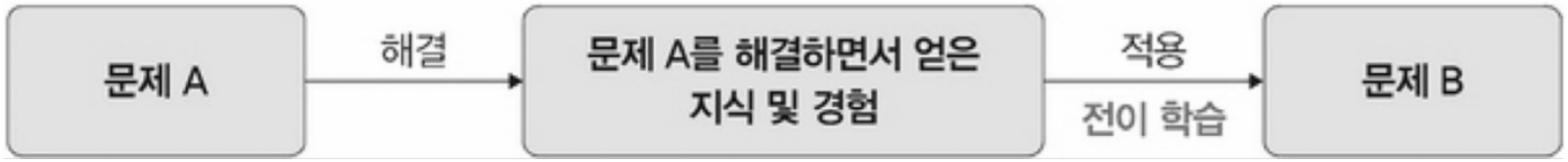
Transfer Learning with Pytorch

Transfer learning

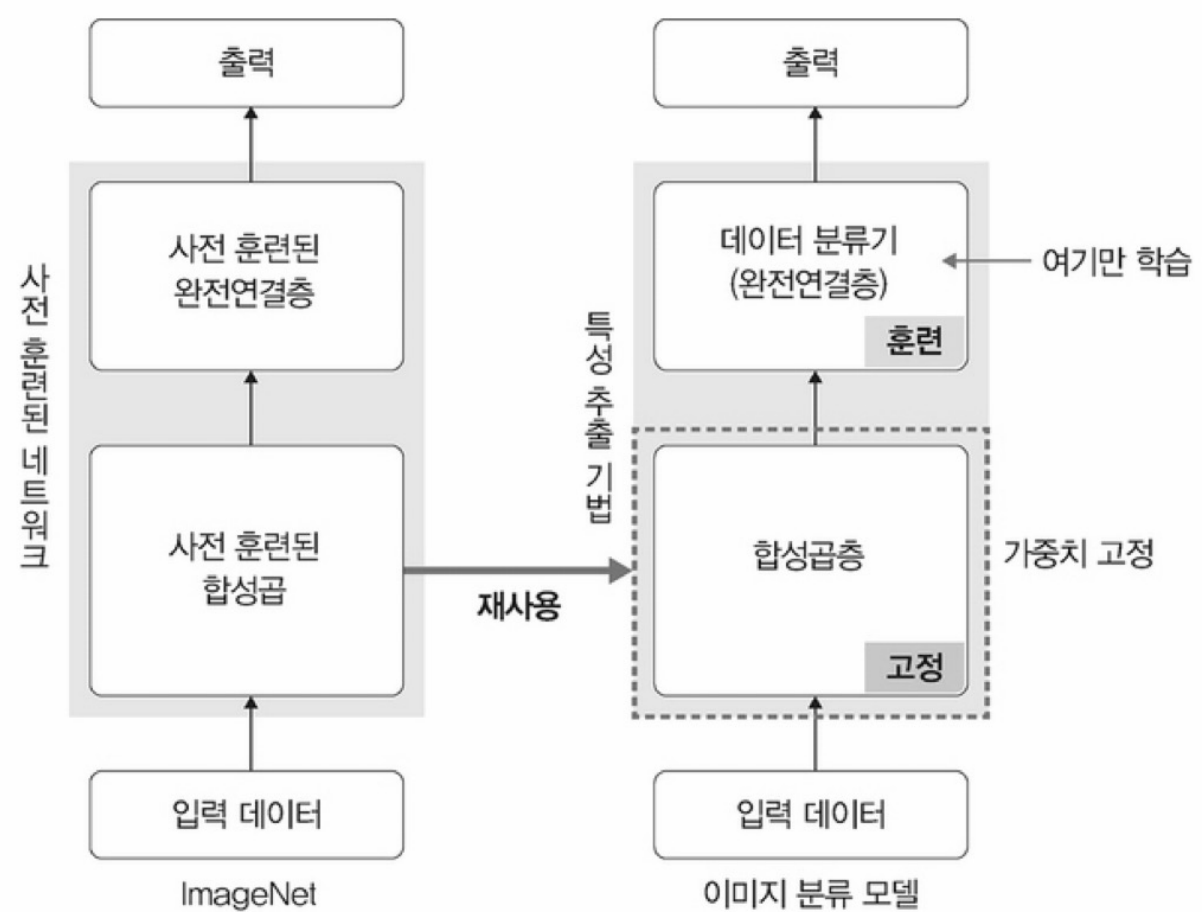
- 데이터 구축은 고비용 (시간 + 돈!)
  - 이미 구축된 데이터를 활용하여 특징을 학습한 후 활용
  - 예: 이미지넷 (120만개의 이미지에 대하여 학습)을 활용하여 특징 학습후에 적은 규모의 데이터셋(e.g. FashionMNIST)에 적용



Transfer learning

- 주로 세가지 전략 사용
  - ConvNet as fixed feature extractor : 마지막 FC 레이어만 학습
  - Fine-tuning the ConvNet : 전체 네트워크를 재학습 (소규모 데이터로..)
  - Pretrained models : 다른 연구자가 공개한 사전학습 모델 활용

Transfer learning



- **고정된 특징 추출기로써의 합성곱 신경망**

ImageNet 데이터셋으로 사전 훈련된 모델을 가져온 후 마지막에 완전연결층 부분만 새로 만듬.

- 합성곱층

:합성곱층과 풀링층으로 구성

- 데이터 분류기(완전연결층)

:추출된 특성을 입력받아 최종적으로 이미지에 대한 클래스를 분류하는 부분

사전 훈련된 네트워크의 합성곱층(가중치 고정)에 새로운 데이터를 통과시키고, 그 출력을 데이터 분류기에서 훈련시킵니다.

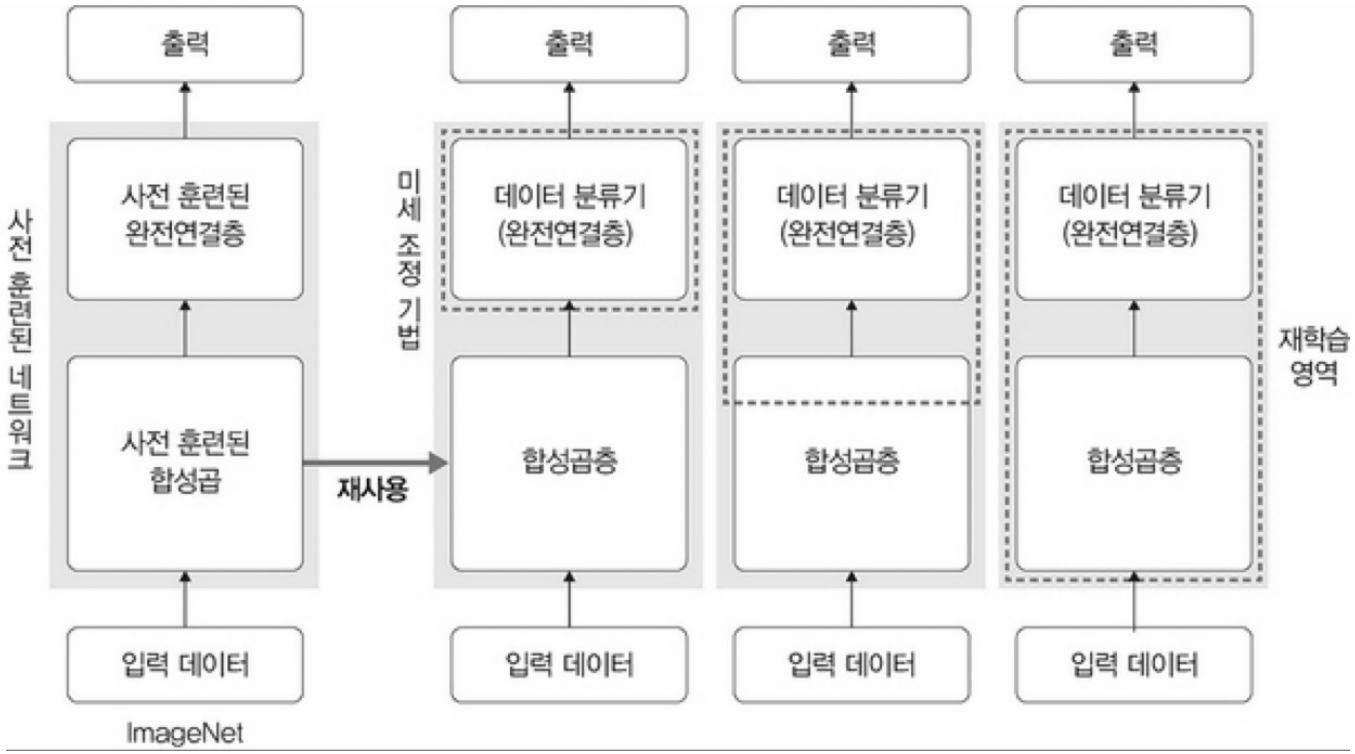
- LeNet-5
- Inception V3.
- ResNet50.
- AlexNet
- VGGNet
- MobileNet

Transfer learning

- **합성곱 신경망의 미세조정(finetuning)**

사전 훈련된 네트워크를 미세 조정하여 분석하려는 데이터셋에 잘 맞도록 모델의 파라미터를 조정하는 기법

- **데이터셋이 크고 사전 훈련된 모델과 유사성이 작을 경우**
- 모델 전체를 재학습
- 데이터셋 크기가 크기 때문에 재학습시키는 것이 좋은 전략
- **데이터셋이 크고 사전 훈련된 모델과 유사성이 클 경우**
- 합성곱층의 뒷부분(완전연결층과 가까운 부분)과 데이터 분류기를 학습
- 데이터셋이 유사하기 때문에 전체를 학습시키는 것보다는 강한 특징이 나타나는 합성곱층의 뒷부분과 데이터 분류기만 새로 학습하더라도 최적의 성능을 낼 수 있음
- **데이터셋이 작고 사전 훈련된 모델과 유사성이 작을 경우**:-
- 합성곱층의 일부분과 데이터 분류기를 학습
- 데이터가 적기 때문에 일부 계층에 미세 조정 기법을 적용한다고 해도 효과가 없을 수 있음
- **데이터셋이 작고 사전 훈련된 모델과 유사성이 클 경우**
- 데이터 분류기만 학습
- 데이터가 적기 때문에 많은 계층에 미세 조정 기법을 적용하면 과적합이 발생할 수 있음



Pytorch Practice – Transfer Learning

개미 vs 벌 분류 문제

```
Hymenoptera_data
    ㄴ train/
        ㄴ ants/
        ㄴ bees/
    ㄴ val/
        ㄴ ants/
        ㄴ bees/
```

Pytorch Practice – Transfer Learning

```
model_ft = models.resnet18(pretrained=True)
```

Pytorch Practice – Transfer Learning

1. Training 및 Test 데이터셋을 불러오고, 정규화하기

```
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
```

Pytorch Practice – Transfer Learning

1. Training 및 Test 데이터셋을 불러오고, 정규화하기

```
# 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])
    ]),
}
```

Pytorch Practice – Transfer Learning

1. Training 및 Test 데이터셋을 불러오고, 정규화하기

```
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")
```

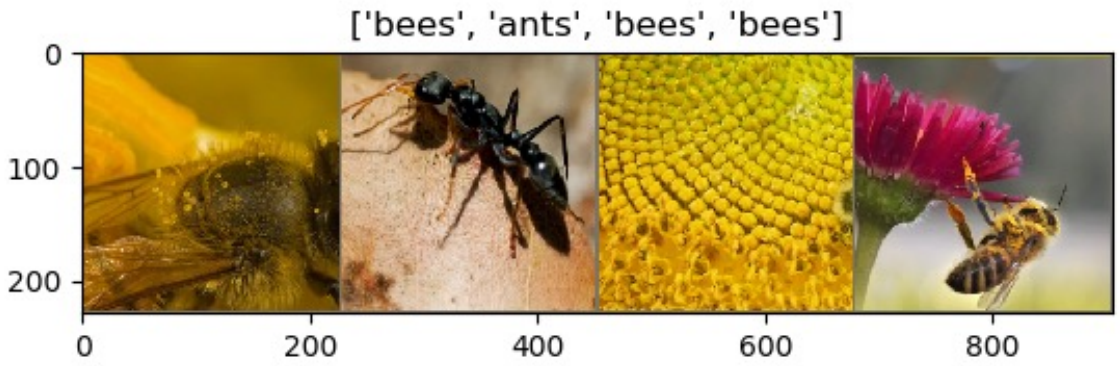
Pytorch Practice – Transfer Learning

* 일부 이미지 시각화하기

```
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])
```

Pytorch Practice – Transfer Learning

* 일부 이미지 시각화하기



Pytorch Practice – Transfer Learning

2. 모델 정의하기

```
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)
```

Pytorch Practice – Transfer Learning

3. 손실함수와 Optimizer 정의하기

```
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)
```

Pytorch Practice – Transfer Learning

4. Training set을 사용하여 신경망 학습하기

```
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()
```

Pytorch Practice – Transfer Learning

4. Training set을 사용하여 신경망 학습하기

```
                # 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
```

Pytorch Practice – Transfer Learning

5. Test set을 사용하여 신경망이 잘 훈련했는지를 검사하기

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

Pytorch Practice – Transfer Learning

5. Test set을 사용하여 신경망이 잘 훈련했는지를 검사하기

Out:

```
Epoch 0/24
----------
train Loss: 0.5926 Acc: 0.6639
val Loss: 0.2288 Acc: 0.9412

Epoch 1/24
----------
train Loss: 0.4602 Acc: 0.8238
val Loss: 0.2509 Acc: 0.9020
```

```
Epoch 24/24
----------
train Loss: 0.3253 Acc: 0.8648
val Loss: 0.2031 Acc: 0.9477

Training complete in 0m 32s
Best val Acc: 0.954248
```

Test_code

- https://tutorials.pytorch.kr/beginner/transfer_learning_tutorial.html