**Assignment10 컴퓨터공학과 2071004 권민서**

**VGG16(vgg16\_full.py & main.py)**

1. Overall explanation

: “vgg16\_full.py” and “main.py” train VGG-16 model with CIFAR-10 datasets. This program loads the model, and then train 1 epoch with CPU.

2. Code

vgg16\_full.py

import torch.nn as nn  
import math  
  
###### VGG16 #############  
  
  
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\_()  
  
 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  
 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)  
  
  
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))

main.py

import torch  
import torch.nn as nn  
import torchvision  
import torchvision.transforms as transforms  
# from vgg16\_full import \*  
from resnet50\_skeleton import \*  
from vgg16\_full import vgg16  
  
device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')  
# device = torch.device('cpu')  
  
  
# Image Preprocessing  
# data augmentation and preprocessing using transform.Compose()  
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=True) # Change Download-flag "True" at the first excution.  
  
test\_dataset = torchvision.datasets.CIFAR10(root='../osproj/data/',  
 train=False,  
 transform=transform\_test)  
  
# load data using 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  
# resnet50  
# model = ResNet50\_layer4().to(device)  
# PATH = './resnet50\_epoch285.ckpt' # test acc would be almost 80  
  
# vgg16  
model = vgg16().to(device)  
PATH = './vgg16\_epoch250.ckpt' # test acc would be almost 85  
####################  
###########################################  
checkpoint = torch.load(PATH, map\_location=torch.device('cpu'))  
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  
# use cross entropy loss and Adam 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() # set model to train mode  
 train\_loss = 0  
  
 # get index and data  
 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() # set gradients to zero  
 loss.backward() # conduct backpropagation  
 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')  
  
# test  
model.eval() # set model to evaluation mode  
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 the test accuracy  
 print('Accuracy of the model on the test images: {} %'.format(100 \* correct / total))

3. Explanation and analysis

테이블이(가) 표시된 사진

자동 생성된 설명

VGG16은 Conv layer와 Maxpooling layer가 위와 같이 반복되는 구조이다. 우선, VGG class에서VGG 모델을 정의한다. 그 후, make\_layers() 메소드와 vgg16() 메소드를 이용하여, Max pooling layer인 경우에는 Maxpool2d() 메소드로 구현하고, conv layer인 경우에는 Conv2d() 메소드로 구현한 뒤 레이어들을 리턴한다.

4. Results

텍스트이(가) 표시된 사진

자동 생성된 설명

Test accuracy is 86.2% (almost 85%)

\*ResNet-50\*

1. Overall explanation

: “resnet50\_skeleton.py” and “main.py” implement the ResNet-50 model with CIFAR-10 datasets. This ResNet-50 model uses the bottle neck building block(residual block) and 4 layers.

2. Code

resnet50\_skeleton.py

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:  
 self.layer = nn.Sequential(  
 ##########################################  
 ############## fill in here (20 points)  
 # Hint : use these functions (conv1x1, conv3x3)  
 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(  
 ##########################################  
 ############# fill in here (20 points)  
 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)  
  
 def forward(self, x):  
 if self.downsample:  
 out = self.layer(x)  
 x = self.downsize(x)  
 return out + x  
 else:  
 out = self.layer(x)  
 if x.size() is not out.size():  
 x = self.make\_equal\_channel(x)  
 return out + x  
###########################################################################  
  
  
  
###########################################################################  
# 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),  
 # 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, 1)  
 )  
 self.layer2 = nn.Sequential(  
 ResidualBlock(64, 64, 256, False),  
 ResidualBlock(256, 64, 256, False),  
 ResidualBlock(256, 64, 256, True)  
 )  
 self.layer3 = nn.Sequential(  
 ##########################################  
 ############# fill in here (20 points)  
 ####### you can refer to the 'layer2' above  
 ResidualBlock(256, 128, 512, False),  
 ResidualBlock(512, 128, 512, False),  
 ResidualBlock(512, 128, 512, False),  
 ResidualBlock(512, 128, 512, True)  
 #########################################  
 )  
 self.layer4 = nn.Sequential(  
 ##########################################  
 ############# fill in here (20 points)  
 ####### you can refer to the 'layer2' above  
 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)  
 #########################################  
 )  
  
 self.fc = nn.Linear(1024, num\_classes) # Hint : Think about the reason why fc layer is needed  
 self.avgpool = nn.AvgPool2d(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  
###########################################################################

main.py

import torch  
import torch.nn as nn  
import torchvision  
import torchvision.transforms as transforms  
# from vgg16\_full import \*  
from resnet50\_skeleton import \*  
from vgg16\_full import vgg16  
  
device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')  
# device = torch.device('cpu')  
  
  
# Image Preprocessing  
# data augmentation and preprocessing using transform.Compose()  
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=True) # Change Download-flag "True" at the first excution.  
  
test\_dataset = torchvision.datasets.CIFAR10(root='../osproj/data/',  
 train=False,  
 transform=transform\_test)  
  
# load data using 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  
# resnet50  
model = ResNet50\_layer4().to(device)  
PATH = './resnet50\_epoch285.ckpt' # test acc would be almost 80  
  
# vgg16  
# model = vgg16().to(device)  
# PATH = './vgg16\_epoch250.ckpt' # test acc would be almost 85  
####################  
###########################################  
checkpoint = torch.load(PATH, map\_location=torch.device('cpu'))  
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  
# use cross entropy loss and Adam 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() # set model to train mode  
 train\_loss = 0  
  
 # get index and data  
 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() # set gradients to zero  
 loss.backward() # conduct backpropagation  
 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')  
  
# test  
model.eval() # set model to evaluation mode  
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 the test accuracy  
 print('Accuracy of the model on the test images: {} %'.format(100 \* correct / total))

3. Explanation and analysis

resnet50\_skeleton.py 에서 구현한 ResNet-50는 bottle neck building block을 이용했다. 하나의Residual block은 1x1, 3x3, 1x1 conv layer 3개로 구성되어 있고, 이와 같은 residual block들과 다른 layer들이 각 Layer1~Layer4에서 반복되어 총 50개의 layer를 이룬다. Layer1은 7x7x64 의 conv layer와 3x3 max pooling layer, Layer2는 residual block 3개, Layer3은 residual block 4개, Layer4는 residual block 6개로 이루어져 있고, 마지막으로 average pooling layer와 fully connected layer도 있다. 이를 ResidualBlock 클래스와 Linear(), AvgPool2d() 메소드를 이용해 구현한다.

4. Results

텍스트이(가) 표시된 사진

자동 생성된 설명

Test accuracy is 83.29%(it is almost 87%)