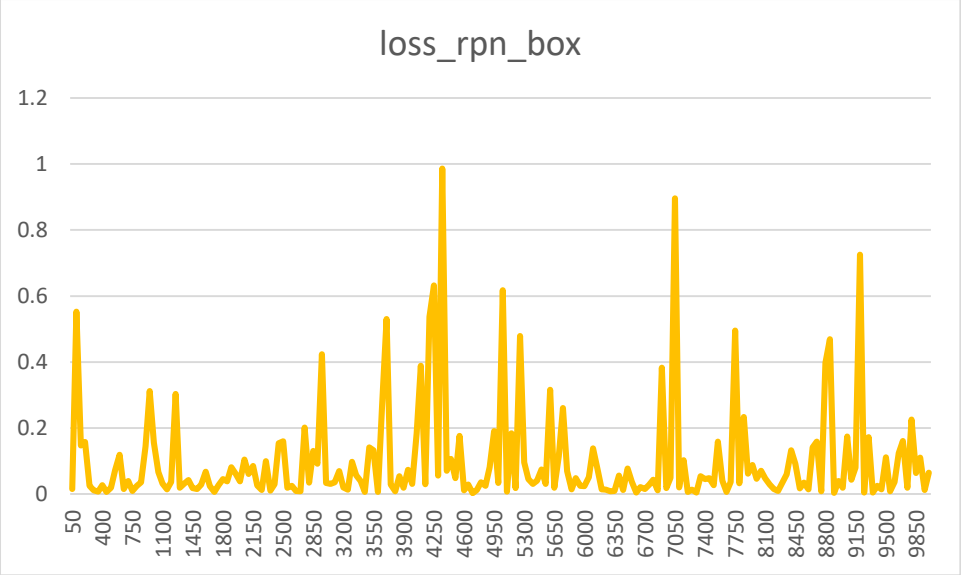
Graph For Reference result (only for 1 epoch due to long training time)

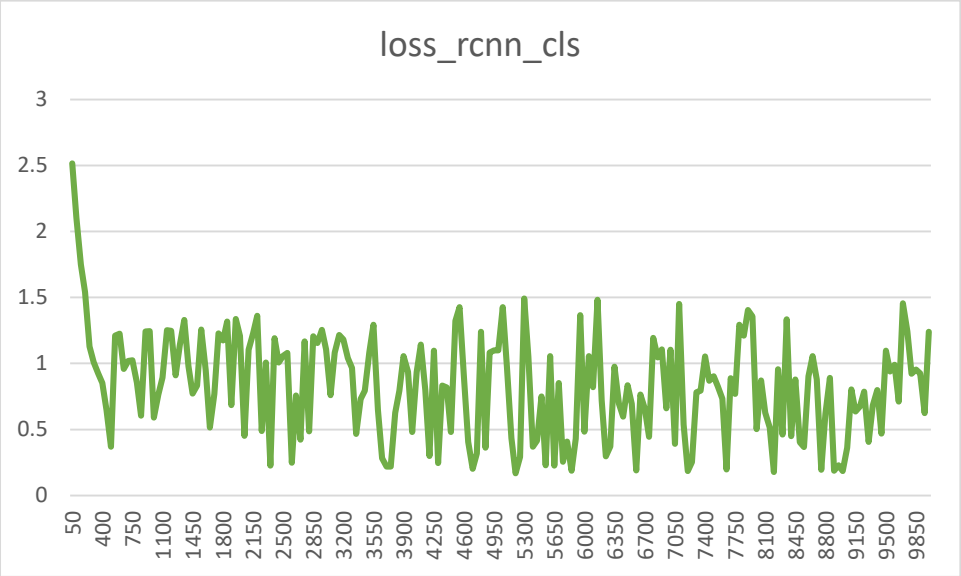
In the original github source code, they use pretrained model to speed up the process, however, due to the change in ResNet layer that will be implemented later, this process is skipped to obtain fair comparison.

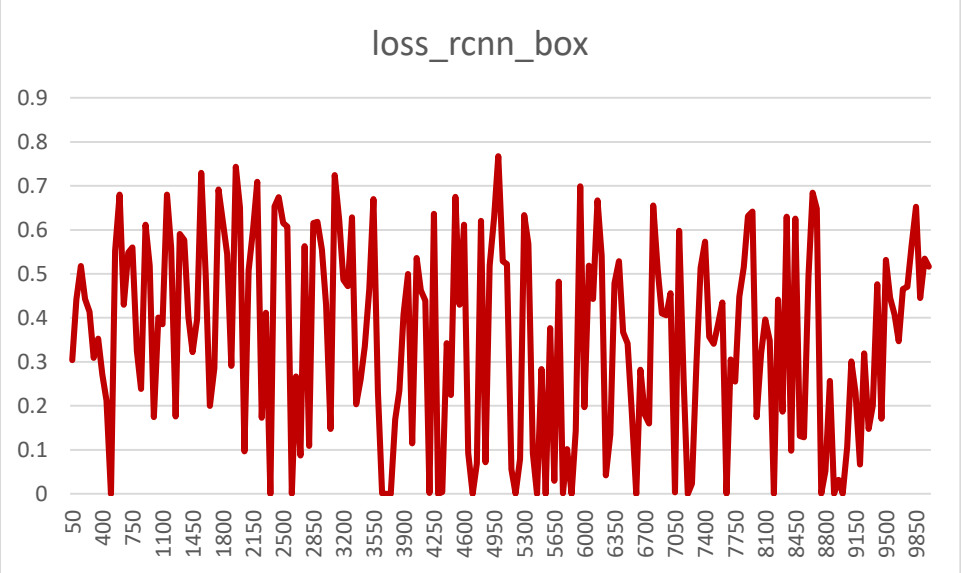
Mean AP = 0.0032





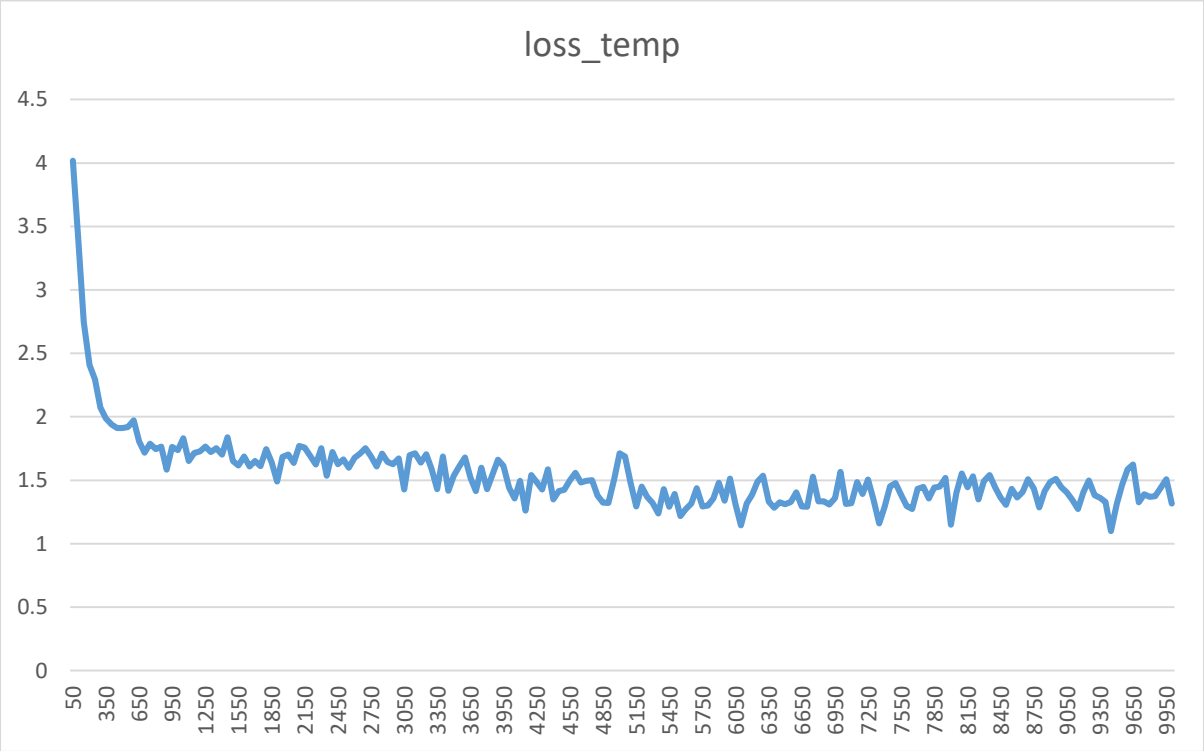


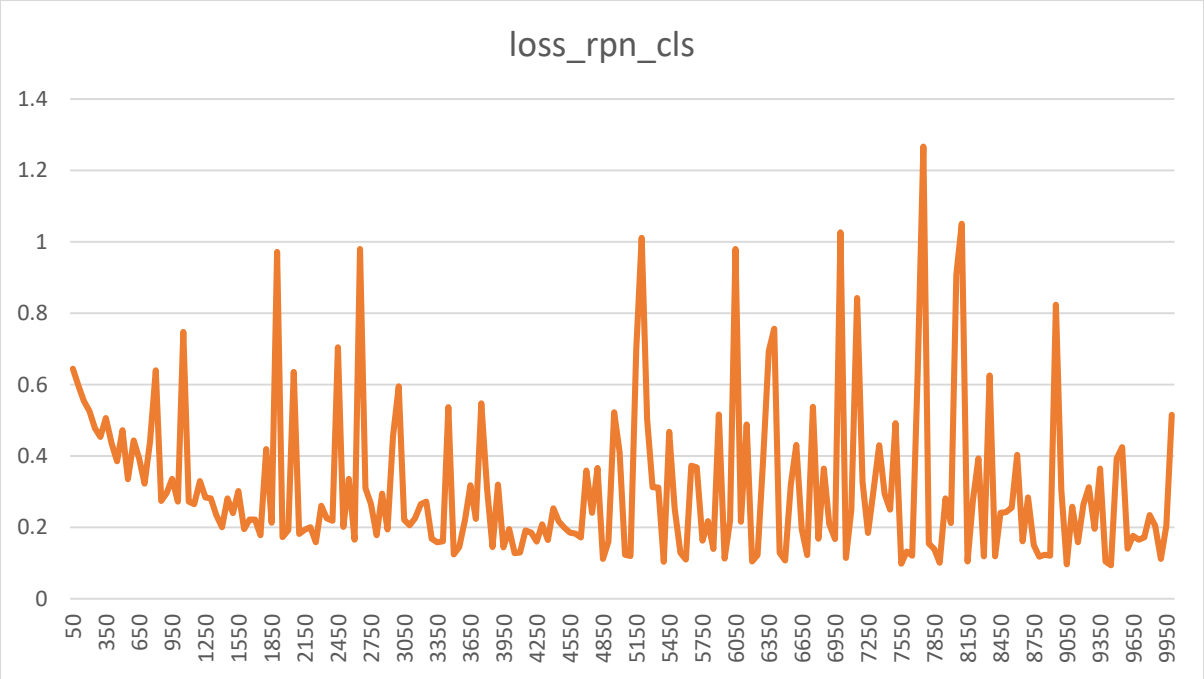


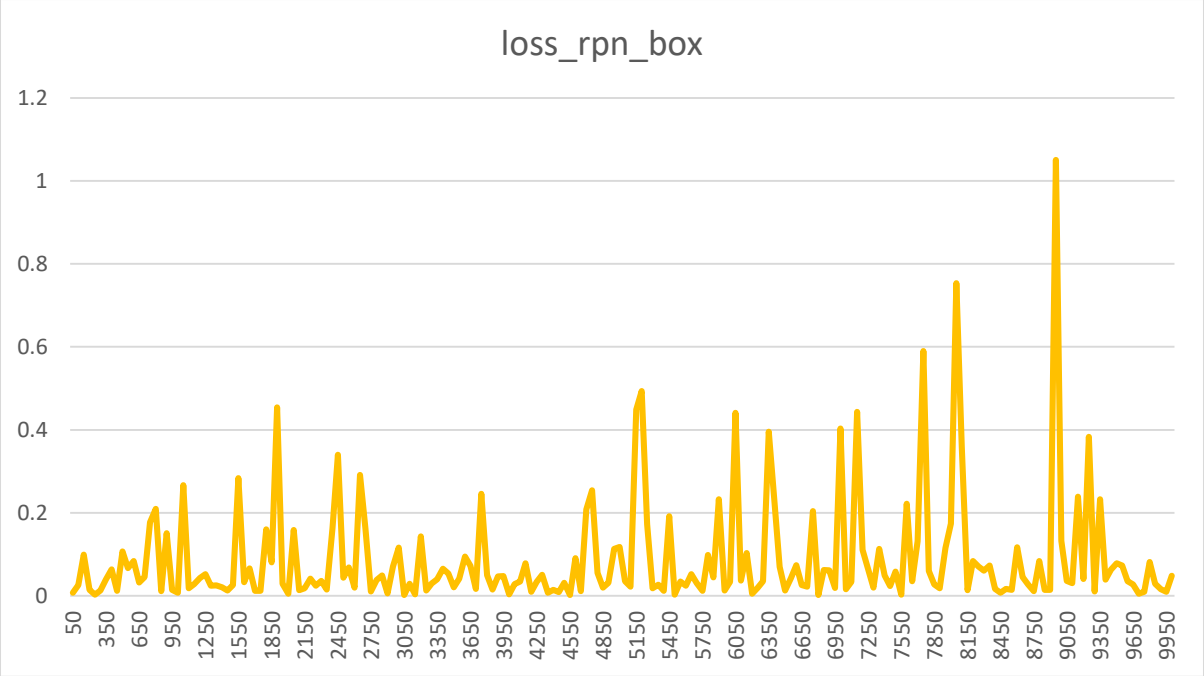


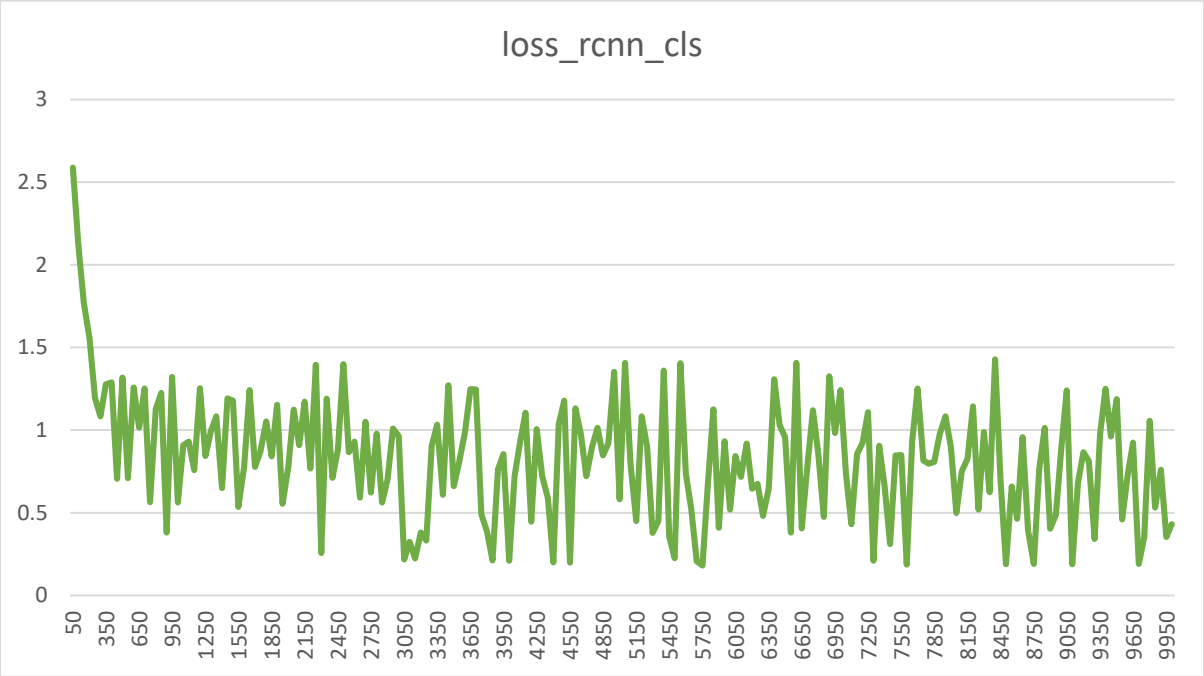
Graph For Modified result (only for 1 epoch due to long training time)

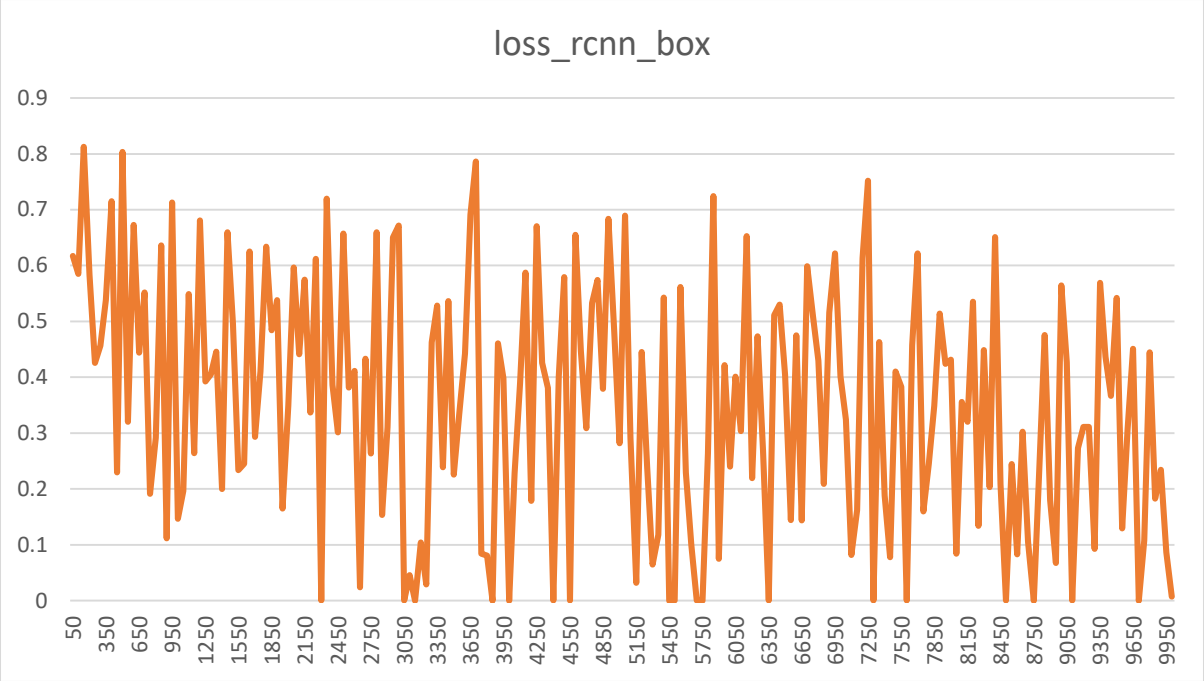
Mean AP = 0.0043






loss_rpn_box
1.2
1
0.8
0.6
0.4
0.2
0
50
350
650
950
1250
1550
1850
2150
2450
2750
3050
3350
3650
3950
4250
4550
4850
5150
5450
5750
6050
6350
6650
6950
7250
7550
7850
8150
8450
8750
9050
9350
9650
9950


loss_rcnn_cls
3
2.5
2
1.5
1
0.5
0
50
350
650
950
1250
1550
1850
2150
2450
2750
3050
3350
3650
3950
4250
4550
4850
5150
5450
5750
6050
6350
6650
6950
7250
7550
7850
8150
8450
8750
9050
9350
9650
9950



Discussions

- Modifying ResNet directly affect the dimension of data when batch of data is considered
- Performance of modified result is better, however, we sacrified slower testing time with not huge impact on training time.
- Due to long time in training and testing time, there is insufficient time to run the data for this report
- Modified codes differ mainly in faster-rcnn.pytorch/lib/model/faster_rcnn/resnet.py compared to reference

Performance

Loss function converges to zero much faster and with better mAP after only 1 epoch.