[PyTorch] TRANSFER LEARNING FOR COMPUTER VISION TUTORIAL

파이토치 전이학습 튜토리얼 ::
https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

딥러닝 모델을 학습시킬 때, 충분한 양의 데이터셋이 있는 경우는 사실 드물기때문에 ImageNet 과 같이 매우 큰 데이터셋으로 합성곱신경망(ConvNet)을 미리 학습을 한 뒤에 내가 원하는 클래스에 대해 finetuning 해주는 경우가 대부분이다.

- **finetuning**: 무작위 초기화 대신 ImageNet 1000과 같은 데이터셋으로 미리 학습한 신경망으로 초기화한다.
- **ConvNet as fixed feature extractor**: 마지막 FC layer를 제외한 모든 신경망의 가중치를 freeze시킨다. 마지막 FC layer는 새로운 랜덤 가중치로 초기화하여 이 레이어만 학습시킨다.

라이브러리 불러오기

```
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()   # interactive mode
```

Load Data

📌 Goal: ants와 bees 구분하기

- ImageNet의 일부로, 각 클래스에 대해 120장의 training 이미지와 75장의 validation 이미지가 있음.
- 학습용 데이터는 data augmentation과 정규화를 해주고, 검증용 데이터는 정규화만 진행한다.
- 이미지 크기는 224 x 224이다
- dataloader는 ‘train’과 ‘val’ phase를 가짐

```
# Data augmentation and normalization for training
# Just normalization for validation
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")
```

이미지 확인하기

```
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)  # pause a bit so that plots are updated


# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs)

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

텐서 이미지를 다시 넘파이로 변화해주고, plt.imshow()를 이용하여 이미지 확인

(이미지가 너무 징그러워서 출력 결과 안 넣음 ㅜㅜ)

Training the model

- Scheduling Learning Rate
- Saving the best model

```
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)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                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}')

            # deep copy the model
```

```
            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}')

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

Learning Rate Scheduler

`torch.optim.lr_scheduler` : epoch 수에 따라 LR 조정

- Learning Rate Decay: 처음 시작시 Learning Rate 값을 크게 준 후 일정 epoch 마다 값을 감소시켜서 최적의 학습까지 더 빠르게 도달할 수 있게 하는 방법
- Step Decay: 특정 epoch 구간(step) 마다 일정한 비율로 감소 시켜주는 방법
- epoch마다 scheduler.step()를 해줌

Visualizing the model predictions

일부 이미지에 대한 예측값 보여주기

- backward가 필요없으므로 with torch.no_grad() 사용

```
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)
```

Finetuning the ConvNet

pretrained 모델을 로드한 후, 마지막 FC layer만 초기화 해줌

- model: resnet18
- torchvision.models: 다양한 이미지 task(classification, object detection 등)의 pretrained model 제공 (ImageNet 기반)
- class 수 = 2(ants, bees) 이므로 FC layer 출력 크기를 2로 설정해줌

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
# 각 출력 샘플의 크기는 2(since #_class is two).
# 또는 nn.Linear(num_ftrs, len(class_names))로 일반화 가능.
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

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

Train and evaluate

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

ConvNet as fixed feature extractor

전이학습을 할 때 어느 레이어까지 학습할 것인가를 정해준다.

→ 참고하면 좋을 링크: https://cs231n.github.io/transfer-learning/ , https://analysisbugs.tistory.com/103

마지막 층을 제외한 모든 부분을 freeze 시켜준다. **requires_grad = False** 로 설정하여 backward 중에 gradient가 계산되지 않도록한다.

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

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as
# opposed to before.
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)
```

Train and Evaluate

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

backward 연산 없이 forward만 하기때문에 이전보다 학습 및 평가 시간이 적게 걸린다.