Artificial Intelligence Programming
Final Report

Department: Dept of Computer Science

Student ID: 2016310526

Student Name: Kim Chanho

0. Abstract……………………………………….……..……..…………..3
1. Data Preprocessing…………………………….……...………………….……4
2. Loss……………………….…………………………..…..……………….5
3. Model…………………………………………….…………….…..…..…..8
4. **Train**…………………..…………………………….…….…………..…..9
5. **Test**……………………………………….………………….…….…..……12
6. **Limitation of Project**…………………………………..……..……….13

0. Abstract

I worked on this project by referring to the Yolo v1 model. Considering the performance of the GPU, I used Pytorch pretrained Resnet50 as a Backbone network, judging that learning the model from scratch would not produce meaningful results.

Resnet was designed to solve the problem of Degradation, a phenomenon in which performance decreases as Layer deeps, and was proved to has good performance in object detection. So I decided to use this model for this project.

The project will be running on Linux(Ubuntu 18.04), but I have only a Computer using Windows OS. So I can’t make same environment. Instead, I tried to use WSL2 as similar as possible, and I will describe the environment used for this project in README.md

1. Data Preprocessing

Before I start this project. I need some preprocessing of data. Given Data was PASCAL VOC 2012 trainval dataset with JPEGImages and Annotation with xml file. So I need to convert annotation to txt file which has Image name and info about bndbox which has xmin, ymin, xmax, ymax in each line. So I just read all xml files and using **xml.etree.ElementTree** package, I parse the info and write in txt file.

Finally txt file has

[Image name, Class_idx_1, xmin_1, ymin_1, xmax_1, ymax_1, Class_idx_2 …] in each line.

And I do train, valid, test split. I think shuffle is not matter because there is no correlation between image sequences. So I just counting each loop and split data 8:1:1 (train:valid:test).

Train: 13313 data

Valid: 2140 data

Test: 1664 data

Total: 17117 data

Although I thought the Valid data was too small compared to the Train data, I wanted to train with a little more data because it trains 448x448 images, and I proceeded in that direction.

VOCDATASET class read the dataset with txt files.

This class parse data total_len – 1(image_name)/5(xmin, ymin, xmax, ymax, class_idx) = num of boxes and Finally total box convert to torch.tensor

Four augmentations were done for train data: random_flip, random_scale, random_shift, random_crop.

I also proceed with normalize (123, 117, 104) with the appropriate mean values.

2. Loss

At first Yolo Loss algorithm is as follows.

Localization loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Confidence loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

<Figure 1 Loss function of Yolo Algorithm>

First, I define loss function as class, and I receive the size of grid S and the number of predicted bounding boxes B by grid cell in constructors. And also received noobj_lambda, coord_lambda in constructors

And in forward function,

First, predict 2 bounding boxes for each grid cell, and use 1 bounding box with a high confidence score for learning.

Pred parameter is tensor which shape is (batch_size, S, S, 30). Each grid(7x7) has 30 values, such as [c1,c2,c3,…,c20,pred_c1,x,y,w,h,pred_c2,x,y,w,h].

The target parameter is also a tensor of (batch_size, 7x7x30).

First, specify coo_mask and noo_mask, respectively. The coo_mask has a positive value for the last channels of the target_tensor, and if it is 0, it is added to the noo_mask.

Apply coo_mask to pred, target respectively. This will used to obtain the Localization Loss of <Figure1> Also apply noo_mask to pred, target to obtain noobj loss using noo_prediction_class and noo_target_class.

In the model Yolo v1, predict two bounding boxes, as described above. Compare this two box with IoU with target_class. And it stores index of box with the large IoU value. Use this box as the final prediction against the actual target.

```
# choose the best iou box
for i in range(0, box_target.size()[0], 2):
    box1 = box_pred[i:i + 2]
    box1_xyxy = Variable(torch.FloatTensor(box1.size()))
    box1_xyxy[:, :2] = box1[:, :2] / 14. - 0.5 * box1[:, 2:4]
    box1_xyxy[:, 2:4] = box1[:, :2] / 14. + 0.5 * box1[:, 2:4]

    box2 = box_target[i].view(-1, 5)
    box2_xyxy = Variable(torch.FloatTensor(box2.size()))
    box2_xyxy[:, :2] = box2[:, :2] / 14. - 0.5 * box2[:, 2:4]
    box2_xyxy[:, 2:4] = box2[:, :2] / 14. + 0.5 * box2[:, 2:4]

    iou = self.compute_iou(box1_xyxy[:, :4], box2_xyxy[:, :4])
    max_iou, max_index = iou.max(0)
    max_index = max_index.data.cuda()

    coo_response_mask[i + max_index] = 1
    coo_not_response_mask[i + 1 - max_index] = 1

    box_target_iou[i + max_index, torch.LongTensor([4]).cuda()] = (max_iou).data.cuda()
```

<Figure 2 Compute IoU with two box and target>

I also added container_loss and not container loss to determine whether the object exists or not. I added all loss and used the value divided by batch_size as the Loss value.

In the iou calculation, two boxes of [N,4] and [M,4] tensors were calculated as input, respectively. After converting to [N,M,2] tensor, we subtracted and replaced the negative value with zero. -> intersection.

Determine the area for each box using the method (box1[:,2]-box1[:,0]) * (box1[:,3] – box1[:,1]).

The intersection/(area1 + area2 - intersection) value was used as the iou value.

3. Model

And next, make resnet_based Yolo model.

I download pretrained resnet 50 in pytorch.org so backbone network is almost same with resnet 50.

So I explained detect layer which is last layer of models.

After resnet based model layers, we need to make detect layer which has in_channels=2048, so First I downsample it and add two more layer with 256 to 256 Conv layers.

And then, make Conv layer which has in_channels=256 and out_channels=30 because Yolo detection make output like (S, S, 30) and finally I add BatchNorm2d(30)

Models structure looks like this

```
class ResNet(nn.Module):

    def __init__(self, block, layers, num_classes=1470):
        self.inplanes = 64
        super(ResNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self.create_conv_layers(block, 64, layers[0])
        self.layer2 = self.create_conv_layers(block, 128, layers[1], stride=2)
        self.layer3 = self.create_conv_layers(block, 256, layers[2], stride=2)
        self.layer4 = self.create_conv_layers(block, 512, layers[3], stride=2)

        self.layer5 = self.create_detect_layer(in_channels=2048)

        self.conv_end = nn.Conv2d(256, 30, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn_end = nn.BatchNorm2d(30)
```

<Figure 3 Model Structure>

4. Train

Then I explain parameters used in train, and how to training.

```
file_root = './data'
learning_rate = 0.001
num_epochs = 50
batch_size = 4
net = resnet50()

print('load pre-trined model')
resnet = models.resnet50(pretrained=True)
new_state_dict = resnet.state_dict()
dd = net.state_dict()
for k in new_state_dict.keys():
    if k in dd.keys() and not k.startswith('fc'):
        dd[k] = new_state_dict[k]
net.load_state_dict(dd)


device = 'cuda' if torch.cuda.is_available() else 'cpu'


criterion = Loss_yolobased(7, 2, 5, 0.5, device=device)
```

<Figure 4 Train config>

I train total 50 epochs. (1 epoch take 15m in my GPU)

I use Initial learning rate 0.001 and when epochs = 30 or 40 I multiply 0.1 to learning rate which makes converge well.

And use criterion with my Loss class which I explained in 2. LOSS and load pretrained resnet 50 without fully-connected layers which will be replaced by detection layers.

And set device = ‘cuda’

Batch_size = 4. In yolo paper batch size was 64, but It makes my cuda run out of memory. So I use 4 Batch_size.

```
params=[]
params_dict = dict(net.named_parameters())
for key,value in params_dict.items():
    if key.startswith('features'):
        params += [{'params':[value],'lr':learning_rate*1}]
    else:
        params += [{'params':[value],'lr':learning_rate}]
optimizer = torch.optim.SGD(params, lr=learning_rate, momentum=0.9, weight_decay=5e-4)
# optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate, weight_decay=1e-4)

train_dataset = VOCDATASET(root=file_root, input_file='2016310526_VOC_TRAIN_DATA.txt', t
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

test_dataset = VOCDATASET(root=file_root, input_file='2016310526_VOC_VALID_DATA.txt', tr
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

scaler = torch.cuda.amp.GradScaler()
```

<Figure 5 Train config>

And I use optimizer SGD. I don’t know actually why, but In this case, SGD works much better than Adam.

And load train, test(valid) dataset which I split in data preprocessing

And I use torch.cuda.amp.GradScaler which makes optimization in training process. It makes Learning speeds up because of optimization of batch_size and model.

And then, I test with validation data in every epoch.

```
load pre-trined model
data init
data init
100%|██████████| 3329/3329 [13:39<00:00,  4.06it/s, EPOCH=[1/50], current_loss=2.5617, total_loss=5.2845]
100%|██████████| 535/535 [00:54<00:00,  9.87it/s, loss=4.7286]
get best test loss 4.72863
100%|██████████| 3329/3329 [14:00<00:00,  3.96it/s, EPOCH=[2/50], current_loss=2.6537, total_loss=4.3275]
100%|██████████| 535/535 [00:55<00:00,  9.56it/s, loss=4.4227]
get best test loss 4.42267
100%|██████████| 3329/3329 [13:59<00:00,  3.97it/s, EPOCH=[3/50], current_loss=2.8901, total_loss=4.0688]
100%|██████████| 535/535 [00:54<00:00,  9.75it/s, loss=4.2213]
get best test loss 4.22130
100%|██████████| 3329/3329 [13:50<00:00,  4.01it/s, EPOCH=[4/50], current_loss=1.8252, total_loss=3.8980]
100%|██████████| 535/535 [00:55<00:00,  9.58it/s, loss=4.0181]
get best test loss 4.01810
```

<Figure 6 Train result>

I start with test loss 4.72863

```
100%|██████████| 535/535 [00:54<00:00,  9.90it/s, loss=2.9479]
get best test loss 2.94793
100%|██████████| 3329/3329 [13:04<00:00,  4.25it/s, EPOCH=[35/50], current_loss=1.5782, total_loss=2.5754]
100%|██████████| 535/535 [00:51<00:00, 10.35it/s, loss=3.0237]
100%|██████████| 3329/3329 [13:05<00:00,  4.24it/s, EPOCH=[36/50], current_loss=2.8266, total_loss=2.5905]
100%|██████████| 535/535 [00:51<00:00, 10.35it/s, loss=2.9914]
100%|██████████| 3329/3329 [13:04<00:00,  4.25it/s, EPOCH=[37/50], current_loss=10.1086, total_loss=2.5851]
100%|██████████| 535/535 [00:53<00:00, 10.04it/s, loss=2.9714]
100%|██████████| 3329/3329 [13:04<00:00,  4.24it/s, EPOCH=[38/50], current_loss=4.1304, total_loss=2.5741]
100%|██████████| 535/535 [00:51<00:00, 10.31it/s, loss=3.0166]
100%|██████████| 3329/3329 [13:05<00:00,  4.24it/s, EPOCH=[39/50], current_loss=2.7426, total_loss=2.5560]
100%|██████████| 535/535 [00:53<00:00, 10.07it/s, loss=2.9550]
100%|██████████| 3329/3329 [13:12<00:00,  4.20it/s, EPOCH=[40/50], current_loss=2.1854, total_loss=2.5616]
100%|██████████| 535/535 [00:51<00:00, 10.37it/s, loss=3.0099]
100%|██████████| 3329/3329 [12:57<00:00,  4.28it/s, EPOCH=[41/50], current_loss=5.3822, total_loss=2.5537]
100%|██████████| 535/535 [00:51<00:00, 10.31it/s, loss=3.0249]
100%|██████████| 3329/3329 [13:00<00:00,  4.26it/s, EPOCH=[42/50], current_loss=4.5489, total_loss=2.5367]
100%|██████████| 535/535 [00:53<00:00,  9.97it/s, loss=2.9893]
100%|██████████| 3329/3329 [13:02<00:00,  4.25it/s, EPOCH=[43/50], current_loss=3.6916, total_loss=2.5295]
100%|██████████| 535/535 [00:53<00:00,  9.99it/s, loss=2.9922]
100%|██████████| 3329/3329 [13:54<00:00,  3.99it/s, EPOCH=[44/50], current_loss=4.1627, total_loss=2.5254]
100%|██████████| 535/535 [00:56<00:00,  9.45it/s, loss=3.0126]
100%|██████████| 3329/3329 [14:22<00:00,  3.86it/s, EPOCH=[45/50], current_loss=0.6544, total_loss=2.5145]
100%|██████████| 535/535 [00:56<00:00,  9.39it/s, loss=3.0750]
100%|██████████| 3329/3329 [13:59<00:00,  3.96it/s, EPOCH=[46/50], current_loss=3.1703, total_loss=2.4934]
100%|██████████| 535/535 [00:55<00:00,  9.62it/s, loss=2.9582]
100%|██████████| 3329/3329 [13:44<00:00,  4.04it/s, EPOCH=[47/50], current_loss=1.9823, total_loss=2.4930]
100%|██████████| 535/535 [00:53<00:00,  9.94it/s, loss=3.0086]
100%|██████████| 3329/3329 [13:36<00:00,  4.08it/s, EPOCH=[48/50], current_loss=11.6023, total_loss=2.4999]
100%|██████████| 535/535 [00:53<00:00, 10.06it/s, loss=3.0736]
100%|██████████| 3329/3329 [13:13<00:00,  4.20it/s, EPOCH=[49/50], current_loss=1.4819, total_loss=2.4889]
100%|██████████| 535/535 [00:52<00:00, 10.13it/s, loss=3.0216]
100%|██████████| 3329/3329 [13:20<00:00,  4.16it/s, EPOCH=[50/50], current_loss=0.5246, total_loss=2.4541]
100%|██████████| 535/535 [00:56<00:00,  9.46it/s, loss=2.9921]
```

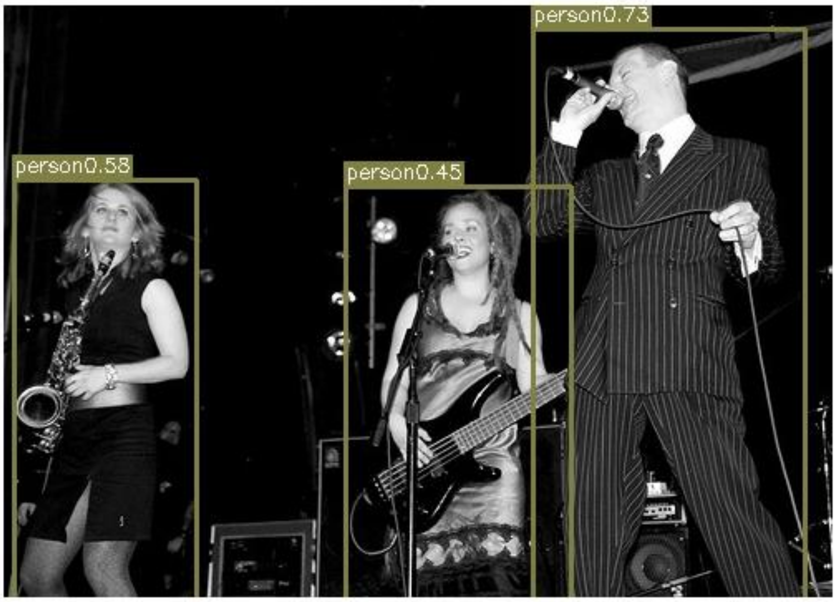
<Figure 7 Train result>

And last best validation loss is 2.94793 in epoch 34.

5. Test

Finally I test with test data I splitted in data preprocessing.

I put Image in trained model in 4.Train and do some simple non maximum suppression with threshold = 0.5.



<Figure 8 Test result1>

<Figure 9 Test result2>

6. Limitation of project

1) Basically PascalVOC dataset is about 20 classes. So It could not classify many classes.

2) And because of GPU memory, I cannot train with batch size more than 4. Maybe If I trained with 64 batch size, the accuracy will be better.

3) Yolo v1 model predict only one class in one cell which means It cannot predict when objects overlapped.

4) It trained based bounding box of training model. So It is hard to new or unique bounding box of test data.

5) And last, small bounding boxes affect more to training IOU which means Localization is very difficult.