Final Project Report

Student ID : 2016312160

Name: 이준성

1. Preprocessing of Data

Parse_Annotations.py

Yolo v1 model를 base로 참고하였습니다. 그래서 데이터 전처리가 필수였고 Annotation의 xml 파일들을 txt 파일로 변환시켰습니다. Xml.etree.ElementTree 라이브러리를 사용하여 annotation안에있는 object를 순환하면서

[그림1, 클라스인덱스1, xmin1, ymin1, xmax1, ymax1, 클라스2,....]

이런 식으로 각 줄에 각각 그림에 대한 object들의 정보가 나열되게 변환하였습니다.

Train, valid, test 데이터의 비율을 8:1:1로 나누어 각각 txt파일을 따로 만들었습니다.

Train: 13691 data

Valid: 1713 data

Test: 1713 data

Divider_Images.py

불필요한 파일이지만 gpu 메모리가 부족하여 test라도 따로 분리하여 사용하기위해 JPEGImages 파일을 train, valid, test 이미지로 분리하는 파일입니다.

Dataset.py

txt파일의 데이터를 읽어 각각의 그림마다 박스들을 encoder를 통해 각각의 정보가 담겨있는 tensor로 변환 후 random_flip, random_scale, random_shift, random_crop의 네가지 augmentation을 적용합니다. 또한 subMean함수를 통해 normalize를 진행하였습니다.

2. Loss and Model

Loss.py

Yolo Loss함수를 베이스로 사용하였습니다.

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

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

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

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

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

그림 1 Loss function Algorithm of YOLO

Grid의 크기(S) 와 bounding box의 예측 수(B) 값을 매개변수로 받았습니다.

Forwarding 함수에서 각각의 grid 마다 2개의 bounding box를 우선 예측

하고 1개의 bounding box를 high confidence 값으로 이용하였습니다.

Pred 변수, Target변수 모두 grid(7x7) x 30의 크기로 되어있습니다.

Object 포함,응답 여부에 따라 coo_mask, noo_mask, coo_response_mask, coo_not response_mask를 각각 tensor로 정의합니다.

각각의 mask를 pred, target에 적용해 loss를 계산합니다.

Object가 포함된 경우의 loss를 계산할 때, 두개의 box를 예측하고 iou값을 계산한다음 큰 값을 저장한후 실제 target과 비교할 때 사용합니다.

Model.py

```
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._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2)
        self.layer5 = self._make_detnet_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)
```

Resnet 을 기반으로 한 yolo model를 사용했습니다.

Pretrained 된 resnet50을 사용하였고 기반으로하였습니다.

Resnet layer이후에는 input channels가 2048인 detect layer를 설정하였습니다. 먼저이를 다운샘플링하고 256 256 convolution layer 두 계층을 추가하였습니다.

그런 다음, yolo가 (x,x,30)과 같은 출력을 만들기 때문에, 256이 input channel 30이 output channel인 convolution layer를 만들고 마지막으로 BatchNorm2d (30)을 추가했습니다.

3. Train and Test

Train

```
learning_rate = 0.001
num_epochs = 40
batch_size = 2
net = resnet50()
resnet = models.resnet50(pretrained=True)
resnet_dic = resnet.state_dict()
dic = net.state_dict()
for k in resnet_dic.keys():
    if k in dic.keys() and not k.startswith('fc'):
        dic[k] = resnet_dic[k]
net.load_state_dict(dic)
```

Total 40 epoch을 돌렸습니다. Learning rate은 초기값을 0.001로하고 epoch이 30이 넘어가면 0.0001값으로 설정하였습니다.

배치사이즈는 많은 숫자를 원하였지만 컴퓨터 사양이 매우 부족하여 gpu memory 부족현상이 나타나 최대한으로 돌릴 수 있는 사이즈가 2였습니다. 이것이 결과적으로 정확성이 떨어진 계기가 된 것 같습니다.

그 다음 resnet 50을 불러와 fully connected layer를 제거하고 가져옵니다. Model에 직접 설정한 layer를 사용하기 위해서입니다.

```
optimizer = torch.optim.SGD(params, lr=learning_rate, momentum=0.9, weight_decay=5e-4)
criterion = lossFunction(7,2,5,0.5)
train_dataset = Dataset(root='./VOC2012/TrainImages/',data_file='./voc2012_train.txt',train=True,transform = [transforms.ToTensor()] )
train_loader = DataLoader(train_dataset,batch_size=batch_size,shuffle=True,num_workers=0)
valid_dataset = Dataset(root='./VOC2012/ValidImages/',data_file='./voc2012_valid.txt',train=False,transform = [transforms.ToTensor()] )
valid_loader = DataLoader(valid_dataset,batch_size=batch_size,shuffle=False,num_workers=0)
```

그 다음 optimizer는 SGD를 사용하였고 criterion은 지정한 yolo loss함수를 사용하였습니다.

그 다음 train valid dataset, loader를 각각 만들었습니다.

그 다음 train을 돌리고 매 애폭마다 valid test를 통해 loss를 갱신할 때마다 model를 생성 업데이트 해주었습니다.

```
Starting epoch 22 / 40
Learning Rate for this epoch: 0.001
Epoch [22/40], Iter [1000/6846] Loss: 3.3352, average_loss: 3.2668
Epoch [22/40], Iter [2000/6846] Loss: 7.4578, average_loss: 3.2817
Epoch [22/40], Iter [3000/6846] Loss: 1.7975, average_loss: 3.2464
Epoch [22/40], Iter [4000/6846] Loss: 4.9072, average_loss: 3.2385
Epoch [22/40], Iter [5000/6846] Loss: 3.1069, average_loss: 3.2338
Epoch [22/40], Iter [6000/6846] Loss: 1.3587, average_loss: 3.2368
Current best test loss 3.54270


Starting epoch 23 / 40
Learning Rate for this epoch: 0.001
Epoch [23/40], Iter [1000/6846] Loss: 2.6660, average_loss: 3.3255
Epoch [23/40], Iter [2000/6846] Loss: 2.4644, average_loss: 3.2647
Epoch [23/40], Iter [3000/6846] Loss: 4.2755, average_loss: 3.2795
Epoch [23/40], Iter [4000/6846] Loss: 8.6257, average_loss: 3.2654
Epoch [23/40], Iter [5000/6846] Loss: 2.0113, average_loss: 3.2823
Epoch [23/40], Iter [6000/6846] Loss: 1.0928, average_loss: 3.2569
Current best test loss 3.50570


Starting epoch 24 / 40
Learning Rate for this epoch: 0.001
Epoch [24/40], Iter [1000/6846] Loss: 1.5265, average_loss: 3.1230
Epoch [24/40], Iter [2000/6846] Loss: 2.3978, average_loss: 3.1700
Epoch [24/40], Iter [3000/6846] Loss: 4.8661, average_loss: 3.1638
Epoch [24/40], Iter [4000/6846] Loss: 3.9850, average_loss: 3.2001
Epoch [24/40], Iter [5000/6846] Loss: 2.2372, average_loss: 3.2118
Epoch [24/40], Iter [6000/6846] Loss: 2.8635, average_loss: 3.1930


Starting epoch 25 / 40
Learning Rate for this epoch: 0.001
Epoch [25/40], Iter [1000/6846] Loss: 4.7675, average_loss: 3.2833
Epoch [25/40], Iter [2000/6846] Loss: 2.3316, average_loss: 3.2158
Epoch [25/40], Iter [3000/6846] Loss: 2.0420, average_loss: 3.2181
Epoch [25/40], Iter [4000/6846] Loss: 3.0235, average_loss: 3.2262
Epoch [25/40], Iter [5000/6846] Loss: 2.6444, average_loss: 3.2362
```

결과적으로 23번째 에폭에서 가장 낮은 best loss가 나왔습니다.

Test

가장 최선의 모델을 불러와 test image들을 테스트하여 박스를 그리고 'Test Result'폴더에 각각 저장하였습니다.