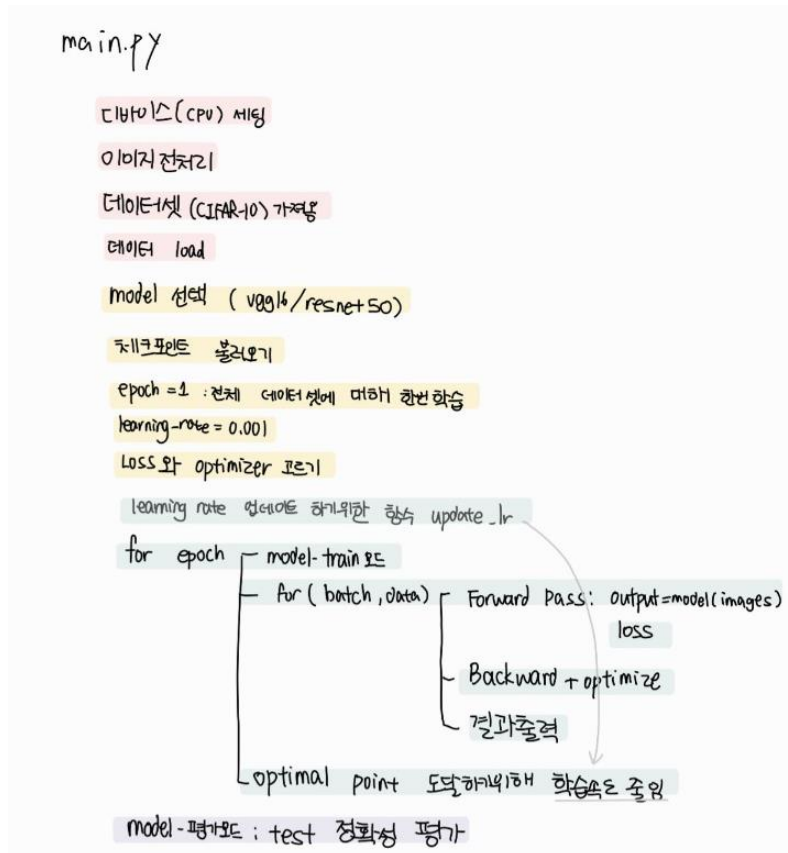


1. main.py

2, 3번의 모델을 이용해 cnn을 구현할 코드이다.

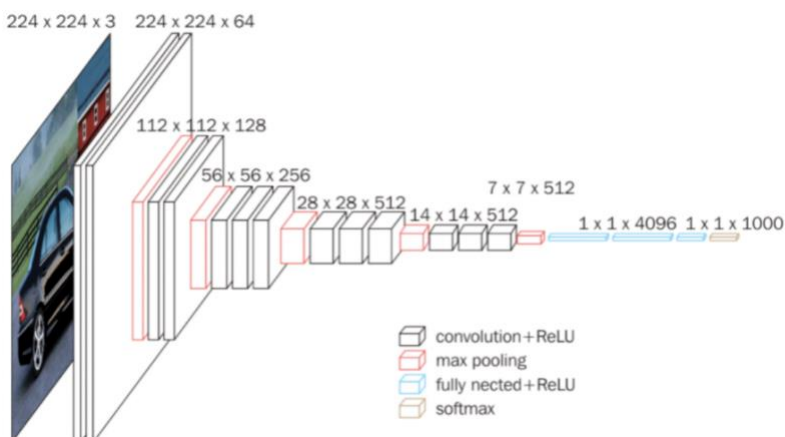
OSP-Lec14-CNN architecture-practice-v2.pdf를 토대로 main함수의 실행 흐름을 정리하였다.



2. vgg16_fully.py

main.py에서 model를 vgg16()로 설정하여 vgg16_fully.py의 객체와 함수를 이용한다.

VGGNet을 구현한 코드로, 작은 size의 filter로 deep한 network이다. 3x3 conv와 max pooling 만을 이용한다. 작은 size로 여러 번 filtering하는 것이 큰 size로 적게 filtering하는 것 보다 좋은 성능을 가지기 때문에 사용한다.



```
def vgg16():
    cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M']
    return VGG(make_layers(cfg))
```

main.py함수에서 vgg16함수를 호출한다. cfg에는 kernel size의 정보가 담겨 있고, max pooling인 경우 'M'이 저장되어 있다. 이 배열을 make_layers()함수에 담고, 반환되는 data를 VGG에 전달한다.

```
def make_layers(cfg, batch_norm=False):
    layers = []
    in_channels = 3
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)
```

make_layer함수는 kernel정보를 받아 해당 연산을 layer 배열에 저장하고, sequential container 형태를 반환한다.

```
class VGG(nn.Module):
    def __init__(self, features):
        super(VGG, self).__init__()
        self.features = features
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(512, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(512, 10),
        )
        # Initialize weights
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
                m.bias.data.zero_()
```

이 정보를 받아 VGG 객체가 생성되며 init함수를 실행한다. 고정된 내용을 class내에 설계한다. for 문을 돌며 모델 클래스에 정의된 layer들을 반환한다.

VGG의 forward함수는 forward prop을 진행하는 함수이다.

실행 결과

```

1 import torch
2 import torch.nn as nn
3 import torchvision
4 import torchvision.transforms as transforms
5 from vgg16_fall import *
6
7 # device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
8 device = torch.device('cpu')
9
10 # Image Preprocessing
11 transform_train = transforms.Compose([
12     transforms.RandomCrop(32, padding=4),
13     transforms.RandomHorizontalFlip(),
14     transforms.ToTensor(),
15     transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
16 ])
17
18 transform_test = transforms.Compose([
19     transforms.ToTensor(),
20     transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
21 ])
22
23 # CIFAR-10 Dataset
24 train_dataset = torchvision.datasets.CIFAR10(root='./data/')
25
26
27 Run: main
28 C:\Users\user\anaconda3\envs\PyTorch_env\python.exe C:\Users\user\PycharmProjects\pythonProject\main.py
29 Epoch [1/1], Step [100/500] Loss: 0.1820
30 Epoch [1/1], Step [200/500] Loss: 0.1731
31 Epoch [1/1], Step [300/500] Loss: 0.1607
32 Epoch [1/1], Step [400/500] Loss: 0.1557
33 Epoch [1/1], Step [500/500] Loss: 0.1906
34 Accuracy of the model on the test images: 84.83 %
35
36 Process finished with exit code 0

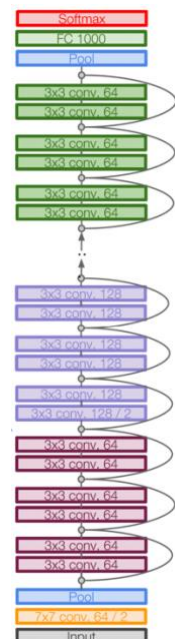
```

3. resnet50.py

main.py에서 model를 ResNet50_layer4()로 설정하여 resnet50.py의 객체와 함수를 이용한다. ResNet을 구현한 코드이다. ResNet은 residual blocks을 사용한다.

이 코드는 기존의 ResNet과는 조금 다른, 아래 표와 같은 구성을 가진다.

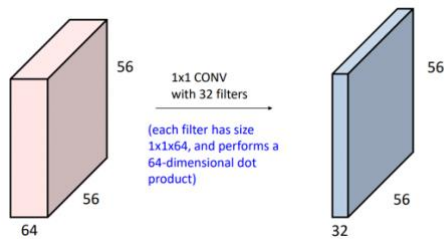
Layer number	Network	Output Image size
Layer 1	7x7 conv, channel = 64, stride = 2 3x3 max pool, stride = 2	8 x 8
Layer 2	[1x1 conv, channel = 64, 3x3 conv, channel = 64, 1x1 conv, channel = 256] x 2 [1x1 conv, channel = 64, stride = 2 3x3 conv, channel = 64, 1x1 conv, channel = 256] x 1	4 x 4
Layer 3	[1x1 conv, channel = 128, 3x3 conv, channel = 128, 1x1 conv, channel = 512] x 3 [1x1 conv, channel = 128, stride = 2 3x3 conv, channel = 128, 1x1 conv, channel = 512] x 1	2 x 2
Layer 4	[1x1 conv, channel = 256, 3x3 conv, channel = 256, 1x1 conv, channel = 1024] x 6	2 x 2
	AvgPool	1 x 1
	Fully connected layer	?



기존의 ResNet은 오른쪽 그림과 같다.

우선 1x1 convolution과 3x3 convolution 함수를 정의한다.

```
# 1x1 convolution
def conv1x1(in_channels, out_channels, stride, padding):
    model = nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, padding=padding),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
    )
    return model
```



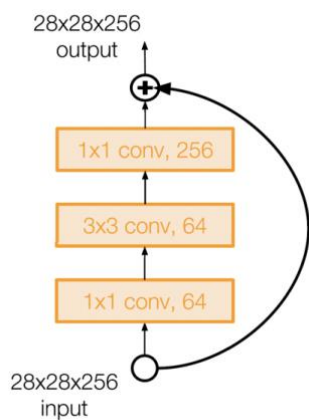
conv -> BatchNorm -> ReLU의 과정을 거친다.

```
# 3x3 convolution
def conv3x3(in_channels, out_channels, stride, padding):
    model = nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=padding),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
    )
    return model
```

3x3 convolution 함수 또한 마찬가지이다.

1) bottle neck building block

```
# Question 1 : Implement the "bottle neck building block" part.
# Hint : Think about difference between downsample True and False. How we make the difference by code?
```



bottleneck building block은 왼쪽의 그림과 같다. 1x1, 3x3, 1x1 convolution을 사용한다. OSP-Lec14-CNN architecture-practice-v2.pdf에 언급된 것 처럼, 이 세 layer을 수행한 후 LeRU를 적용한다. 앞 layer의 output은 다음 layer의 input으로 사용한다.

downsampling을 한다는 것은 이미지의 크기가 줄어드는 것이기 때문에 아래의 식에서 stride를 2로 하면 Output size가 절반이 된다. stride가 1이라면 이미지의 크기는 변하지 않는다.

$$\text{Output size} = (N - F) / \text{stride} + 1$$

따라서 downsampling이 true인 경우에는 stride를 2로, false인 경우엔 stride를 1로 하면 된다.

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, middle_channels, out_channels, downsample=False):
        super(ResidualBlock, self).__init__()
        self.downsample = downsample

        if self.downsample:
            self.layer = nn.Sequential(
                conv1x1(in_channels, middle_channels, 2, 0),
                conv3x3(middle_channels, middle_channels, 1, 1),
                conv1x1(middle_channels, out_channels, 1, 0))
            self.downsize = conv1x1(in_channels, out_channels, 2, 0)

        else:
            self.layer = nn.Sequential(
                conv1x1(in_channels, middle_channels, 1, 0),
                conv3x3(middle_channels, middle_channels, 1, 1),
                conv1x1(middle_channels, out_channels, 1, 0)
            )

        self.make_equal_channel = conv1x1(in_channels, out_channels, 1, 0)
```

이렇게 stride의 사용 여부에 따라 다르게 구성할 수 있는 Residual Block이 완성되었다. Residual Block을 여러 layers로 이용하여 ResNet을 구현할 수 있다.

2) ResNet50_layer4

1번에서 만든 Residual Block을 이용해 제시된 net 구성에 맞는 layer들을 만든다.

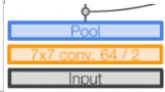
```
# Question 2 : Implement the "class, ResNet50_layer4" part.
# Understand ResNet architecture and fill in the blanks below. (25 points)
# (blank : #blank#, 1 points per blank )
# Implement the code.
```

```
class ResNet50_layer4(nn.Module):
    def __init__(self, num_classes=10):
        super(ResNet50_layer4, self).__init__()
```

Cifar-10의 class는 이름에 있듯이 10개이다.

layer1이다.

```
self.layer1 = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
)
```

Layer 1	7x7 conv, channel = 64, stride = 2 3x3 max pool, stride = 2	8 x 8	
---------	--	-------	---

7x7conv이므로 filter size는 7, stride는 2이다. output 이미지 size는 8이다. stride가 2라 activation map의 사이즈가 절반으로 줄어들어야 하기 때문에, input 이미지 size는 16이다.

$$W_2 = (W_1 - F + 2P) / S + 1$$

이 식을 이용하면,

$$8 = \lfloor (16 - 7 + 2P) / 2 \rfloor + 1$$

$$7 = \lfloor (16 - 7 + 2P) / 2 \rfloor$$

$$14 = 9 + 2P$$

$$5 = 2P$$

$$P = 3 \text{이다}$$

따라서 padding은 3이다. 매개변수를 순서대로 채워준다.

3x3의 max pool이 stride=2을 가지므로, MaxPool 에도 값을 채운다.

layer2이다.

```
self.layer2 = nn.Sequential(
    ResidualBlock(in_channels=64, middle_channels=64, out_channels=256, downsample=False),
    ResidualBlock(in_channels=256, middle_channels=64, out_channels=256, downsample=False),
    ResidualBlock(in_channels=256, middle_channels=64, out_channels=256, downsample=True)
)
```

Layer 2	[1x1 conv, channel = 64, 3x3 conv, channel = 64, 1x1 conv, channel = 256] x 2 [1x1 conv, channel = 64, stride = 2 3x3 conv, channel = 64, 1x1 conv, channel = 256] x 1	4 x 4
---------	---	-------

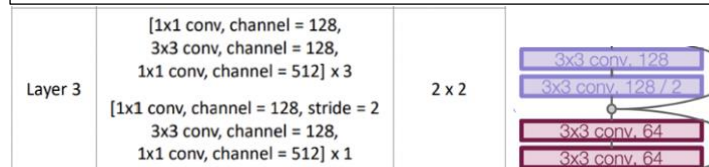
이전 Layer의 output을 input으로 사용한다.

3개의 residual block이 들어있고, 마지막 residual의 결과로 이미지의 크기가 절반으로 줄어들어야 한다. 따라서 마지막 ResidualBlock 함수에서 downsample을 True로 해준다.

이미지는 8x8에서 4x4가 되고, channel size는 64에서 256이 된다.

layer3이다.

```
self.layer3 = nn.Sequential(
    ResidualBlock(256, 128, 512, False),
    ResidualBlock(512, 128, 512, False),
    ResidualBlock(512, 128, 512, False),
    ResidualBlock(512, 128, 512, True)
)
```

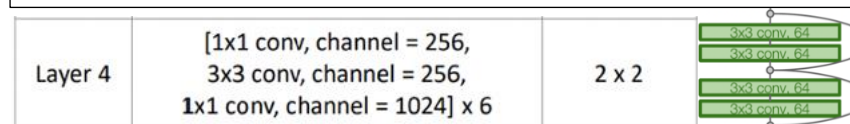


4개의 residual block이 있고, layer2와 동일하게 마지막 ResidualBlock함수의 downsample을 True로 하여 호출한다. 이전 Layer의 output을 input으로 사용한다.

이미지 크기는 8x8에서 4x4가 되었고, channel size는 256에서 512가 되었다.

layer4이다.

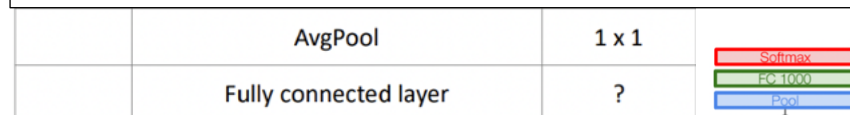
```
self.layer4 = nn.Sequential(
    ResidualBlock(512, 256, 1024, False),
    ResidualBlock(1024, 256, 1024, False),
    ResidualBlock(1024, 256, 1024, False),
    ResidualBlock(1024, 256, 1024, False),
    ResidualBlock(1024, 256, 1024, False),
    ResidualBlock(1024, 256, 1024, False)
)
```



6개의 residual block이 있다. 이미지 크기는 변화없고, 채널개수가 증가한다

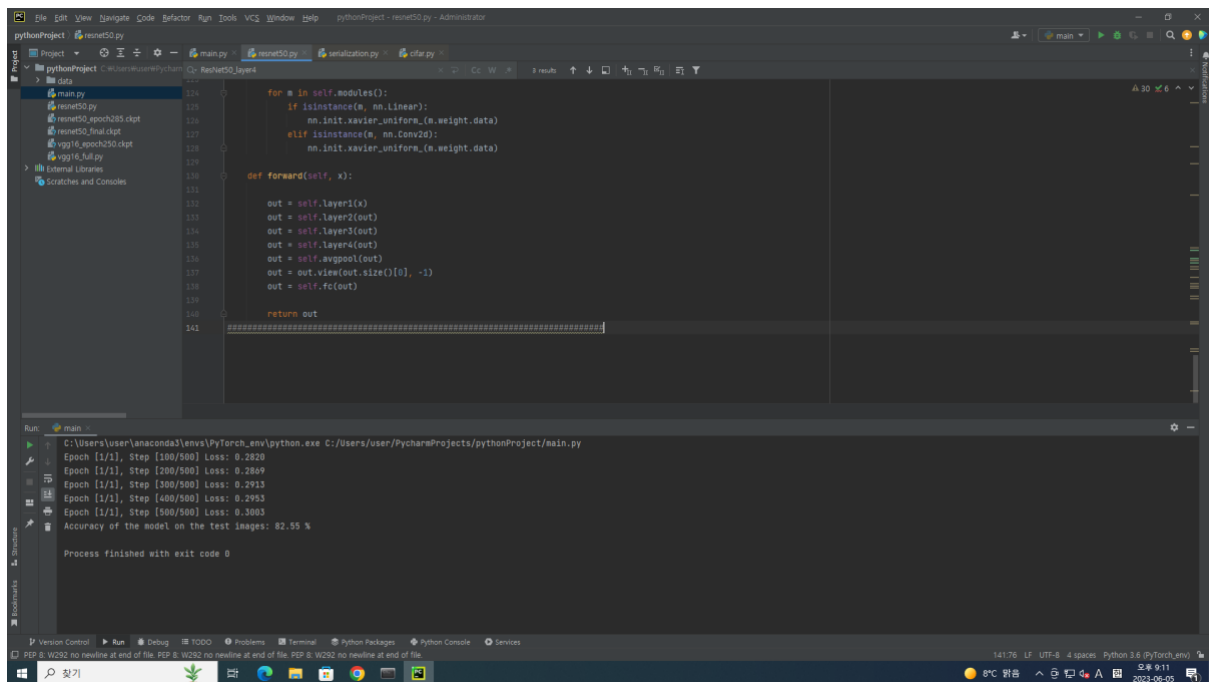
```
self.fc = nn.Linear(1024, 10)
self.avgpool = nn.AvgPool2d(2, 2)

for m in self.modules():
    if isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight.data)
    elif isinstance(m, nn.Conv2d):
        nn.init.xavier_uniform_(m.weight.data)
```



Fully connected layer를 사용하여 channel size를 조절한다.FC1000을 10개의 class에 맞도록 조절하는 것이다. AvgPool을 수행한다.Pool은 4개씩 묶어서 하므로 2,2를 넣어주었다.

실행 결과



```
pythonProject - resnet50.py - Administrator
pythonProject
  Project
  pythonProject
    data
    main.py
    resnet50.py
    resnet50_epoch285ckpt
    resnet50_finalckpt
    vgg16_epoch250ckpt
    vgg16_full.py
    External Libraries
    Scratches and Consoles
  Run
  Run: main ...
  Epoch [1/1], Step [100/500] Loss: 0.2828
  Epoch [1/1], Step [200/500] Loss: 0.2849
  Epoch [1/1], Step [300/500] Loss: 0.2913
  Epoch [1/1], Step [400/500] Loss: 0.2953
  Epoch [1/1], Step [500/500] Loss: 0.3003
  Accuracy of the model on the test images: 82.55 %
  Process finished with exit code 0
```

참고자료:

오픈소프트웨어프로젝트 수업자료 12장, 13장, 14장, 14장 연습 pdf