

프로젝트 3-2 레포트

딥러닝 모델을 하기에 앞서, 여러 데이터 모델을 보고, 그 중 VGGNet 16을 구현해보기로 결정했다.

- class MyDataset(Dataset)

먼저, 커스텀 데이터 세트를 만들어주었다.

```

70 class MyDataset(Dataset) :
71
72     def __init__(self, meta_path, root_dir, transform=None) :
73         super(MyDataset, self).__init__()
74
75         with open(meta_path, "r") as answer:
76             self.labels = json.load(answer)
77             self.classes = self.labels.get('categories')
78             self.labels = self.labels.get('annotations')
79
80
81         # print(self.classes)
82
83         self.root_dir = root_dir
84         self.transform = transform
85
86     def __len__(self) :
87         return len(self.labels)
88
89     def __getitem__(self, idx) :
90         ipath = self.root_dir + "/" + self.labels[idx].get('file_name')
91         #print(ipath)
92         image = Image.open(ipath)
93         if image.mode != 'RGB':
94             image = image.convert('RGB')
95         #print(image)
96         label = [int(self.labels[idx].get('category')), self.labels[idx].get('file_name')]

```

데이터 세트의 구현 자체는 PyTorch 공식 사이트와 설명을 참조해서 만들었다. 파일의 이름이 포함된 json파일을 불러와서 annotations를 추출한 뒤, 각각의 이미지와 라벨을 불러오도록 구현하였다. 라벨은 리스트로 label[0]은 category를, label[1]은 file_name을 저장하도록 구현하였다.

데이터셋의 len은 파일 명 리스트 길이와 같을 것임으로 self.labels의 길이를 불러오도록 설정하였다.

- class MyModel(nn.Module)

먼저 VGGNet의 형태는 다음과 같다.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

이 중, B열에 있는 VGGNet 13을 구현하였는데, 코드는 다음과 같다.

```

def blocks(self):
    block = []
    #layer1
    block.extend([nn.Conv2d(3,64,kernel_size=3,padding=1),nn.BatchNorm2d(64),nn.ReLU())])
    block.extend([nn.Conv2d(64,64,kernel_size=3,padding=1),nn.BatchNorm2d(64),nn.ReLU())])
    block.extend([nn.MaxPool2d(2, stride=2)])
    #layer2
    block.extend([nn.Conv2d(64,128,kernel_size=3,padding=1),nn.BatchNorm2d(128),nn.ReLU())])
    block.extend([nn.Conv2d(128,128,kernel_size=3,padding=1),nn.BatchNorm2d(128),nn.ReLU())])
    block.extend([nn.MaxPool2d(2, stride=2)])
    #layer3
    block.extend([nn.Conv2d(128,256,kernel_size=3,padding=1),nn.BatchNorm2d(256),nn.ReLU())])
    block.extend([nn.Conv2d(256,256,kernel_size=3,padding=1),nn.BatchNorm2d(256),nn.ReLU())])

    block.extend([nn.MaxPool2d(2, stride=2)])
    #layer4
    block.extend([nn.Conv2d(256,512,kernel_size=3,padding=1),nn.BatchNorm2d(512),nn.ReLU())])
    block.extend([nn.Conv2d(512,512,kernel_size=3,padding=1),nn.BatchNorm2d(512),nn.ReLU())])

    block.extend([nn.MaxPool2d(2, stride=2)])

    #layer5
    block.extend([nn.Conv2d(512,512,kernel_size=3,padding=1),nn.BatchNorm2d(512),nn.ReLU())])
    block.extend([nn.Conv2d(512,512,kernel_size=3,padding=1),nn.BatchNorm2d(512),nn.ReLU())])

    block.extend([nn.MaxPool2d(2, stride=2)])

    return nn.Sequential(*block)

```

```

def linears(self):
    block = []

    #layer6 ( )
    block.extend([nn.Linear(25088,4096),nn.ReLU(),nn.Linear(4096,4096),nn.ReLU(),nn.Linear(4096,1000),nn.ReLU(),nn.Linear(1000,80)])
    return nn.Sequential(*block)

```

def blocks() 안 쪽에 VGGNet의 모양을 그대로 구현하였다. 13 레이어로 선택한 이유는 레이어가 비교적 적어 연산이 빠를 것이라 생각했기 때문이다.

MaxPool2d를 제외한 각각의 레이어는 nn.Conv2d,nn.BatchNorm2d,nn.ReLU를 순서대로 적용한 블록을 포함하고 있다. nn.BatchNorm2d는 적용하지 않아도 되지만, 모델의 성능을 위해 추가로 넣어주었다. 그 후 x의 사이즈를 조정해준 후 FC layer를 통과시켜서 classification이 되도록 하였다.

- def train()

```

114 def train() :
115     #TODO: Make your own training code
116
117     # You SHOULD save your model by
118     # torch.save(model.state_dict(), './checkpoint.pth')
119     # You SHOULD not modify the save path
120
121     if torch.cuda.is_available():
122         device = torch.device("cuda")
123     else:
124         device = torch.device("cpu")
125
126
127     train_transform = transforms.Compose([
128         #transforms.ToPILImage(),
129         transforms.RandomResizedCrop(224),
130         transforms.ColorJitter(hue=0.1,contrast=0.1,saturation=0.1,),
131         transforms.RandomHorizontalFlip(),
132         transforms.ToTensor()
133     ])
134
135
136     train_dataset = MyDataset('./answer.json', './train_data', transform=train_transform)
137     train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
138     model = MyModel()
139     model = model.to(device)
140
141     criterion = nn.CrossEntropyLoss()
142     optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

```

```

145         for epoch in range(3):
146
147             running_loss= 0
148             for i,data in enumerate(train_loader,0):
149                 image, label = data
150                 image = image.to(device)
151                 #label = list(map(int, label))
152                 label = label[0]
153                 label = torch.tensor(label)
154                 label = label.to(device)
155                 optimizer.zero_grad()
156
157                 output = model(image)
158                 loss = criterion(output,label)
159                 loss.backward()
160                 optimizer.step()
161
162                 running_loss += loss.item()
163                 if i % 100 == 99:
164                     print('%d,%5d] loss: %.3f' %
165                           (epoch+1, i+1, running_loss/100))
166                     running_loss = 0

```

위와 같은 코드를 통해 train을 진행해주었다. 먼저 cuda가 사용가능한지 체크해주었고, 그 후 이

미지의 overfitting을 막기 위해 여러 transform을 추가해준 후, tensor로 변환해주었다. Crop의 사이즈가 224인 이유는 VGGNet에서 인풋을 224*224로 주었기 때문에 비슷하게 구현하였다. 그 후, criterion과 optimizer를 선언해준 후, for loop로 트레이닝을 진행해주었다. Label을 tensor로 바꾸어 주기 위한 코드가 153줄에 있고, 나머지는 강의에 나온 그대로 만들어주었다.

```
[1, 100] loss: 4.796
[1, 200] loss: 4.371
[1, 300] loss: 4.355
[1, 400] loss: 4.337
[1, 500] loss: 4.300
[1, 600] loss: 4.265
[2, 100] loss: 4.215
[2, 200] loss: 4.200
[2, 300] loss: 4.147
```

100개의 배치마다 로스를 확인한 결과다. Loss가 점점 줄어들고 있는 것을 확인할 수 있다.

- def test()

```
data_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.RandomResizedCrop(224)
])
jsonmaker(data_dir)
keydata = './tojson.json'
# Create training and validation datasets
test_datasets = MyDataset(keydata, data_dir, data_transforms)

# Create training and validation dataloaders
test_dataloader = torch.utils.data.DataLoader(test_datasets, batch_size=64, shuffle=False, num_workers=4)

# Detect if we have a GPU available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Send the model to GPU
model = model.to(device)

# Set model as evaluation mode
for param in model.parameters():
    param.requires_grad = False
model.eval()
```

아래와 같이 리사이징을 적용하고, 스케লে톤에 있는 코드를 거의 그대로 사용하였다.

```
104 def jsonmaker(path):
105
106     data = {"annotations": [], "categories": []}
107
108     for i in os.listdir(path):
109         data['annotations'].append({"file_name": i, "category": 0})
110     with open('./tojson.json', 'w', encoding='utf-8') as file:
111         json.dump(data, file)
```

Jsonmaker는 구현한 MyDataset이 json파일이 있어야 작동하기 때문에 같은 형식으로 json파일을 만들어주는 함수를 간단하게 작성하였다.

- def main()

```
def main() :  
    train()  
    test()
```

모든 함수를 train과 test안에서 돌아가도록 작성하였기 때문에 두 함수만 호출하도록 했다.