Week 8 예습과제

이미지 분류

데이터?

이미지, 텍스트, 오디오 또는 비디오 데이터를 처리해야하는 경우 데이터를 numpy를 이용해 배열로 로드 하는 파이썬 패키지를 사용할 수 있다. (이미지 → pillow, OpenCV // 오디오 → scipy, librosa // 텍스트 → 원시 python, Cython 기반 로딩 혹은 NLTK 및 SpaCy)

이미지 분류기 훈련

1. torchvision을 이용해 CIFAR10 학습 및 test data set을 load하고 regularization한다.

```
import torch
import torchvision
import torchvision.transforms as transforms
```

위 코드를 통해 torchvision으로 CIFAR10을 로드한다.

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

Torchvision data set의 출력은 [0, 1] 범위의 PILImage이다. regularization된 범위 [-1, 1]의 tensor로 변환한다.

```
import matplotlib.pyplot as plt
import numpy as np
```

```
# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```

2. Convolution Neural Network를 정의한다.

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
```

```
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

이전에 신경망 섹션에서 신경망을 복사하고 3 channel image를 사용하도록 수정한다.

3. Loss function을 정의한다.

   Cross Entropy Loss function을 정의하고 momentum, SGD를 사용한다.

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

4. train data set으로 network를 훈련시킨다.

   data를 반복 횟수만큼 반복해서 input으로 넣어주고 optimize한다.

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

print('Finished Training')
```

```
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

학습된 모델을 위와 같이 저장한다.

5. test datad으로 network의 성능을 확인한다.

   train data set에 대해서 2번의 pass에 대해 network를 훈련시켰다. 이 network가 잘 학습했는지 확인하기 위해서 신경망이 출력하는 class lable predict 값과 ground truth를 비교한다.
   이를 위해서 아래의 코드로 test set의 image를 load한다.

```
dataiter = iter(testloader)
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join(f'{classes[labels[j]]:5s}' for j in range(4)))
```


```
net = Net()
net.load_state_dict(torch.load(PATH))
outputs = net(images)
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join(f'{classes[predicted[j]]:5s}'
                              for j in range(4)))
```

저장된 학습 모델을 로드 → 신경망이 예측하는 class lable 출력 → 출력: 10개의 클래스에 대한 에너지, 클래스에 대한 에너지가 높을 수록 network는 image가 특정 class에 부합한다고 여김

```
Predicted:  cat   car   ship  ship
```

```
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

```
Accuracy of the network on the 10000 test images: 54 %
```

완전히 random으로 class를 선택하는 10%의 정확도보다는 높다!

```
# prepare to count predictions for each class
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

# again no gradients needed
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1


# print accuracy for each class
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

```
Accuracy for class: plane is 59.1 %
Accuracy for class: car   is 68.5 %
Accuracy for class: bird  is 30.6 %
Accuracy for class: cat   is 33.5 %
Accuracy for class: deer  is 34.9 %
Accuracy for class: dog   is 54.6 %
Accuracy for class: frog  is 65.9 %
Accuracy for class: horse is 63.8 %
Accuracy for class: ship  is 76.7 %
Accuracy for class: truck is 56.6 %
```

각각의 class마다 정확도가 어떻게 되는지 확인했다.

GPU 학습

Tensor를 GPU로 전송하는 것과 마찬가지로 신경망을 GPU로 전송한다. CUDA를 사용할 수 있는 경우 먼저 장치를 첫 번째로 표시되는 cuda 장치로 정의한다.

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

# Assuming that we are on a CUDA machine, this should print a CUDA device:

print(device)
```

이 섹션의 나머지 부분에서는 이것이 device CUDA 장치라고 가정한다. 그런 다음 이러한 메서드는 모든 모듈을 재귀적으로 살펴보고 매개변수와 버퍼를 CUDA tensor로 변환한다.

```
net.to(device)
```

```
inputs, labels = data[0].to(device), data[1].to(device)
```

모든 단계에서 입력과 목표를 GPU로 보낸다.

CPU에 비해 엄청난 속도 향상을 느끼지 못 한다면 그것은 network의 크기가 작기 때문이다.

Transfer Learning

전이 학습을 이용해 이미지 분류를 위한 합성곱 신경망을 학습시키는 방법에 대한 글이다.

전이 학습의 주요 시나리오 2가지는

- Convolution Neural Network의 “Finetuning”: 무작위 초기화 대신, 신경망을 ImageNet 1000 data set 등으로 미리 학습한 신경망으로 초기화한다. 학습의 나머지 과정들은 평상시와 같다.
- 고정된 Feature Extractor로써의 CNN: 마지막의 FC층들을 제외한 모든 신경망의 가중치를 고정한다. 마지막 부분의 FC 층들은 새로운 무작위의 가중치를 갖는 계층으로 대체되어 이 부분만 학습이 이루어진다.

```
# License: BSD
# Author: Sasank Chilamkurthy

from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy

cudnn.benchmark = True
plt.ion()   # 대화형 모드
```

데이터 불러오기

데이터를 불러올 때는 torchvision과 torch.utils.data 패키지를 사용한다.

이 예제에서는 개미와 벌의 사진을 분류하는 모델을 다룬다.
Train data set에는 120장, Test data set에는 75장의 검증용 이미지가 있다. 일반적으로 맨 처음부터 학습을 한다면 이는 일반화하기에는 아주 작은 데이터셋이지만 전이학습을 진행할 것이기 때문에 일반화를 잘 할 수 있을 것이라 본다.

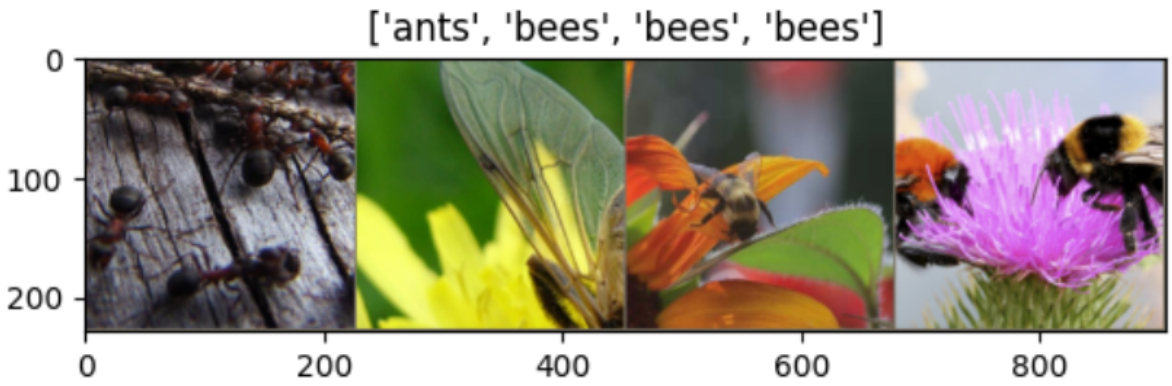
```
# 학습을 위해 데이터 증가(augmentation) 및 일반화(normalization)
# 검증을 위한 일반화
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
```

```
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

data_dir = 'data/hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                              shuffle=True, num_workers=4)
              for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

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

일부 이미지 시각화하기

데이터 증가를 이해하기 위해 일부 학습용 이미지를 시각화한다.

```
def imshow(inp, title=None):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # 갱신이 될 때까지 잠시 기다립니다.


# 학습 데이터의 배치를 얻습니다.
inputs, classes = next(iter(dataloaders['train']))

# 배치로부터 격자 형태의 이미지를 만듭니다.
out = torchvision.utils.make_grid(inputs)

imshow(out, title=[class_names[x] for x in classes])
```



모델 학습하기

학습을 위해서 다음의 것들을 정의한다.

- Learning Rate, scheduling
- 최적의 모델 구하기 (Optimization)

아래 코드에서 scheduler 매개변수는 torch.optim.lr_scheduler의 LR 스케쥴러 객체이다.

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        # 각 에폭(epoch)은 학습 단계와 검증 단계를 갖습니다.
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # 모델을 학습 모드로 설정
            else:
                model.eval()   # 모델을 평가 모드로 설정

            running_loss = 0.0
            running_corrects = 0

            # 데이터를 반복
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # 매개변수 경사도를 0으로 설정
                optimizer.zero_grad()

                # 순전파
                # 학습 시에만 연산 기록을 추적
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
```

```
                    loss = criterion(outputs, labels)

                    # 학습 단계인 경우 역전파 + 최적화
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # 통계
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

            # 모델을 깊은 복사(deep copy)함
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
    print(f'Best val Acc: {best_acc:4f}')

    # 가장 나은 모델 가중치를 불러옴
    model.load_state_dict(best_model_wts)
    return model
```

모델 예측값 시각화하기

```
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloaders['val']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title(f'predicted: {class_names[preds[j]]}')
                imshow(inputs.cpu().data[j])
```

```
                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
```

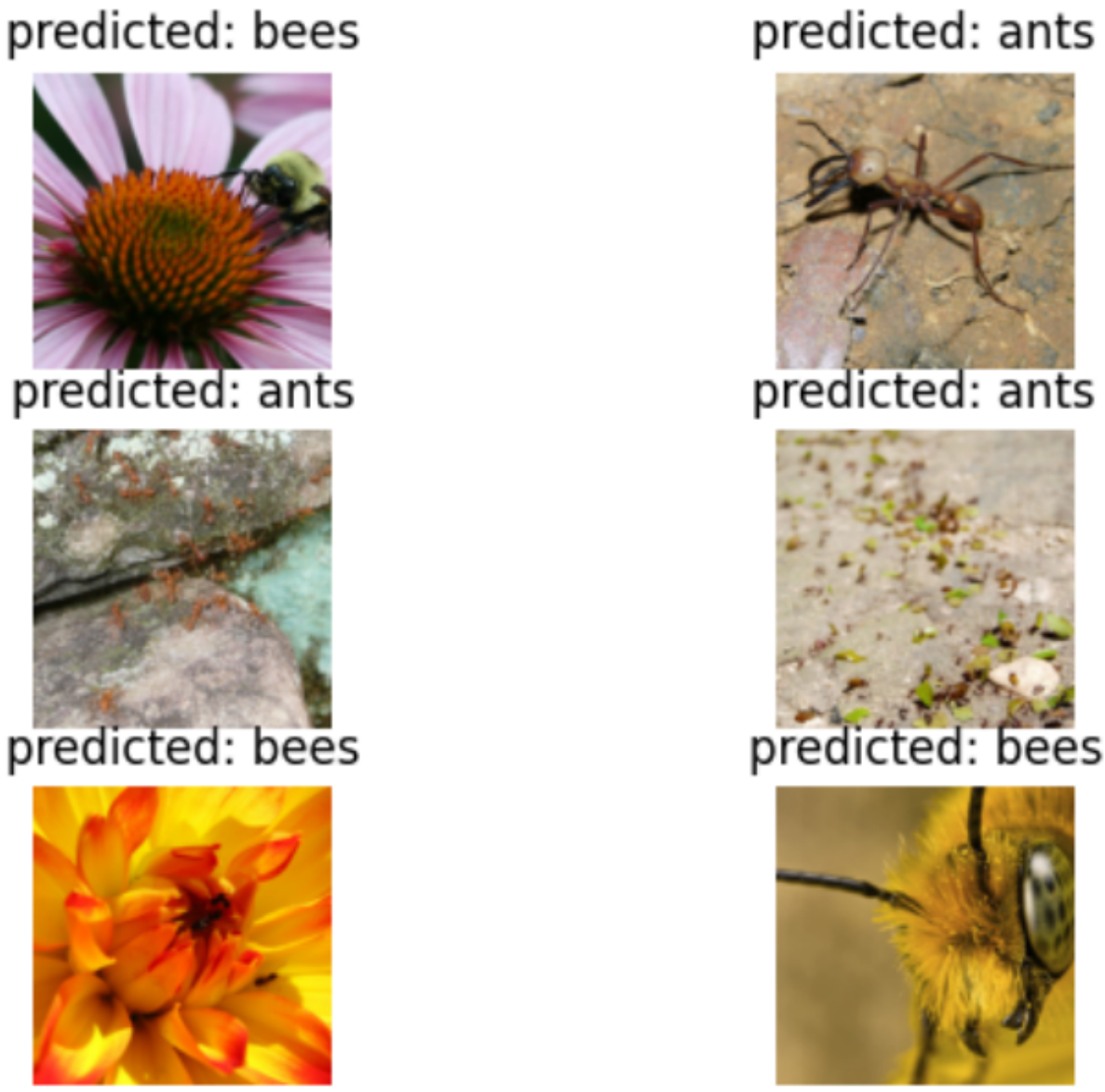
CNN Finetuning

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
# 여기서 각 출력 샘플의 크기는 2로 설정합니다.
# 또는, nn.Linear(num_ftrs, len (class_names))로 일반화할 수 있습니다.
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# 모든 매개변수들이 최적화되었는지 관찰
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# 7 에폭마다 0.1씩 학습률 감소
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

시나리오대로 이미 학습된 모델을 불러온 뒤 마지막의 FC층들을 초기화한다.

학습 및 평가하기

```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=25)
visualize_model(model_ft)
```



고정된 feature extracter로써의 CNN

마지막 계층들을 제외한 신경망의 모든 부분을 고정한다. requires_grad = False로 설정해 매개변수를 고정해 backward() 중에 경사도가 계산되지 않도록 해야한다.

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# 새로 생성된 모듈의 매개변수는 기본값이 requires_grad=True 임
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()
```

```
# 이전과는 다르게 마지막 계층의 매개변수들만 최적화되는지 관찰
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

# 7 에폭마다 0.1씩 학습률 감소
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)
```

학습 및 평가하기

```
model_conv = train_model(model_conv, criterion, optimizer_conv,
                         exp_lr_scheduler, num_epochs=25)
visualize_model(model_conv)

plt.ioff()
plt.show()
```

predicted: ants

predicted: bees

predicted: ants

predicted: bees

predicted: bees

predicted: ants