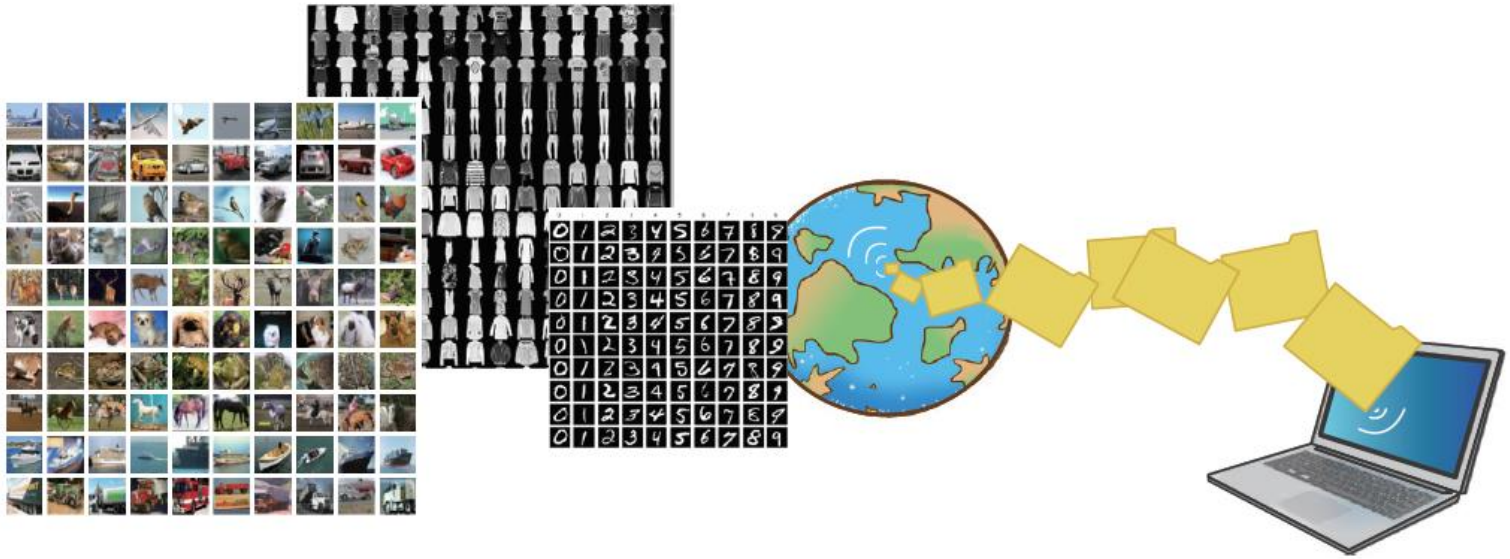
20장 전이 학습을 통해 딥러닝의 성능 극대화하기

목 차

전이 학습을 통해 딥러닝의 성능 극대화하기



- 전이 학습(transfer learning) :

  여러 방법 중에서 수만장에 달하는 기존의 이미지에서 학습한 정보를 가져와 내 프로젝트에 활용하는 것

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 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 구분됨

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 이번에 해 볼 프로젝트는 MRI 뇌 사진을 보고 치매 환자의 뇌인지, 일반인의 뇌인지를 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 선택하여 진행하겠음

1 소규모 데이터셋으로 만드는 강력한 학습 모델

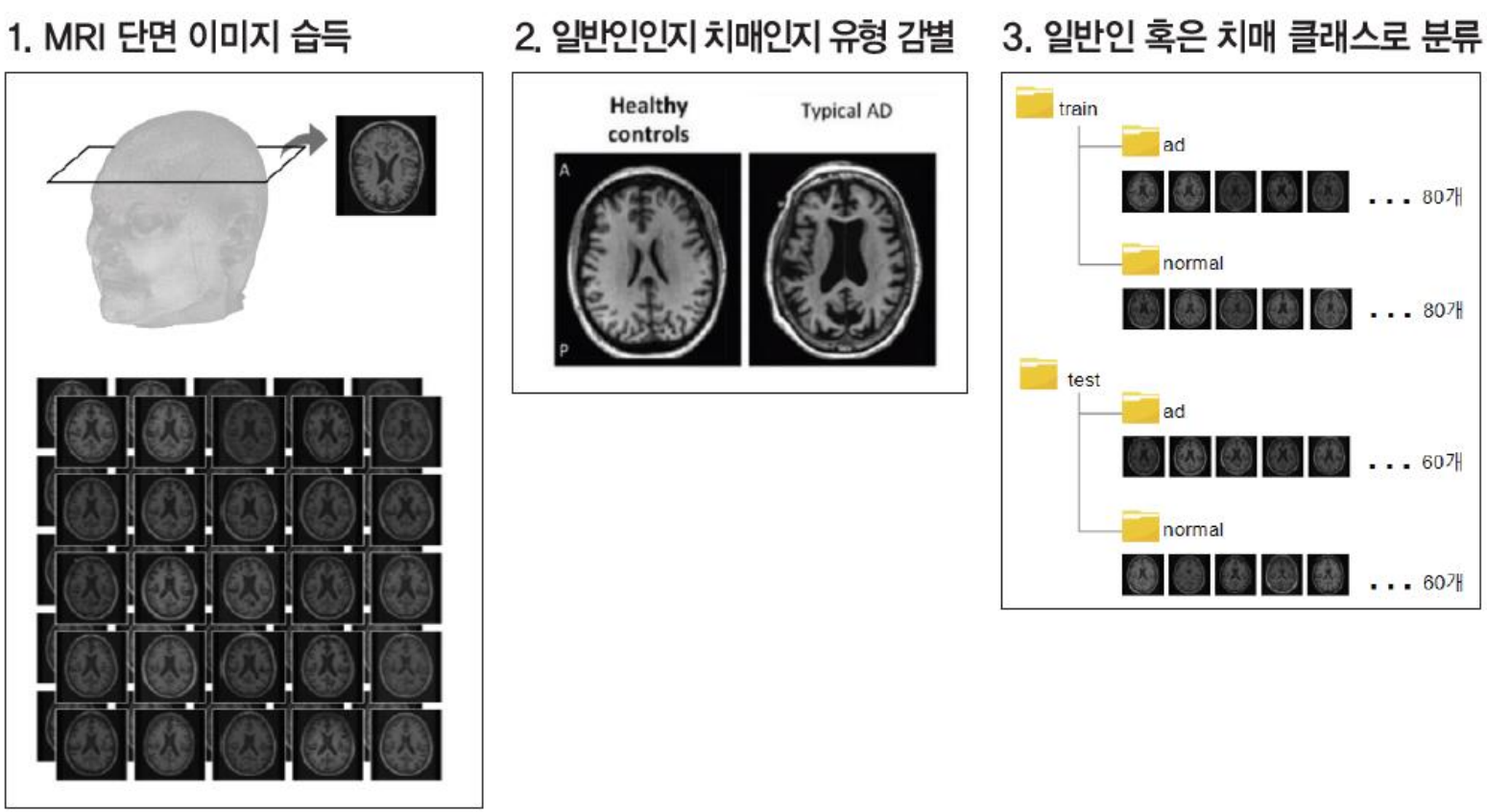


그림 20-1 MRI 뇌 사진 데이터의 구성

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 데이터의 수를 늘리는 ImageDataGenerator( ) 함수와 폴더에 저장된 데이터를 불러오는 flow_from_directory( ) 함수를 사용함
- ImageDataGenerator( ) 함수는 주어진 데이터를 이용해 변형된 이미지를 만들어 학습셋에 포함시키는 편리한 기능을 제공함
- 이미지 데이터의 수를 확장 할 때 효과적으로 사용할 수 있음

1 소규모 데이터셋으로 만드는 강력한 학습 모델

```
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

- rescale :

  주어진 이미지의 크기를 바꾸어 줌

- horizontal_flip, vertical_flip :

  주어진 이미지를 수평 또는 수직으로 뒤집음

- zoom_range :

  정해진 범위 안에서 축소 또는 확대함

- width_shift, height_shift :

  정해진 범위 안에서 그림을 수평 또는 수직으로 랜덤하게 평행 이동시킴

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- rotation_rang :

  정해진 각도만큼 이미지를 회전시킴

- shear_range :

  좌표 하나를 고정시키고 다른 몇 개의 좌표를 이동시키는 변환을 함

- fill_mode :

  이미지를 축소 또는 회전하거나 이동할 때 생기는 빈 공간을 어떻게 채울지 결정함

  nearest 옵션을 선택하면 가장 비슷한 색으로 채워짐

1 소규모 데이터셋으로 만드는 강력한 학습 모델

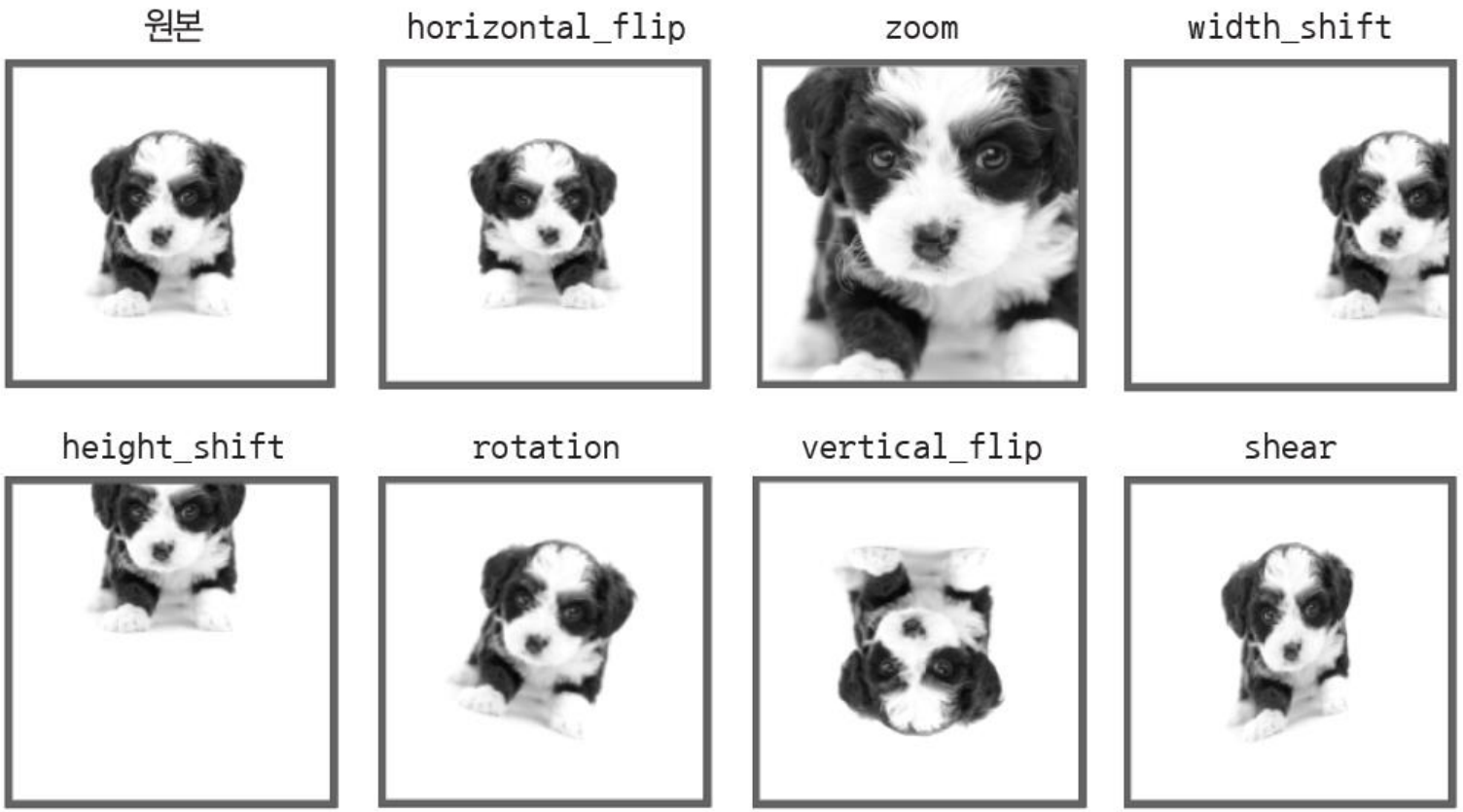


그림 20-2 ImageDataGenerator 옵션의 결과

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 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 사용함
- width_shift, height_shift 인자를 이용해 조금씩 좌우로 수평 이동시킨

  이미지도 만들어 사용함

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 참고로, 데이터 부풀리기는 학습셋에만 적용하는 것이 좋음
- 테스트셋은 실제 정보를 그대로 유지하게 하는 편이 과적합의 위험을 줄일 수 있기 때문임
- 테스트셋은 다음과 같이 정규화만 진행해 줌

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

1 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 사용함

```
train_generator = train_datagen.flow_from_directory(
        './train',                  # 이미지가 위치한 폴더 위치
        target_size=(150, 150),     # 이미지 크기
        batch_size=5,
        class_mode='binary')        # 치매/정상 2진 분류이므로 바이너리 모드로 실행
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 같은 과정을 거쳐서 테스트셋도 생성해 줌

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

1 소규모 데이터셋으로 만드는 강력한 학습 모델

- 이제 컴파일 한 뒤 실행시키면 되는데, 실행 함수로 fit( )가 아닌 fit_generator( )를 써야 하는 것에 주의함

```
model.fit_generator(
        train_generator,          # 앞서 만들어진 train_generator를 학습 모델로 사용
        steps_per_epoch=100,      # 이미지 생성기에서 몇 개의 샘플을 뽑을지 결정
        epochs=20,
        validation_data=test_generator, validation_steps=4)
        # 앞서 만들어진 test_generator를 테스트셋으로 사용
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

코드 20-1 치매 환자의 뇌인지 일반인의 뇌인지 예측하기

- 예제 소스: run_project/22_Data_Augmentation.ipynb

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, Flatten,
Dense, Conv2D, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers, initializers, regularizers,
metrics
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

```
np.random.seed(3)
tf.random.set_seed(3)

train_datagen = ImageDataGenerator(rescale=1./255,
        horizontal_flip=True,       # 수평 대칭 이미지를 50% 확률로 만들어 추가
        width_shift_range=0.1,      # 전체 크기의 10% 범위에서 좌우로 이동
        height_shift_range=0.1,     # 마찬가지로 위아래로 이동
                                    # rotation_range=5,
                                    # shear_range=0.7,
                                    # zoom_range=[0.9, 2.2],
                                    # vertical_flip=True,
        fill_mode='nearest')
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

```
train_generator = train_datagen.flow_from_directory(
        './train',                    # 학습셋이 있는 폴더 위치
        target_size=(150, 150),
        batch_size=5,
        class_mode='binary')

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

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

1 소규모 데이터셋으로 만드는 강력한 학습 모델

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

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

1 소규모 데이터셋으로 만드는 강력한 학습 모델

```
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(2))
model.add(Activation('sigmoid'))
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

```
# 모델 컴파일
model.compile(loss='sparse_categorical_crossentropy',
optimizer=optimizers.Adam(learning_rate=0.0002), metrics
=['accuracy'])

# 모델 실행
history = model.fit_generator(        .
          train_generator,
          steps_per_epoch=100,
          epochs=20,
          validation_data=test_generator,
          validation_steps=4)
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

```
# 결과를 그래프로 표현하는 부분
acc= history.history['accuracy']
val_acc= history.history['val_accuracy']
y_vloss = history.history['val_loss']
y_loss = history.history['loss']

x_len = np.arange(len(y_loss))
plt.plot(x_len, acc, marker='.', c="red", label='Trainset_acc')
plt.plot(x_len, val_acc, marker='.', c="lightcoral", label=
'Testset_acc')
plt.plot(x_len, y_vloss, marker='.', c="cornflowerblue", label=
'Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_
loss')
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

```
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss/acc')
plt.show()
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

실행 결과

```
Epoch 1/20
1/100 [..............................] – ETA: 1:22 – loss: 0.6786 – accuracy: 0.6000
  3/100 [..............................] – ETA: 29s – loss: 0.8702 – accuracy: 0.5333
  4/100 [>.............................] – ETA: 23s – loss: 0.8272 – accuracy: 0.5500
  6/100 [>.............................] – ETA: 16s – loss: 0.8238 – accuracy: 0.5000
  8/100 [=>............................] – ETA: 13s – loss: 0.7732 – accuracy: 0.5250
 10/100 [==>...........................] – ETA: 11s – loss: 0.7788 – accuracy: 0.5200
...
(중략)
Epoch 20/20
...
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

```
 90/100 [============================>...] - ETA: 0s - loss: 0.1939 -
accuracy: 0.9178
 92/100 [============================>...] - ETA: 0s - loss: 0.1911 -
 accuracy: 0.9196
 94/100 [============================>..] - ETA: 0s - loss: 0.1890 -
accuracy: 0.9213
 96/100 [============================>..] - ETA: 0s - loss: 0.1892 -
accuracy: 0.9229
 97/100 [============================>.] - ETA: 0s - loss: 0.1873 -
accuracy: 0.9237
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

```
 98/100 [============================>.] - ETA: 0s - loss: 0.1880 -
accuracy: 0.9245
100/100 [==============================] - 5s 50ms/step - loss: 0.1849
- accuracy: 0.9260 - val_loss: 0.0693 - val_accuracy: 1.0000
```

1 소규모 데이터셋으로 만드는 강력한 학습 모델

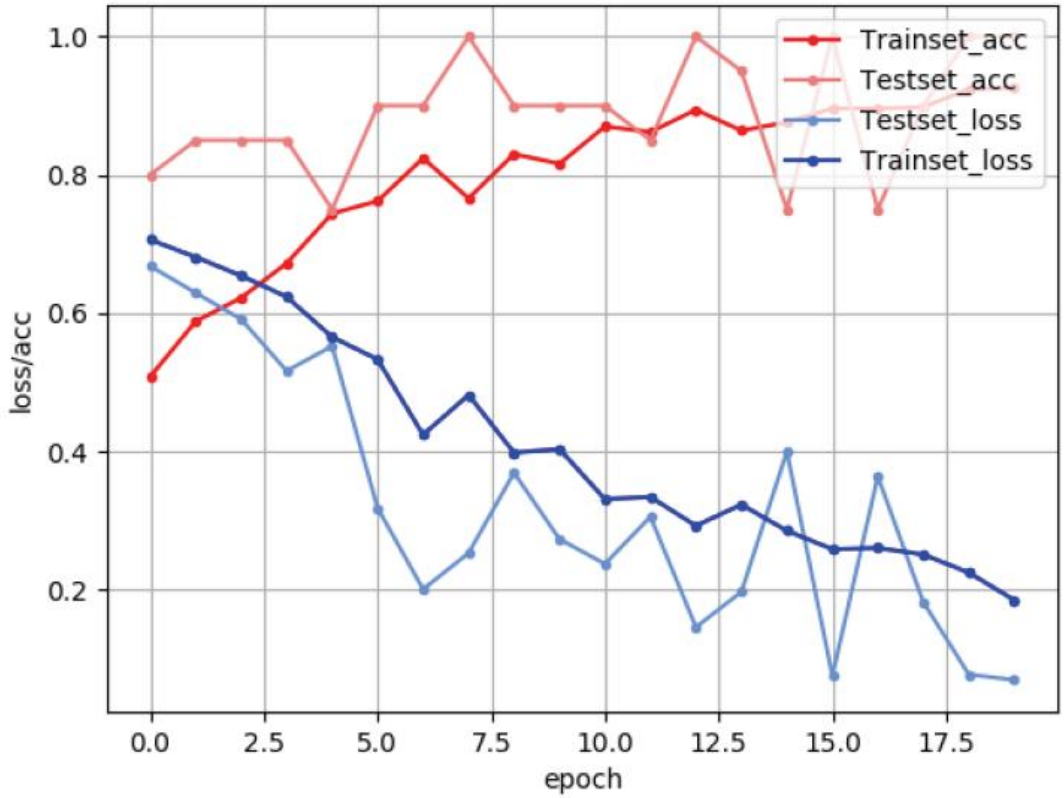


그림 20-3 그래프를 통해 학습셋과 테스트셋의 오차 및 정확도 확인하기

2 전이 학습으로 모델 성능 극대화하기

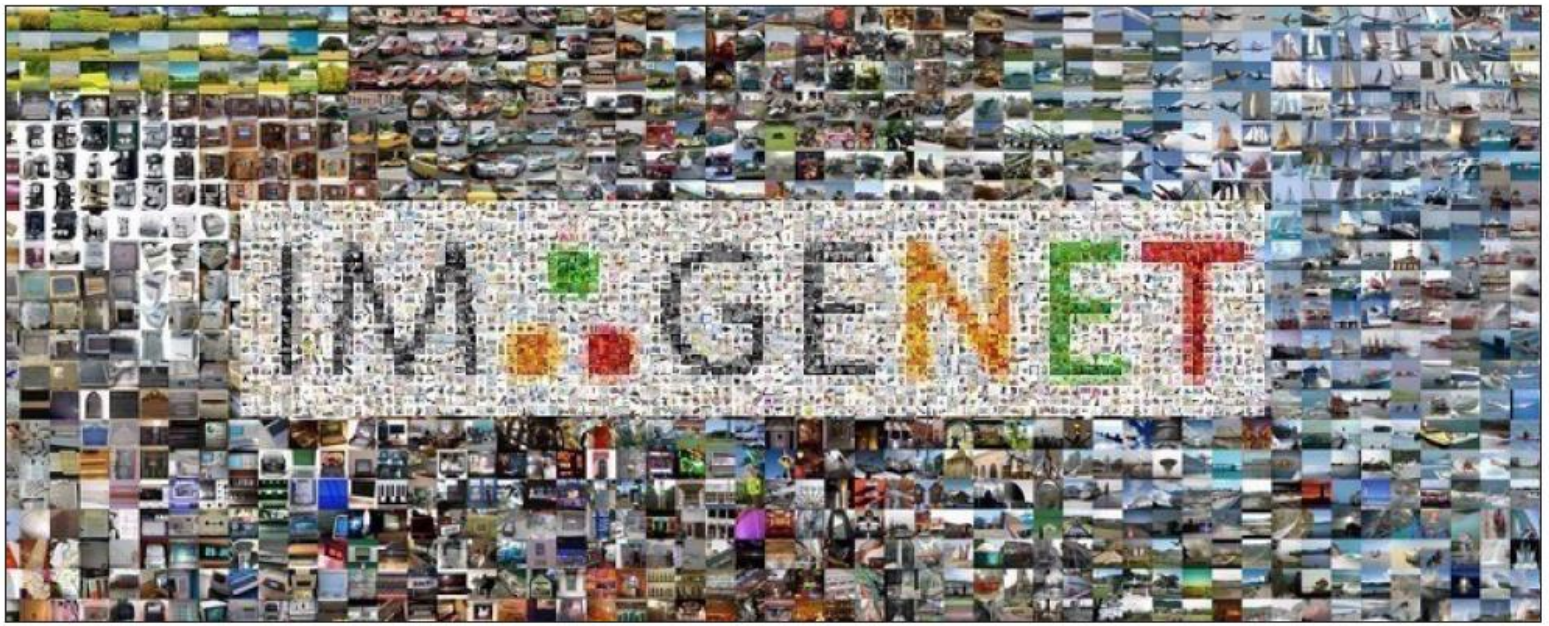


그림 20-4 이미지넷 데이터셋에서 추출한 사진들

2 전이 학습으로 모델 성능 극대화하기

- 이미지넷은 1,000가지 종류로 나뉜 1백 20만 개가 넘는 이미지를 놓고 어떤 물체인지를 맞히는 '이미지넷 이미지 인식 대회(ILSVRC)'에 사용되는 데이터셋
- 전이 학습은 앞서 잠깐 언급한 대로 '기존의 학습 결과를 가져와서 유사한 프로젝트에 사용하는 방법'을 말함
- 뇌 사진만 다루는 치매 분류기를 만드는 데 뇌사진과 관련없는 수백만 장의 이미지넷 학습 정보가 큰 역할을 하는 이유는, '형태'를 구분하는 기본적인 학습이 되어 있기 때문임

2 전이 학습으로 모델 성능 극대화하기

- 전이 학습을 하는 방법
  - 먼저 대규모 데이터 셋에서 학습된 기존의 네트워크를 불러옴
  - CNN 모델의 앞쪽을 이 네트워크로 채움
  - 뒤쪽 레이어에서 나의 프로젝트와 연결함
  - 이 두 네트워크가 잘 맞물리게끔 미세 조정(Fine tuning)을 하면 됨

2 전이 학습으로 모델 성능 극대화하기

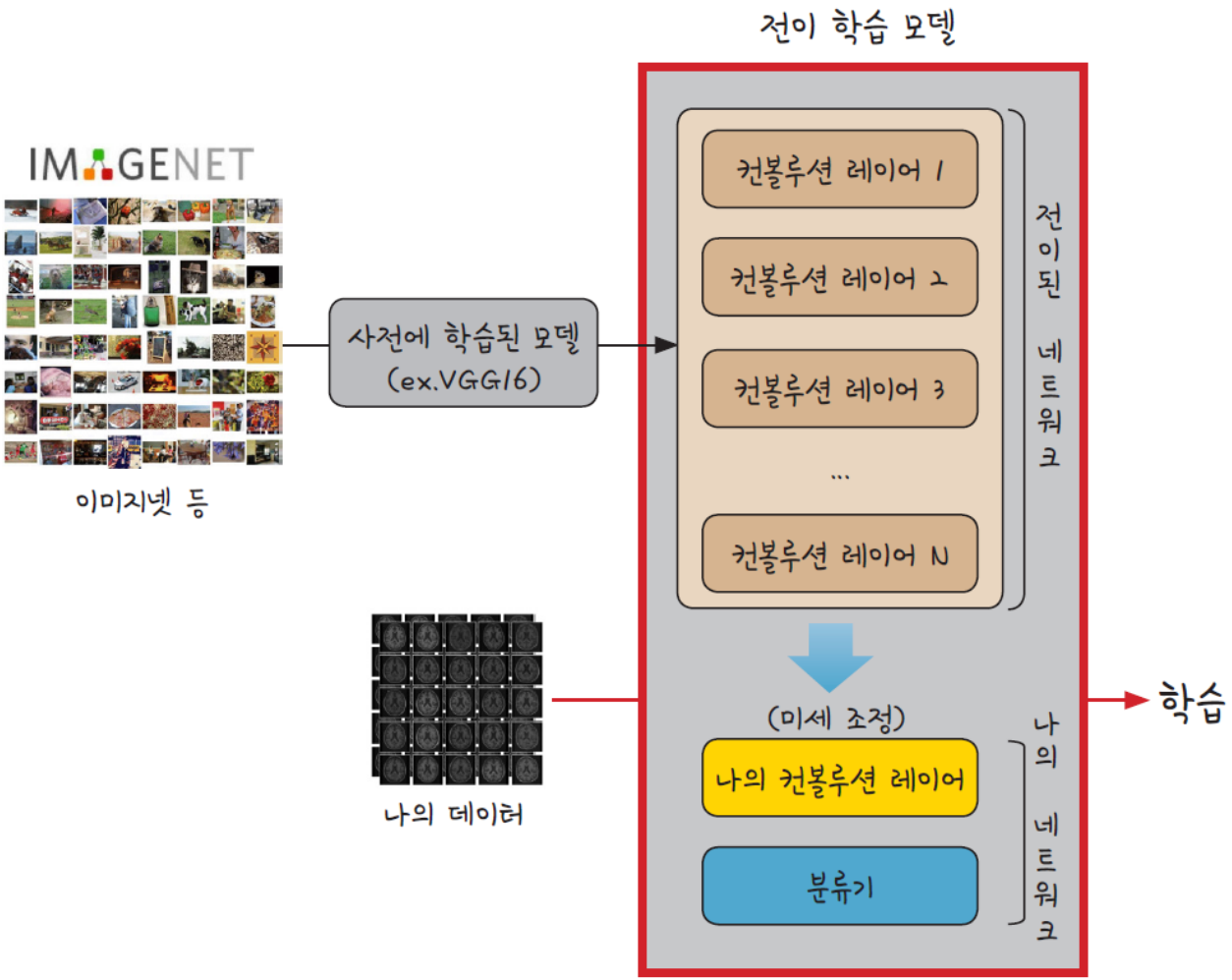


그림 20-5 전이 학습의 구조

2 전이 학습으로 모델 성능 극대화하기

- VGGNet은 옥스포드대학의 연구팀 VGG에 의해 개발된 모델로 2014년 이미지넷 이미지 인식 대회에서 2위를 차지한 모델

TIP

VGG외에도 ResNet, Inception, ModileNet, DenseNet 등 많은 모델을 불러올 수 있습니다. 각 네트워크에 대한 상세한 설명은 케라스 공식 홈페이지를 참조하세요(https://keras.io/applications/).

2 전이 학습으로 모델 성능 극대화하기

- 다음은 모델명을 transfer_model로 정하고 VGG16을 불러온 모습임
- include_top은 전체 VGG16의 마지막 층, 즉 분류를 담당하는 곳을 불러올지 말지를 정하는 옵션
- 우리가 만든 로컬 네트워크를 연결할 것이므로 False로 설정함
- 불러올 부분은 새롭게 학습되는 것이 아니므로 학습이 되지 않도록 transfer_model.trainable 옵션 역시 False로 설정함

```
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

- transfer_model.summary( ) 함수를 통해 학습 구조를 보면 다음과 같음

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

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

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
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808

2 전이 학습으로 모델 성능 극대화하기

```
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

- 학습 가능한 파라미터(Trainable params)가 없음을 확인함
- 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, activation='relu'))
finetune_model.add(Dense(2, activation='softmax'))
finetune_model.summary()
```

2 전이 학습으로 모델 성능 극대화하기

- 다음은 finetune_model.summary( ) 함수를 통해 학습 구조를 살펴본 결과

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 64)	524352
dense_1 (Dense)	(None, 2)	130

2 전이 학습으로 모델 성능 극대화하기

```
=================================================================

Total params: 15,239,170

Trainable params: 524,482

Non-trainable params: 14,714,688

_________________________________________________________________
```

2 전이 학습으로 모델 성능 극대화하기

코드 20-2 전이 학습 실습하기

- 예제 소스: run_project/23_Transfer_Learning.ipynb

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

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

2 전이 학습으로 모델 성능 극대화하기

```
np.random.seed(3)
tf.compat.v1.set_random_seed(3)

train_datagen = ImageDataGenerator(rescale=1./255, horizontal_
flip=True, width_shift_range=0.1, height_shift_range=0.1, fill_
mode='nearest')

train_generator = train_datagen.flow_from_directory(
        'train',
        target_size=(150, 150),
        batch_size=5,
        class_mode='binary')
test_datagen = ImageDataGenerator(rescale=1./255,
```

2 전이 학습으로 모델 성능 극대화하기

```
                                    horizontal_flip=True,
                                    width_shift_range=0.1,
                                    height_shift_range=0.1,
                                    fill_mode='nearest')

test_generator = test_datagen.flow_from_directory('test', target_
size=(150, 150), batch_size=5, class_mode='binary')

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

```
finetune_model = models.Sequential()
finetune_model.add(transfer_model)
finetune_model.add(Flatten())
finetune_model.add(Dense(64, activation='relu'))
finetune_model.add(Dense(2, activation='softmax'))
finetune_model.summary()

finetune_model.compile(loss='sparse_categorical_
crossentropy', optimizer=optimizers.Adam(learning_rate=0.0002),
metrics=['accuracy'])
```

2 전이 학습으로 모델 성능 극대화하기

```
history = finetune_model.fit_generator(train_generator, steps_
per_epoch=100, epochs=20, validation_data=test_generator,
validation_steps=4)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
y_vloss = history.history['val_loss']
y_loss = history.history['loss']
```

2 전이 학습으로 모델 성능 극대화하기

```
# 그래프로 표현
x_len = np.arange(len(y_loss))
plt.plot(x_len, acc, marker='.', c="red", label='Trainset_acc')
plt.plot(x_len, val_acc, marker='.', c="lightcoral",
label='Testset_acc')
plt.plot(x_len, y_vloss, marker='.', c="cornflowerblue",
label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')
```

2 전이 학습으로 모델 성능 극대화하기

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

2 전이 학습으로 모델 성능 극대화하기

실행 결과

```
Epoch 1/20
1/100 [..............................] – ETA: 1:25 – loss: 0.7850 – accuracy: 0.6000
 2/100 [..............................] – ETA: 48s – loss: 0.7797 – accuracy: 0.5000
 3/100 [..............................] – ETA: 36s – loss: 0.7769 – accuracy: 0.4667
 4/100 [>.............................] – ETA: 29s – loss: 0.8710 – accuracy: 0.3500
 5/100 [>.............................] – ETA: 25s – loss: 0.8126 – accuracy: 0.4400
 6/100 [>.............................] – ETA: 23s – loss: 0.8000 – accuracy: 0.4000
 7/100 [=>............................] – ETA: 21s – loss: 0.7849 – accuracy: 0.4286
 8/100 [=>............................] – ETA: 20s – loss: 0.7671 – accuracy: 0.4500
 9/100 [=>............................] – ETA: 18s – loss: 0.7326 – accuracy: 0.4667
10/100 [==>...........................] – ETA: 17s – loss: 0.7450 – accuracy: 0.4400
...
(중략)
```

2 전이 학습으로 모델 성능 극대화하기

```
Epoch 20/20
...
 97/100 [============================>.] - ETA: 0s - loss: 0.0124 -
accuracy: 1.0000
 98/100 [============================>.] - ETA: 0s - loss: 0.0123 -
accuracy: 1.0000
 99/100 [============================>.] - ETA: 0s - loss: 0.0124 -
accuracy: 1.0000
100/100 [==============================] - 13s 132ms/step - loss: 0.0123
- accuracy: 1.0000 - val_loss: 0.0540 - val_accuracy: 1.0000
```

2 전이 학습으로 모델 성능 극대화하기

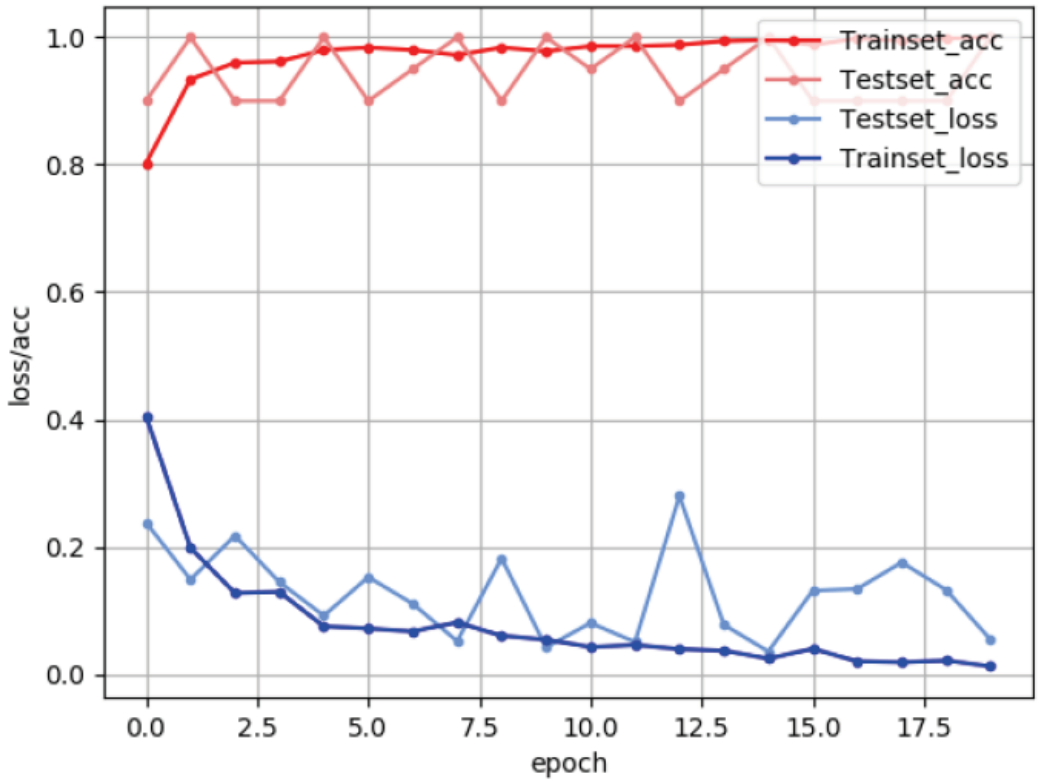


그림 20-6 그래프 출력 결과

3 맺음말

- 이제 남은 일은, 스스로의 데이터를 찾아서 이를 활용해 보는 것
- 캐글(www.kaggle.com) 등에서 더 많고 다양한 데이터를 내려받을 수 있고, 새로운 논문이나 깃허브(github) 검색으로 최신 방법들을 실행하고 분석할 수도 있음