

Technical Report

2071044 컴퓨터공학과 정민정

0. vgg16_full.py

1) overall explanation of code

VGG model을 구현한 code이다.

2) Code

```
import torch.nn as nn
import math

##### VGG16 #####
class VGG(nn.Module): # torch.nn.Module을 상속받는 VGG class
    # VGG class 생성자
    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():
            # isinstance: 차례로 layer를 입력하여 layer의 형태를 반환
            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_()
        # Network forward function
    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x

def make_layers(cfg, batch_norm=False):
    layers = []
    in_channels = 3
    # cfg의 입력값에 따라 layers에 차례로 쌓아감
    for v in cfg:
        if v == 'M': # max pooling 수행
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            # Convolution + ReLu 수행
```

```

        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)

def vgg16():
    # cfg shows 'kernel size'
    # 'M' means 'max pooling'
    cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M']
    return VGG(make_layers(cfg))

```

3) Analysis and Explanation of code

■ VGG class는 torch.nn.Module을 상속받는 class로, VGG16 model의 객체를 나타내는 class다

◦ features 메소드는 input image에 VGG의 layer들을 적용한다. VGG의 layer 정보는 cfg 배열을 통해 알 수 있으며, make_layer 함수를 이용하여 layer들의 sequence를 생성한다. VGG의 feature를 생성한 다음 VGG class의 객체를 만드는 과정은 vgg16함수에서 수행된다.

■ make_layers 함수는 cfg 배열의 값에 따라 layers 배열에 layer를 차례로 추가한 다음, 이를 torch.nn.Sequential메소드로 묶어 return하는 함수이다. cfg의 값이 'M'이면 kernel size=2, stride=2인 max pooling layer를 추가하고, cfg의 값이 숫자면 conv layer + ReLU layer를 추가한다.

4) Result는 main.py 실행 결과에 함께 첨부하였다.

1. resnet50_skeleton.py

1) overall explanation of code

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	?

위의 구조를 가지는 ResNet50을 구현한 code이다.

2) Code

```
import torch.nn as nn

# 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

# 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

#####
##
# 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?
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:
            # the output image's size is half of the input image size.
            self.layer = nn.Sequential(
                #####
                ##### fill in here (20 points)
                # Hint : use these functions (conv1x1, conv3x3)
                #####
                conv1x1(in_channels, middle_channels, 2, 0), # number of channels:
                in_channels -> middle_channels, size: be halved
                conv3x3(middle_channels, middle_channels, 1, 1), # number of channels &
                size: no changed
                conv1x1(middle_channels, out_channels, 1, 0) # number of channels:
                middle_channels -> out_channels, size: no changed
            )
            # the input image's size is halved.
            self.downsize = conv1x1(in_channels, out_channels, 2, 0)
        else:
            # the output image's size is not changed.
            self.layer = nn.Sequential(
                #####
                ##### fill in here (20 points)
```

```

#####
conv1x1(in_channels, middle_channels, 1, 0), # number of channels:
in_channels -> middle_channels
conv3x3(middle_channels, middle_channels, 1, 1),
conv1x1(middle_channels, out_channels, 1, 0) # number of channels:
middle_channels -> out_channels
)
# make number of input image's channel = number of output image's channel
self.make_equal_channel = conv1x1(in_channels, out_channels, 1, 0) # number of
channels: in_channels -> out_channels

def forward(self, x):
    if self.downsample:
        out = self.layer(x) # output image
        x = self.downsize(x) # input image
        return out + x # output image + input image
    else:
        out = self.layer(x) # output image
        if x.size() is not out.size(): # input image
            x = self.make_equal_channel(x)
        return out + x # output image + input image
#####
##

#####
##
# 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): # Hint : How many classes in Cifar-10 dataset?
        super(ResNet50_layer4, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 64, 7, 2, 3), # input channel sizes:3, output channel sizes:64, kernel
size:7, stride:2, padding:3
            # Hint : Through this conv-layer, the input image size is halved.
            # Consider stride, kernel size, padding and input & output channel
sizes.
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(3, 2, 0) # input image size is halved.(kernel size:3, stride:2,
padding:0)
        )
        self.layer2 = nn.Sequential(
            ResidualBlock(64, 64, 256, False), # number of channels: 64->64->64->256, image
size: no change
            ResidualBlock(256, 64, 256, False), # number of channels: 256->64->64->256,
image size: no change
            ResidualBlock(256, 64, 256, True) # number of channels: 256->64->64->256, image
size: be halved
        )
        self.layer3 = nn.Sequential(

```

```

#####
##### fill in here (20 points)
##### you can refer to the 'layer2' above
#####
ResidualBlock(256, 128, 512, False), # number of channels: 256->128->128->512,
image size: no change
ResidualBlock(512, 128, 512, False), # number of channels: 512->128->128->512,
image size: no change
ResidualBlock(512, 128, 512, False), # number of channels: 512->128->128->512,
image size: no change
ResidualBlock(512, 128, 512, True) # number of channels: 512->128->128->512,
image size: behalved
)
self.layer4 = nn.Sequential(
#####
##### fill in here (20 points)
##### you can refer to the 'layer2' above
#####
ResidualBlock(512, 256, 1024, False), # number of channels: 512->256->256-
>1024, image size: no change
ResidualBlock(1024, 256, 1024, False), # number of channels: 1024->256->256-
>1024, image size: no change
ResidualBlock(1024, 256, 1024, False), # number of channels: 1024->256->256-
>1024, image size: no change
ResidualBlock(1024, 256, 1024, False), # number of channels: 1024->256->256-
>1024, image size: no change
ResidualBlock(1024, 256, 1024, False), # number of channels: 1024->256->256-
>1024, image size: no change
ResidualBlock(1024, 256, 1024, False) # number of channels: 1024->256->256-
>1024, image size: no change
)

self.fc = nn.Linear(1024, num_classes) # Hint : Think about the reason why fc layer
is needed
self.avgpool = nn.AvgPool2d(2, 1) # image size: 2x2 -> 1x1 (kernel size:2, stride:1)

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)

def forward(self, x):

    out = self.layer1(x)
    out = self.layer2(out)
    out = self.layer3(out)
    out = self.layer4(out)
    out = self.avgpool(out)
    out = out.view(out.size()[0], -1)
    out = self.fc(out)

    return out
#####
##

```

3) Analysis and Explanation of code

■ Question 1: Implement the "bottle neck building block"

ResidualBlock class는 torch.nn.Module을 상속받는 class로, Residual Block 객체를 나타내는 class이다. downsample = True이면 output image의 크기를 input image의 1/2배로 줄이고, downsample = False이면 output image의 크기를 변화시키지 않는다. 계산이 끝난 이후엔 output 결과와 input 결과를 더한 값을 리턴한다. ($F(x) + x$)

▷ downsample = True

- layer 메소드는 input image에 1x1 conv layer→3x3 conv layer→1x1 conv layer를 차례로 적용한다. 첫번째 1x1 conv layer의 stride는 2로 설정하여 image의 크기를 input image의 1/2줄이고, 이후의 3x3 conv layer는 stride=1, padding=1($\because \text{padding} = \frac{3-1}{2} = 1$), 1x1 conv layer는 stride=1, padding=0으로 설정하여 image의 크기가 변하지 않도록 한다.
- image의 channel 개수는 아래와 같이 변한다.

1x1 conv layer를 적용한 이후: in_channels→middle_channels
3x3 conv layer를 적용한 이후: middle_channels→middle_channels
1x1 conv layer를 적용한 이후: middle_channels→out_channels

- ResNet에서는 output 결과와 input 결과를 더한 값을 리턴하므로, ($F(x) + x$)

1/2배로 줄어든 output 결과와 크기를 맞추기 위해 downsize 메소드를 이용한다. downsize 메소드는 stride=2, padding=0인 1x1 conv layer를 적용하여 input image의 크기를 반으로 줄이고 channel의 개수를 out_channels로 맞춘다.

▷ downsample = False

- layer 메소드는 input image에 1x1 conv layer→3x3 conv layer→1x1 conv layer를 차례로 적용한다. 첫번째 1x1 conv layer는 stride=1, padding=0, 이후의 3x3 conv layer는 stride=1, padding=1($\because \text{padding} = \frac{3-1}{2} = 1$), 1x1 conv layer는 stride=1, padding=0으로 설정하여 image의 크기가 변하지 않도록 한다.
- image의 channel 개수는 아래와 같이 변한다.

1x1 conv layer를 적용한 이후: in_channels→middle_channels
3x3 conv layer를 적용한 이후: middle_channels→middle_channels
1x1 conv layer를 적용한 이후: middle_channels→out_channels

- ResNet에서는 output 결과와 input 결과를 더한 값을 리턴하므로, ($F(x) + x$)

input 결과와 output 결과의 channel 개수를 맞추기 위해 make_equal_channel 메소드를 이용한다. make_equal_channel 메소드는 stride=1, padding=0인 1x1 conv layer를 적용하여 input image의 channel 개수를 out_channels로 맞춘다. make_equal_channel 메소드는 input image의 크기와 output image의 크기가 다른 경우에만 input image에 적용한다.

■ Question 2: Implement the "ResNet50_layer4"

ResNet50_layer4는 torch.nn.Module을 상속받는 class로, Layer가 4개인 ResNet50 model의 객체를 나타내는 class이다.

◦ layer1 메소드는 input image에 kernel size=7, output channel size: 64, stride=2 padding=3인 conv layer와 kernel size=3, stride=2, padding=0인 max pool layer, 그리고 그 사이에 ReLU layer를 적용하는 메소드이다. conv layer를 적용하는 torch.nn.module.Conv2d 메소드와 max pool layer를 적용하는 torch.nn.module.MaxPool2d 메소드를 이용한다. 7x7 conv layer를 통과하면 image의 channel은 3개에서 64개로 늘어나고, max pool layer를 통과하면 image의 크기는 절반으로 줄어든다.

◦ layer 2 메소드는 input image에 3개의 Residual Block를 통과시켜 channel을 64개에서 256개로, image의 크기는 절반으로 줄인다. 마지막 Residual Block의 downsample 파라미터를 True로 설정하여 image의 크기가 절반으로 줄어들도록 만든다. image의 channel 개수는 아래와 같이 변한다.

첫 번째 Residual Block를 적용: 64→64→64→256
두 번째 Residual Block을 적용: 256→64→64→256
세 번째 Residual Block을 적용: 256→64→64→256

◦ layer 3 메소드는 input image에 4개의 Residual Block를 통과시켜 channel을 256개에서 512개로, image의 크기는 절반으로 줄인다. 마지막 Residual Block의 downsample 파라미터를 True로 설정하여 image의 크기가 절반으로 줄어들도록 만든다. image의 channel 개수는 아래와 같이 변한다.

첫 번째 Residual Block를 적용: 256→128→128→512
두 번째 Residual Block을 적용: 512→128→128→512
세 번째 Residual Block을 적용: 512→128→128→512
네 번째 Residual Block을 적용: 512→128→128→512

◦ layer 4 메소드는 input image에 6개의 Residual Block를 통과시켜 channel을 512개에서 1024개로 늘리고, image의 크기는 변화시키지 않는다. 모든 Residual Block의 downsample 파라미터를 False로 설정하여 image의 크기가 변하지 않도록 만든다. image의 channel 개수는 아래와 같이 변한다.

첫 번째 Residual Block를 적용: 512→256→256→1024
두 번째 Residual Block을 적용: 1024→256→256→1024
세 번째 Residual Block을 적용: 1024→256→256→1024
네 번째 Residual Block을 적용: 1024→256→256→1024
다섯 번째 Residual Block을 적용: 1024→256→256→1024
여섯 번째 Residual Block을 적용: 1024→256→256→1024

- avgpool 메소드는 torch.nn.module.AvgPool2d 메소드를 이용해서 image의 크기를 1x1로 바꾼다. 주어진 input image에 따르면 avgpool 메소드에 입력되는 image의 크기는 2x2이므로, kernel size=2, stride=1로 설정하여 image의 크기를 1x1로 바꾼다.

- fc 메소드는 torch.nn.module.Linear 메소드를 이용하여 선형회귀모델을 구현한다. 첫 번째 파라미터에는 입력의 차원, 두 번째 파라미터에는 출력의 차원을 입력한다. fc 메소드에 입력되는 image의 크기는 1x1024이고, 데이터는 총 num_classes개의 종류로 나뉘어지므로 첫번째 파라미터에 1024, 두 번째 파라미터에 num_classes(=우리가 사용한 데이터는 10)를 전달한다.

4) Result는 main.py 실행 결과에 함께 첨부하였다.

2. main.py

1) overall explanation of code

model을 VGG16과 ResNet50중 선택한 다음, model을 train하는 code이다. CIFAR-10 Dataset을 이용하고, Loss 함수로는 CrossEntropyLoss를 선택했다.

2) Code

(1) Train "VGG-16"

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from vgg16_full import *

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
# device = torch.device('cpu')

# Image Preprocessing
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])
```



```

# CIFAR-10 Dataset
train_dataset = torchvision.datasets.CIFAR10(root='../osproj/data/',
                                              train=True,
                                              transform=transform_train,
                                              download=False) # Change Download-flag
"True" at the first excution.

test_dataset = torchvision.datasets.CIFAR10(root='../osproj/data/',
                                             train=False,
                                             transform=transform_test)

# data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=100,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=100,
                                          shuffle=False)

#####
model = vgg16().to(device) # initialize
PATH = '../Downloads/skeleton code-Lec14/skeleton code-Lec14/vgg16_epoch250.ckpt' #
test acc would be almost 85
#####
checkpoint = torch.load(PATH, map_location=device)
model.load_state_dict(checkpoint)

# Train Model
# Hyper-parameters
num_epochs = 1 # students should train 1 epoch because they will use cpu
learning_rate = 0.001

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# For updating learning rate
def update_lr(optimizer, lr):

```

```

for param_group in optimizer.param_groups:
    param_group['lr'] = lr

# Train the model
total_step = len(train_loader)
current_lr = learning_rate

for epoch in range(num_epochs):

    model.train()
    train_loss = 0

    for batch_index, (images, labels) in enumerate(train_loader):
        # print(images.shape)
        images = images.to(device) # "images" = "inputs"
        labels = labels.to(device) # "labels" = "targets"

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

        if (batch_index + 1) % 100 == 0:
            print("Epoch [{}/{}], Step [{}/{}] Loss: {:.4f}"
                  .format(epoch + 1, num_epochs, batch_index + 1, total_step, train_loss /
                        (batch_index + 1)))

    # Decay learning rate
    if (epoch + 1) % 20 == 0:
        current_lr /= 3
        update_lr(optimizer, current_lr)
        torch.save(model.state_dict(), './resnet50_epoch' + str(epoch+1)+'.ckpt')

```

```

# Save the model checkpoint
torch.save(model.state_dict(), './resnet50_final.ckpt')

model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print('Accuracy of the model on the test images: {} %'.format(100 * correct / total))

```

(2) Implement "ResNet-50"

```

import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from resnet50_skeleton import *

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# Image Preprocessing
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

```

```

# CIFAR-10 Dataset
train_dataset = torchvision.datasets.CIFAR10(root='../osproj/data/',
                                             train=True,
                                             transform=transform_train,
                                             download=False) # Change Download-flag
"True" at the first excution.

test_dataset = torchvision.datasets.CIFAR10(root='../osproj/data/',
                                             train=False,
                                             transform=transform_test)

# data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=100,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                           batch_size=100,
                                           shuffle=False)

#####
# Choose model
model = ResNet50_layer4().to(device) # initialize
PATH = '../Downloads/skeleton code-Lec14/skeleton code-Lec14/resnet50_epoch285.ckpt'
# test acc would be almost 80
#####
checkpoint = torch.load(PATH, map_location=device)
model.load_state_dict(checkpoint)

# Train Model
# Hyper-parameters
num_epochs = 1 # students should train 1 epoch because they will use cpu
learning_rate = 0.001

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# For updating learning rate
def update_lr(optimizer, lr):

```

```

    for param_group in optimizer.param_groups:
        param_group['lr'] = lr

# Train the model
total_step = len(train_loader)
current_lr = learning_rate

for epoch in range(num_epochs):

    model.train()
    train_loss = 0

    for batch_index, (images, labels) in enumerate(train_loader):
        # print(images.shape)
        images = images.to(device) # "images" = "inputs"
        labels = labels.to(device) # "labels" = "targets"

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

        if (batch_index + 1) % 100 == 0:
            print("Epoch [{}/{}], Step [{}/{}] Loss: {:.4f}"
                  .format(epoch + 1, num_epochs, batch_index + 1, total_step, train_loss /
                        (batch_index + 1)))

    # Decay learning rate
    if (epoch + 1) % 20 == 0:
        current_lr /= 3
        update_lr(optimizer, current_lr)
        torch.save(model.state_dict(), './resnet50_epoch' + str(epoch+1)+'.ckpt')

```

```
# Save the model checkpoint
torch.save(model.state_dict(), './resnet50_final.ckpt')

model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print('Accuracy of the model on the test images: {} %'.format(100 * correct / total))
```

3) Analysis and Explanation of code

◦ Train 단계

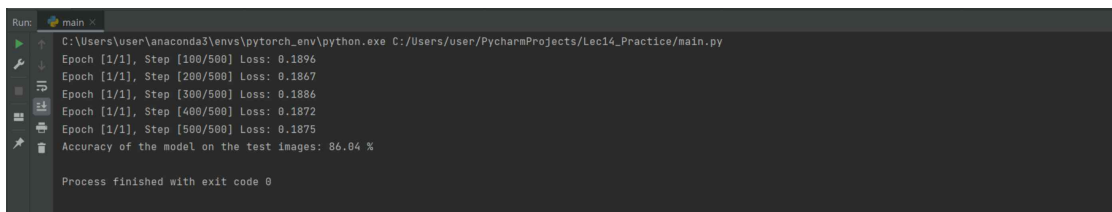
torch.device 메소드를 이용하여 device를 설정한 다음, VGG16과 ResNet50중 하나를 model로 선택한다. 이후 Loss와 Optimizer를 선택한다. (위의 코드에서는 CrossEntropyLoss를 선택하였다.) 이후 train()함수를 이용하여 model을 train mode로 설정하고, enumerate()함수를 이용하기 위해 index와 data를 가져온다. gradient를 0으로 설정한 다음, loss.backward()함수를 이용하여 backpropagation을 실행한다. 1번의 iteration에서 training할 example의 개수를 의미하는 batch size를 설정하고 (위의 코드에서는 batch size를 100으로 설정하였다.), optimal point에 도달하기 위해 learning rate를 감소시킨다. 이후 지정 경로에 model을 저장한다.

◦ Test 단계

eval()을 이용하여 model을 evaluation mode로 설정하고, torch.no_grad()를 실행한다.

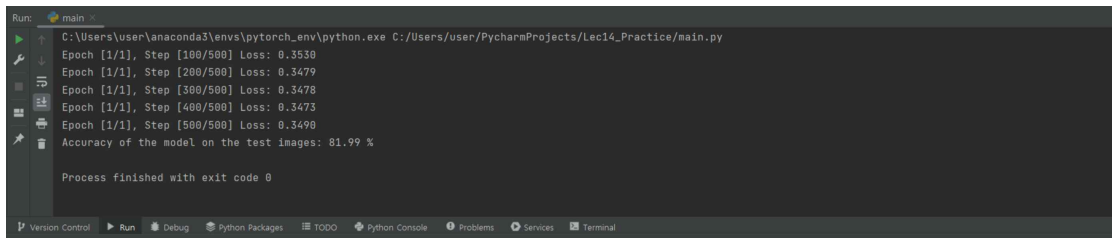
4) Result imagez

(1) model: VGG-16



```
Run: main
C:\Users\User\anaconda3\envs\pytorch_env\python.exe C:/Users/user/PycharmProjects/Lec14_Practice/main.py
Epoch [1/1], Step [100/500] Loss: 0.1896
Epoch [1/1], Step [200/500] Loss: 0.1867
Epoch [1/1], Step [300/500] Loss: 0.1886
Epoch [1/1], Step [400/500] Loss: 0.1872
Epoch [1/1], Step [500/500] Loss: 0.1875
Accuracy of the model on the test images: 86.84 %
Process finished with exit code 0
```

(2) model: ResNet50



The image shows a PyCharm Run console window with a dark theme. The console output displays the progress of a ResNet50 model training over one epoch. The path to the Python executable is shown at the top, followed by five lines of training progress: Epoch [1/1], Step [100/500] Loss: 0.3530, Epoch [1/1], Step [200/500] Loss: 0.3479, Epoch [1/1], Step [300/500] Loss: 0.3478, Epoch [1/1], Step [400/500] Loss: 0.3473, and Epoch [1/1], Step [500/500] Loss: 0.3490. Below these, the test accuracy is reported as 81.99 %. The console ends with the message 'Process finished with exit code 0'. The bottom of the window shows the PyCharm interface with tabs for Version Control, Run, Debug, Python Packages, TODO, Python Console, Problems, Services, and Terminal.

```
Run: main
C:\Users\User\anaconda3\envs\pytorch_env\python.exe C:/Users/user/PycharmProjects/Lec14_Practice/main.py
Epoch [1/1], Step [100/500] Loss: 0.3530
Epoch [1/1], Step [200/500] Loss: 0.3479
Epoch [1/1], Step [300/500] Loss: 0.3478
Epoch [1/1], Step [400/500] Loss: 0.3473
Epoch [1/1], Step [500/500] Loss: 0.3490
Accuracy of the model on the test images: 81.99 %

Process finished with exit code 0
```