다섯째마당
딥러닝 활용하기

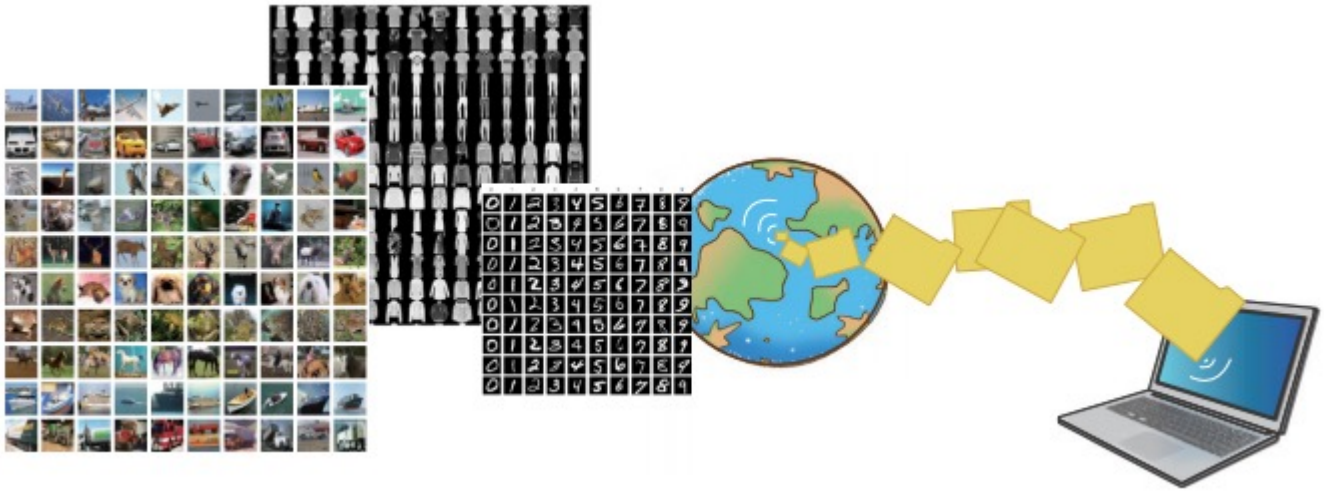
20장 전이 학습을 통해 딥러닝의 성능 극대화하기

1 소규모 데이터셋으로 만드는 강력한 학습 모델

2 전이 학습으로 모델 성능 극대화하기

전이 학습을 통해 딥러닝의 성능 극대화하기

- 전이 학습을 통해 딥러닝의 성능 극대화하기



전이 학습을 통해 딥러닝의 성능 극대화하기

- **전이 학습을 통해 딥러닝의 성능 극대화하기**
  - 딥러닝으로 좋은 성과를 내려면 딥러닝 프로그램을 잘 짜는 것도 중요하지만, 딥러닝에 입력할 데이터를 모으는 것이 더 중요함
  - 기존 머신 러닝과 달리 딥러닝은 스스로 중요한 속성을 뽑아 학습하기 때문에 비교적 많은 양의 데이터가 필요함
  - 데이터가 충분하지 않은 상황도 발생
  - 이 장에서는 나만의 프로젝트를 기획하고 실습하는 과정을 따라해 보며, 딥러닝의 데이터양이 충분하지 않을 때 활용할 수 있는 방법들을 배우겠음
  - 여러 방법 중에서 수만 장에 달하는 기존의 이미지에서 학습한 정보를 가져와 내 프로젝트에 활용하는 것을 **전이 학습**(transfer learning)이라고 함
  - 방대한 자료를 통해 미리 학습한 가중치(weight) 값을 가져와 내 프로젝트에 사용하는 방법으로 컴퓨터 비전, 자연어 처리 등 다양한 분야에서 전이 학습을 적용해 예측률을 높이고 있음

1 소규모 데이터셋으로 만드는 강력한 학습 모델

1 소규모 데이터셋으로 만드는 강력한 학습 모델

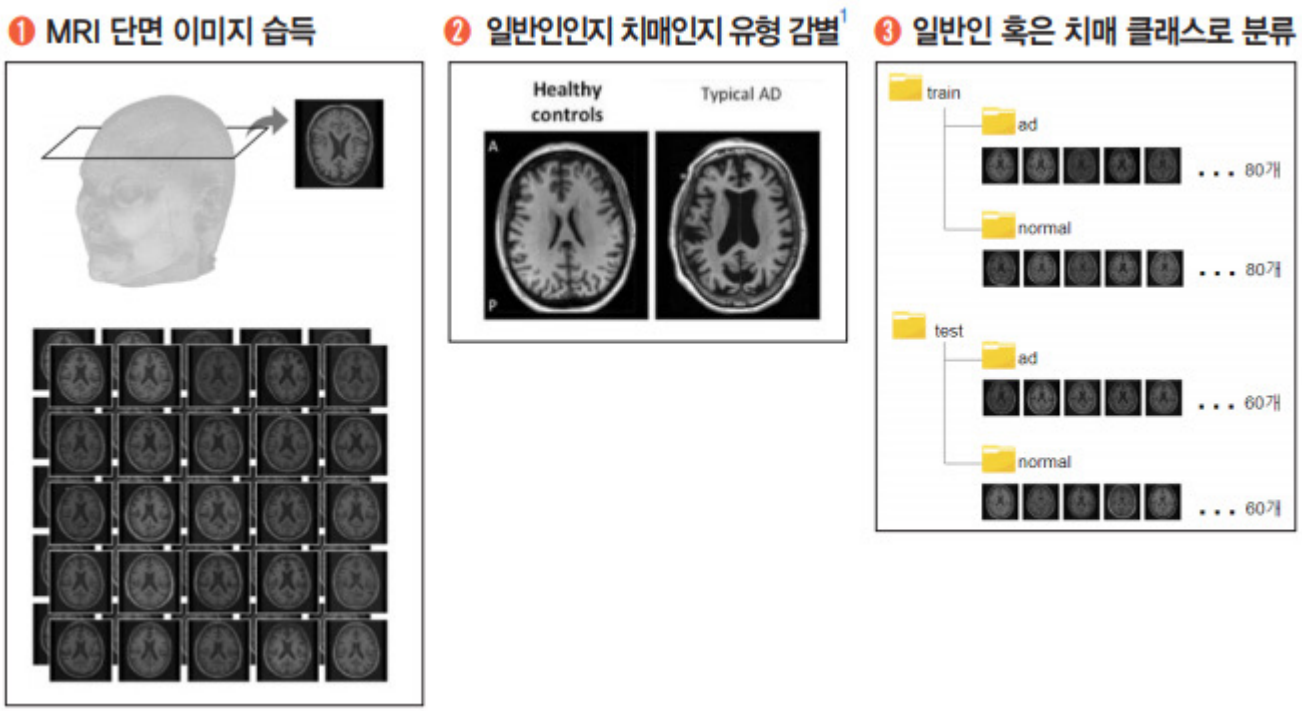
- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 딥러닝을 이용한 프로젝트는 어떤 데이터를 가지고 있는지, 어떤 목적을 가지고 있는지 잘 살펴보는 것부터 시작
  - 내가 가진 데이터에 따라 딥러닝 알고리즘을 결정해야 하는데, 딥러닝 및 머신러닝 알고리즘은 크게 두 가지 유형으로 나뉨
  - 정답을 알려 주고 시작하는가 아닌가에 따라 **지도 학습**(supervised learning) 방식과 **비지도 학습**(unsupervised learning) 방식으로 구분
  - 지금까지 이 책에서 살펴본 폐암 수술 환자의 생존율 예측, 피마 인디언의 당뇨병 예측, CNN을 이용한 MNIST 분류 등은 각 데이터 또는 사진마다 '클래스'라는 정답을 주고 시작
  - 모두 '지도 학습'의 예가 됨
  - 반면 19장에서 배운 GAN이나 오토인코더는 정답을 예측하는 것이 아니라 주어진 데이터의 특성을 찾았기 때문에 '비지도 학습'의 예가 됨

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 이번에 진행할 프로젝트는 MRI 뇌 사진을 보고 치매 환자의 뇌인지, 일반인의 뇌인지 예측하는 것
  - 각 사진마다 치매 혹은 일반인으로 클래스가 주어지므로 지도 학습의 예라고 할 수 있음
  - 이미지를 분류할 것이므로 이미지 분류의 대표적인 알고리즘인 컨볼루션 신경망(CNN)을 선택해 진행

1 소규모 데이터셋으로 만드는 강력한 학습 모델

▼ 그림 20-1 | MRI 뇌 사진 데이터의 구성



1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 총 280장으로 이루어진 뇌의 단면 사진
  - 치매 환자의 특성을 보이는 뇌 사진 140장과 일반인의 뇌 사진 140장으로 구성되어 있음
  - 280개의 이미지 중 160개는 train 폴더에, 120개는 test 폴더에 넣어 두었음
  - 각 폴더 밑에는 ad와 normal이라는 두 개의 폴더가 있는데, 치매 환자의 뇌 사진은 ad 폴더에, 일반인의 뇌 사진은 normal 폴더에 저장
  - 데이터의 습득 과정과 구성을 잘 보여 주는 그림이라고 할 수 있음

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 앞서 MNIST 손글씨나 로이터 뉴스, 영화 리뷰의 예제들과는 다르게 케라스에서 제공하는 데이터를 불러오는 것이 아니라, 내 데이터를 읽어 오는 것이므로 새로운 함수가 필요함
  - 데이터의 수를 늘리는 ImageDataGenerator() 함수와 폴더에 저장된 데이터를 불러오는 flow_from_directory() 함수를 사용

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - ImageDataGenerator() 함수는 주어진 데이터를 이용해 변형된 이미지를 만들어 학습셋에 포함시키는 편리한 기능을 제공
  - 이미지 데이터의 수를 확장할 때 효과적으로 사용할 수 있음
  - 다음은 함수를 사용한 예

```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   horizontal_flip=True,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   rotation_range=5,
                                   shear_range=0.7,
                                   zoom_range=1.2,
                                   vertical_flip=True,
                                   fill_mode='nearest')
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 각각 인자에 대한 설명은 다음과 같음(그림 20-2에도 정리되어 있음)

    - rescale: 주어진 이미지의 크기를 바꾸어 줌

      예를 들어 원본 영상이 0~255의 RGB값을 가지고 있으므로 255로 나누면 0~1의 값으로

      변환되어 학습이 좀 더 빠르고 쉬워짐
    - 앞서 배운 정규화 과정과 같음
    - horizontal_flip, vertical_flip: 주어진 이미지를 수평 또는 수직으로 뒤집음

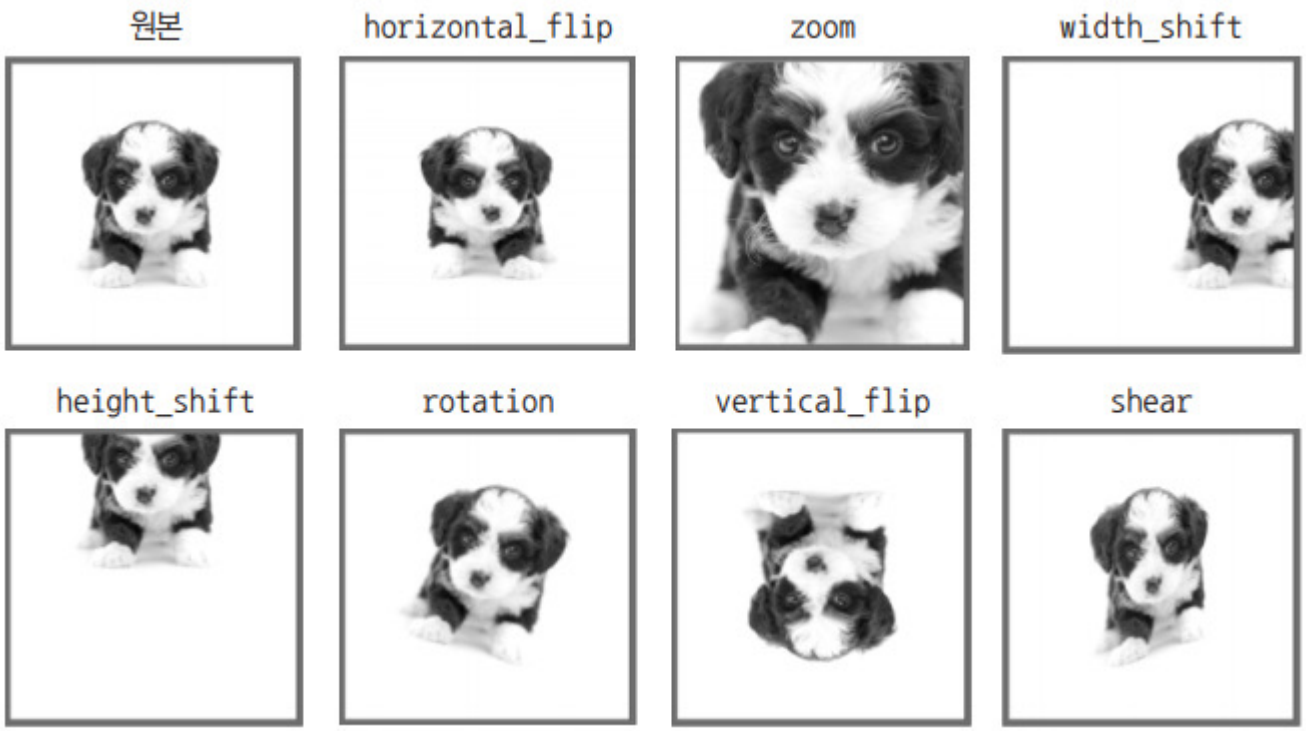
      zoom_range: 정해진 범위 안에서 축소 또는 확대

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - width_shift_range, height_shift_range: 정해진 범위 안에서 그림을 수평 또는 수직으로
    랜덤하게 평행 이동시킴
  - rotation_range: 정해진 각도만큼 이미지를 회전시킴
  - shear_range: 좌표 하나를 고정시키고 다른 몇 개의 좌표를 이동시키는 변환
    fill_mode: 이미지를 축소 또는 회전하거나 이동할 때 생기는 빈 공간을 어떻게 채울지 결정
    nearest 옵션을 선택하면 가장 비슷한 색으로 채워짐

1 소규모 데이터셋으로 만드는 강력한 학습 모델

▼ 그림 20-2 | ImageDataGenerator 옵션의 결과



1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 단, 이 모든 인자를 다 적용하면 불필요한 데이터를 만들게 되어 오히려 학습 시간이 늘어난다는 것에 주의해야 함
  - 주어진 데이터의 특성을 잘 파악한 후 이에 맞게 사용하는 것이 좋음
  - 우리는 좌우의 차이가 그다지 중요하지 않은 뇌 사진을 이용할 것이므로 수평으로 대칭시키는 horizontal_flip 인자를 사용
  - width_shift, height_shift 인자를 이용해 조금씩 좌우로 수평 이동시킨 이미지도 만들어 사용
  - 참고로 데이터 부풀리기는 학습셋에만 적용하는 것이 좋음
  - 테스트셋은 실제 정보를 그대로 유지하게 하는 편이 과적합의 위험을 줄일 수 있기 때문임

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 테스트셋은 다음과 같이 정규화만 진행해 줌

```
test_datagen = ImageDataGenerator(rescale=1./255)
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 이미지 생성 옵션을 정하고 나면 실제 데이터가 있는 곳을 알려 주고 이미지를 불러오는 작업을 해야 함
  - 이를 위해 flow_from_directory() 함수를 사용

```
train_generator = train_datagen.flow_from_directory(
    './data-ch20/train',   # 학습셋이 있는 폴더 위치
    target_size=(150,150), # 이미지 크기
    batch_size=5,
    class_mode='binary')   # 치매/정상 이진 분류이므로 바이너리 모드로 실행
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 같은 과정을 거쳐서 테스트셋도 생성

```
test_generator = test_datagen.flow_from_directory(
    './data-ch20/test'      # 테스트셋이 있는 폴더 위치
    target_size=(150,150),
    batch_size=5,
    class_mode='binary')
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 모델 실행을 위한 옵션을 만들어 줌
  - 옵티마이저로 Adam을 선택하는데, 이번에는 케라스 ❶의 optimizers 클래스를 이용해 학습률을 따로 지정해 보았음
  - 조기 중단을 설정하고 model.fit()을 실행하는데, 이때 학습셋과 검증셋을 조금 전 만들어 준

❷ ❸

```
# 모델의 실행 옵션을 설정합니다.
model.compile(loss='binary_crossentropy', optimizer=optimizers.Adam
(learning_rate=0.0002), metrics=['accuracy']) ····❶

# 학습의 조기 중단을 설정합니다.
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=5)

# 모델을 실행합니다.
history = model.fit(train_generator, ····❷
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 소규모 데이터셋으로 만드는 강력한 학습 모델

```
    epochs=100,
    validation_data=test_generator, ····❸
    validation_steps=10,
    callbacks=[early_stopping_callback])
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 이제 CNN을 이용해 모델을 만들겠음
  - 이 실습에는 사이파이(SciPy) 라이브러리가 필요함
  - 코랩의 경우에는 기본으로 제공하지만, 주피터 노트북을 이용해 실습 중이라면 다음 명령으로 라이브러리를 설치

    !pip install Scipy

실습 | 치매 환자의 뇌인지 일반인의 뇌인지 예측하기

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, Flatten,
Conv2D, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import optimizers
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 소규모 데이터셋으로 만드는 강력한 학습 모델

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

# 깃허브에 준비된 데이터를 가져옵니다.
!git clone https://github.com/taehojo/data-ch20.git

# 학습셋의 변형을 설정하는 부분입니다.
train_datagen = ImageDataGenerator(rescale=1./255, # 주어진 이미지 크기를 설정
    horizontal_flip=True,   # 수평 대칭 이미지를 50% 확률로 만들어 추가합니다.
    width_shift_range=0.1,  # 전체 크기의 15% 범위에서 좌우로 이동합니다.
    height_shift_range=0.1, # 마찬가지로 위아래로 이동합니다.
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 소규모 데이터셋으로 만드는 강력한 학습 모델

```
    #rotation_range=5,      # 정해진 각도만큼 회전시킵니다.
    #shear_range=0.7,       # 좌표 하나를 고정시키고 나머지를 이동시킵니다.
    #zoom_range=[0.9, 2.2], # 확대 또는 축소시킵니다.
    #vertical_flip=True,    # 수직 대칭 이미지를 만듭니다.
    #fill_mode='nearest'    # 빈 공간을 채우는 방법입니다.
                            # nearest 옵션은 가장 비슷한 색으로 채우게 됩니다.
    )

train_generator = train_datagen.flow_from_directory(
    './data-ch20/train',    # 학습셋이 있는 폴더의 위치입니다.
    target_size=(150,150),
    batch_size=5,
    class_mode='binary')
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 소규모 데이터셋으로 만드는 강력한 학습 모델

```
# 테스트셋은 이미지 부풀리기 과정을 진행하지 않습니다.
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
    './data-ch20/test',     # 테스트셋이 있는 폴더의 위치입니다.
    target_size=(150,150),
    batch_size=5,
    class_mode='binary')

# 앞서 배운 CNN 모델을 만들어 적용해 보겠습니다.
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(150,150,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 소규모 데이터셋으로 만드는 강력한 학습 모델

```
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 소규모 데이터셋으로 만드는 강력한 학습 모델

```
# 모델 실행의 옵션을 설정합니다.
model.compile(loss='binary_crossentropy', optimizer=optimizers.Adam
(learning_rate=0.0002), metrics=['accuracy'])

# 학습의 조기 중단을 설정합니다.
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=5)

# 모델을 실행합니다.
history = model.fit(
    train_generator,
    epochs=100,
    validation_data=test_generator,
    validation_steps=10,
    callbacks=[early_stopping_callback])
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 소규모 데이터셋으로 만드는 강력한 학습 모델

```
# 검증셋과 학습셋의 오차를 저장합니다.
y_vloss = history.history['val_loss']
y_loss = history.history['loss']

# 그래프로 표현해 봅니다.
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

# 그래프에 그리드를 주고 레이블을 표시하겠습니다.
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 소규모 데이터셋으로 만드는 강력한 학습 모델

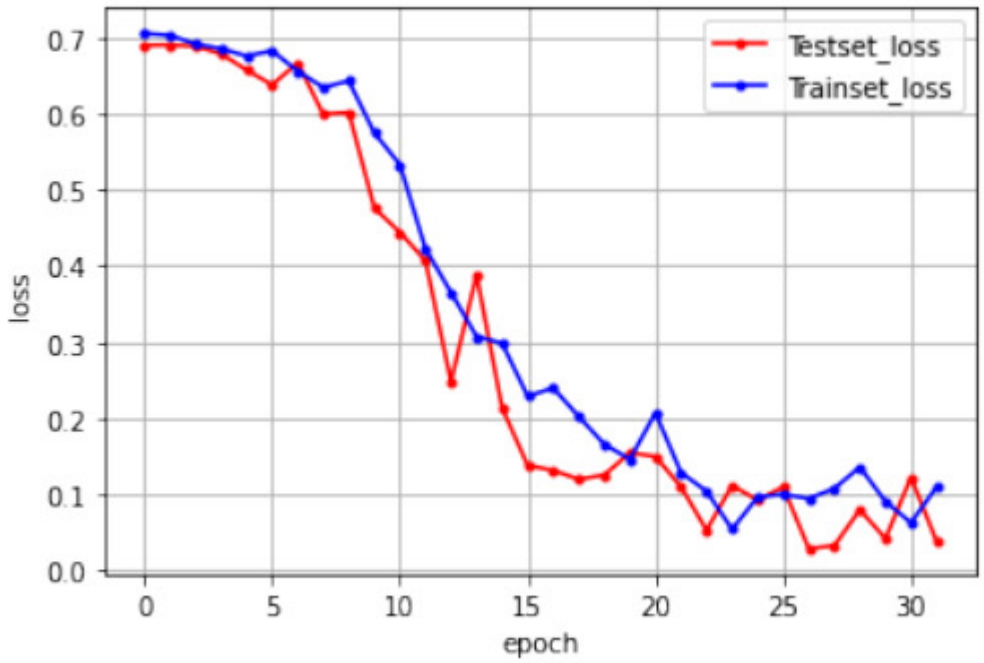
실행 결과

```
Epoch 1/100
32/32 [==============================] - 3s 68ms/step - loss: 0.7053 -
accuracy: 0.5063 - val_loss: 0.6896 - val_accuracy: 0.5000
... (중략) ...
Epoch 32/100
32/32 [==============================] - 2s 65ms/step - loss: 0.1099 -
accuracy: 0.9563 - val_loss: 0.0388 - val_accuracy: 1.0000
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 소규모 데이터셋으로 만드는 강력한 학습 모델

▼ 그림 20-3 | 그림으로 확인하는 학습 결과



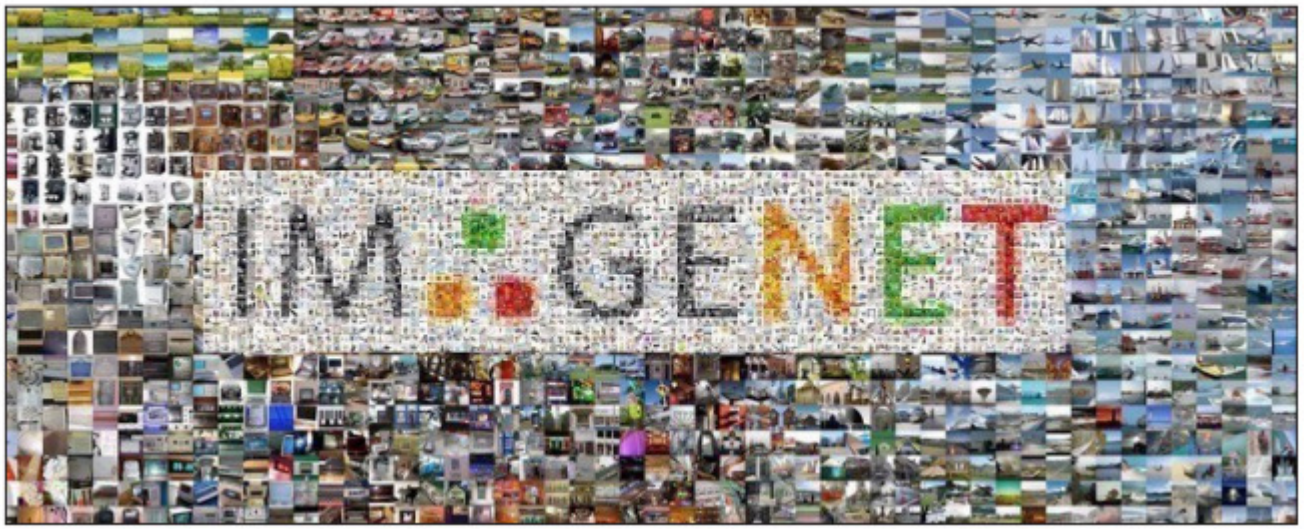
1 소규모 데이터셋으로 만드는 강력한 학습 모델

- **소규모 데이터셋으로 만드는 강력한 학습 모델**
  - 첫 정확도 50%에서 출발해 32번째 에포크에서 100%의 정확도를 보이며 멈추었음
  - 그래프를 통해 완만하게 하강하는 오차 곡선들을 볼 수 있음

2 전이 학습으로 모델 성능 극대화하기

2 전이 학습으로 모델 성능 극대화하기

▼ 그림 20-4 | 이미지넷 데이터셋에서 추출한 사진들



2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - 사진은 이미지넷(ImageNet) 데이터셋에서 추출한 사진들로 구성된 것
  - 이미지넷은 1,000가지 종류로 나뉜 120만 개가 넘는 이미지를 놓고 어떤 물체인지 맞히는 '이미지넷 이미지 인식 대회(ILSVRC)'에 사용되는 데이터셋
  - MNIST와 더불어 가장 유명한 데이터셋 중 하나
  - 전체 크기가 200GB에 이를 만큼 커다란 이 데이터를 놓고 그동안 수많은 그룹이 경쟁하며 최고의 분류기를 만들기 위해 노력해 왔음

2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - 치매/일반인 뇌 사진 분류 프로젝트를 하고 있는 우리에게도 이 자료가 중요한 이유는 지금부터 이 방대한 양의 데이터셋에서 추출한 정보를 가져와서 우리 예측률을 극대화하는 '전이 학습'을 할 것이기 때문임
  - 전이 학습은 앞서 잠깐 언급한 대로 '기존의 학습 결과를 가져와서 유사한 프로젝트에 사용하는 방법'을 의미
  - 뇌 사진만 다루는 치매 분류기를 만드는데, 뇌 사진과 관련 없는 수백만 장의 이미지넷 학습 정보가 큰 역할을 하는 이유는 '형태'를 구분하는 기본적인 학습이 되어 있기 때문임

2 전이 학습으로 모델 성능 극대화하기

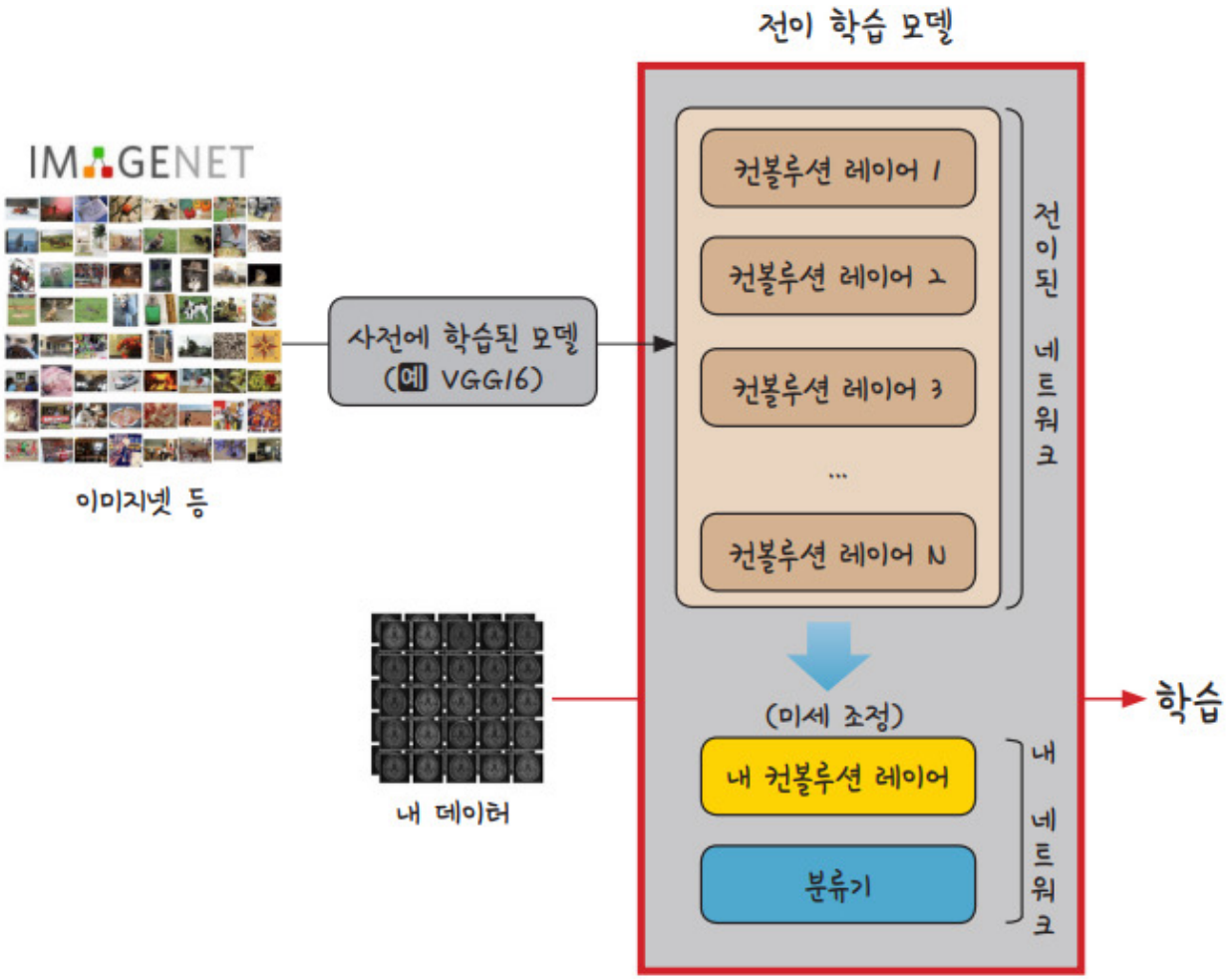
- **전이 학습으로 모델 성능 극대화하기**
  - 예를 들어 딥러닝은 학습이 시작되면 어떤 픽셀의 조합이 '선'이고 어떤 형태의 그룹이 '면'이 되는지부터 파악해야 함
  - 아무런 정보도 없이 MRI 사진 판별을 시작한다면 이러한 기본적인 정보를 얻어내는 데도 많은 시간을 쏟아야 함
  - 전이 학습이 해결해 주는 것이 바로 이 부분
  - 대용량의 데이터를 이용해 학습한 가중치 정보를 가져와 내 모델에 적용한 후 프로젝트를 계속해서 진행할 수 있는 것

2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - 전이 학습을 적용하는 방법은 그림 20-5와 같음
  - 먼저 대규모 데이터셋에서 학습된 기존의 네트워크를 불러옴
  - CNN 모델의 앞쪽을 이 네트워크로 채움
  - 뒤쪽 레이어에서 내 프로젝트와 연결
  - 이 두 네트워크가 잘 맞물리게끔 미세 조정(fine tuning)을 하면 됨

2 전이 학습으로 모델 성능 극대화하기

▼ 그림 20-5 | 전이 학습의 구조



2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - 이제 앞서 우리가 만든 모델에 이미지넷 데이터셋에서 미리 학습된 모델인 VGGNet을 가지고 오는 예제를 실행해 보자
  - VGGNet은 옥스포드 대학의 연구 팀 VGG에 의해 개발된 모델로, 2014년 이미지넷 이미지 인식 대회에서 2위를 차지한 모델
  - 학습 구조에 따라 VGG16, VGG19 등 이름이 주어졌는데, 우리는 VGG16을 사용

2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - VGG 외에도 ResNet, Inception, MobileNet, DenseNet 등 많은 모델을 불러올 수 있음
  - 각 네트워크에 대한 상세한 설명은 케라스 공식 사이트를 참조 (https://keras.io/applications/)

2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - 다음은 모델 이름을 transfer_model로 정하고 VGG16을 불러온 모습
  - include_top은 전체 VGG16의 마지막 층, 즉 분류를 담당하는 곳을 불러올지 말지를 정하는 옵션
  - 우리가 만든 로컬 네트워크를 연결할 것이므로 False로 설정
  - 또한, 불러올 부분은 새롭게 학습되는 것이 아니므로 학습되지 않도록 transfer_model.trainable 옵션 역시 False로 설정

```
transfer_model = VGG16(weights='imagenet', include_top=False, input_
shape=(150,150,3))
transfer_model.trainable = False
transfer_model.summary()
```

2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - transfer_model.summary() 함수를 통해 학습 구조를 보면 다음과 같음

실행 결과

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
```

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

```
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
```

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

```
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
```

2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - 학습 가능한 파라미터(Trainable params)가 없음을 확인
  - 이제 우리의 로컬 네트워크를 다음과 같이 만들어 줌

```
finetune_model = models.Sequential()
finetune_model.add(transfer_model)
finetune_model.add(Flatten())
finetune_model.add(Dense(64))
finetune_model.add(Activation('relu'))
finetune_model.add(Dropout(0.5))
finetune_model.add(Dense(1))
finetune_model.add(Activation('sigmoid'))
finetune_model.summary()
```

2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - finetune_model이라는 이름의 모델을 만들었음
  - 위와 같이 첫 번째 층은 앞서 불러온 transfer_model을 그대로 불러온 후 최종 예측하는 층을 추가하면 됨
  - 다음은 finetune_model.summary() 함수를 통해 학습 구조를 살펴본 결과

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 64)                524352
_________________________________________________________________
```

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

```
activation (Activation)      (None, 64)                0
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 15,239,105
Trainable params: 524,417
Non-trainable params: 14,714,688
_________________________________________________________________
```

2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - 앞서 넘겨받은 파라미터들(14,714,688)을 그대로 유지한 채 최종 분류를 위해서만 새롭게 학습하는 것을 알 수 있음
  - 코드를 종합하면 다음과 같음

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Input, models, layers, optimizers, metrics
from tensorflow.keras.layers import Dense, Flatten, Activation, Dropout
from tensorflow.keras.applications import VGG16
from tensorflow.keras.callbacks import EarlyStopping

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

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

```
# 깃허브에 준비된 데이터를 가져옵니다.
!git clone https://github.com/taehojo/data-ch20.git

# 학습셋의 변형을 설정하는 부분입니다.
train_datagen = ImageDataGenerator(rescale=1./255,
                                   horizontal_flip=True,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                  )

train_generator = train_datagen.flow_from_directory(
    './data-ch20/train',
    target_size=(150,150),
    batch_size=5,
    class_mode='binary')
```

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

```
# 테스트셋의 정규화를 설정합니다.
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
    './data-ch20/test',
    target_size=(150,150),
    batch_size=5,
    class_mode='binary')

# VGG16 모델을 불러옵니다.
transfer_model = VGG16(weights='imagenet', include_top=False, input_
shape=(150,150,3))
transfer_model.trainable = False
```

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

```
# 우리의 모델을 설정합니다.
finetune_model = models.Sequential()
finetune_model.add(transfer_model)
finetune_model.add(Flatten())
finetune_model.add(Dense(64))
finetune_model.add(Activation('relu'))
finetune_model.add(Dropout(0.5))
finetune_model.add(Dense(1))
finetune_model.add(Activation('sigmoid'))
finetune_model.summary()
```

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

```
# 모델의 실행 옵션을 설정합니다.
finetune_model.compile(loss='binary_crossentropy', optimizer=optimizers.
Adam(learning_rate=0.0002), metrics=['accuracy'])

# 학습의 조기 중단을 설정합니다.
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=5)

# 모델을 실행합니다.
history = finetune_model.fit(
        train_generator,
        epochs=20,
        validation_data=test_generator,
        validation_steps=10,
        callbacks=[early_stopping_callback])
```

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

```
# 검증셋과 학습셋의 오차를 저장합니다.
y_vloss = history.history['val_loss']
y_loss = history.history['loss']

# 그래프로 표현해 봅니다.
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')
```

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

```
# 그래프에 그리드를 주고 레이블을 표시하겠습니다.
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

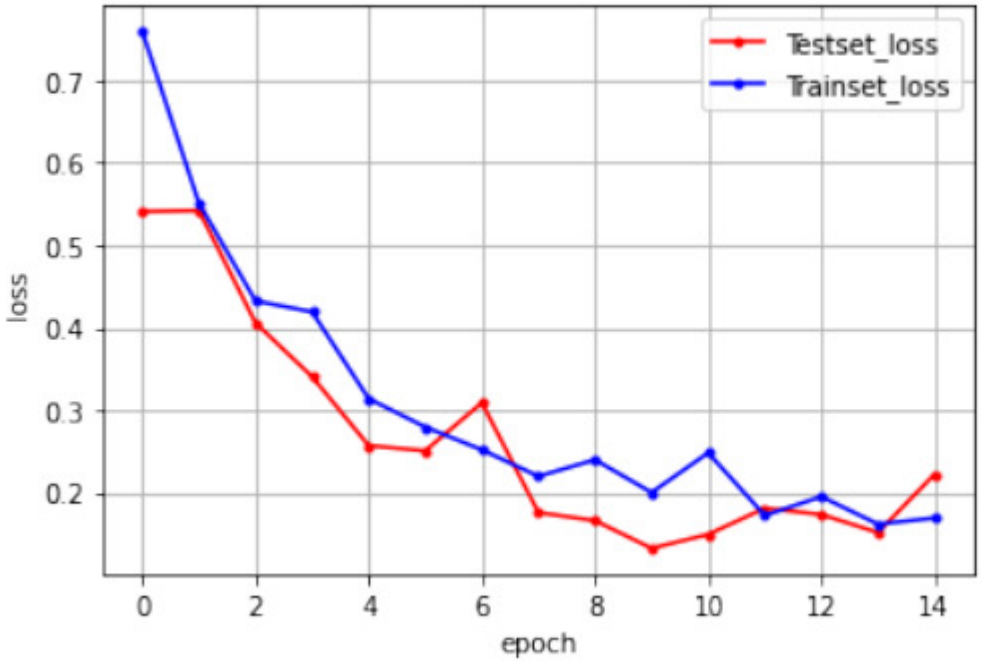
실행 결과

```
Epoch 1/20
32/32 [==============================] - 7s 217ms/step - loss: 0.7590 -
accuracy: 0.5688 - val_loss: 0.5409 - val_accuracy: 0.7800
... (중략) ...
Epoch 15/20
32/32 [==============================] - 10s 300ms/step - loss: 0.1693 -
accuracy: 0.9563 - val_loss: 0.2223 - val_accuracy: 0.9200
```

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습으로 모델 성능 극대화하기

▼ 그림 20-6 | 그림으로 확인하는 학습 결과



2 전이 학습으로 모델 성능 극대화하기

- **전이 학습으로 모델 성능 극대화하기**
  - 첫 정확도 78%에서 학습이 시작
  - 전이 학습을 사용하지 않았던 이전보다 더 높은 정확도로 출발하는 것을 볼 수 있고, 학습 속도도 빨라진 것이 확인
  - 또 그래프의 변화 추이가 안정적임을 확인할 수 있음