13. 사전 훈련된 모델 다루기

개요

- 딥러닝이 발전함에 따라 다양한 네트워크가 개발됐습니다.
- 좋은 성능을 보이는 네트워크는 수십, 수백개의 레이어를 쌓은 경우가 대부분이고, 레이어가 늘어남에 따라 네트워크를 훈련시키는 데 걸리는 시간도 증가합니다.
- 유명한 CNN 중 하나인 ResNet-50은 8장의 P100 GPU를 사용해 ImageNet의 사진을 잘 분류하도록 학습시키는 데 29 시간이 걸립니다.
- 페이스북 연구팀은 이 시간을 1 시간으로 줄였지만 그 대신 훨씬 많은 256장의 GPU를 사용했습니다.
- 최근 이미지 분류 문제에서 더 좋은 결과를 내고 있는 NAS Net은 AutoML 기법을 사용해 최적의 네트워크 구조를 스스로 학습하기 때문에 훨씬 더 많은 훈련 시간이 필요합니다.

개요

- 다행히 딥러닝 기술은 개방적인 환경에서 빠르게 발전하고 있습니다.
- 연구자들은 자신이 만든 사전 훈련된 모델(pre-trained model)을 인터넷에 올려놓아 다른 사람들이 쉽게 내려 받을 수 있게 합니다.
- 이렇게 얻은 모델을 그대로 시용할 수도 있고, 전이 학습 (Transfer Learning) 이나 신경 스타일 전이 (Neural Style Transfer) 처럼 다른 과제를 위해 재가공해서 사용할 수도 있습니다.

텐서플로 허브

- 텐서 플로에서 제공하는 텐서플로 허브(TensorFlow Hub)는 재사용 가능한 모델을 쉽게 이용할 수 있는 라이브러리입니다.
- 텐서플로 허브 홈페이지에서는 이미지, 텍스트, 비디오 등의 분야에서 사전 훈련된 모델들을 검색해 볼 수 있습니다.
- 텐서플로 2.0을 사용하면 텐서플로 허브는 따로 설치할 필요가 없고 라이브리리를 바로 불러올 수 있습니다.

텐서플로 허브

- 텐서플로 허브에서 사전 훈련된 MobileNet 모델 불러오기

```
import tensorflow as tf
import tensorflow_hub as hub

mobile_net_url =
        "https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/2"
model = tf.keras.Sequential([
    hub.KerasLayer(handle=mobile_net_url,    input_shape=(224,    224,    3),
trainable=False)
])

model.summary()
```

텐서플로 허브

- 텐서플로 허브에서 사전 훈련된 MobileNet 모델 불러오기

```
# 그림 8.2 좌측 전체 네트워크 구조 출력 코드
from tensorflow.keras.applications import MobileNetV2

mobilev2 = MobileNetV2()
tf.keras.utils.plot_model(mobilev2)
```

텐서플로 허브

- 먼저 tensorflow_hub 라이브러리를 hub 라는 약어로 임포트 합니다.
- 그 다음 컨볼루션 신경망의 하나인 MobileNet 버전 2를 불러옵니다.
- MobileNet은 계산 부담이 큰 컨볼루션 신경망을 연산 성능이 제한된 모바일 환경에서도 작동 가능하도록 네트워크 구조를 경량화한 것 입니다.
- MobileNet 버전 2는 버전 1을 개선했고 파라미터 수도 더 줄어들었습니다.

텐서플로 허브

- 텐서플로 허브에 올라와 있는 모델은 hub.KerasLayer() 명령으로 tf.keras에서 사용 가능한 레이어로 변환할 수 있습니다.
- model.summary() 로 파라미터 개수를 확인해보면 354 만 개 정도입니다.
- ResNet-50에는 2,560만 개, ResNet-152에는 6천만 개의 파라미터가 존재하는 것에 비교하면 상대적으로 파라미터 수가 적은 편입니다.

텐서플로 허브

- 앞의 코드에서 불러온 MobileNet은 ImageNet 데이터로 학습시켰습니다.
- 학습된 이미지의 분류 중 일부는 표에 나와 있습니다.

인덱스	분류
0	background
1	tench
2	goldfish
3	great white shark
4	tiger shark
5	hammerhead
6	electric ray
7	stingray
8	cock
9	hen
...	...
998	bolete
999	ear
1000	toilet tissue

텐서플로 허브

- MobileNet은 ImageNet에 존재하는 1,000종류의 이미지를 분류할 수 있으며, 이 가운데 어떤 것에도 속하지 않는다고 판단될 때는 background에 해당하는 인덱스 0을 반환합니다.
- 이미지의 분류는 수탉(cock)과 암탉(hen)을 분류할 정도로 상세하고 화장지(toilet tissue) 같은 사물도 포함하고 있습니다.
- MobileNet의 성능을 평가하기 위해 이미지를 학습시켰을 때 얼마나 적합한 라벨로 분류하는지 알아보겠습니다.
- ImageNet의 데이터 중 일부만 모아놓은 ImageNetV2를 사용하겠습니다.
- ImageNetV2는 아마존 메커니컬 터크(Amazon Mechanical Turk)를 이용해 다수의 참가자에게서 클래스 예측값을 받아서 선별한 데이터입니다.
- 여기서는 각 클래스에서 가정 많은 선택을 받은 이미지 10장씩을 모아놓은 10,000장의 이미지가 포함된 TopImages 데이터를 사용하겠습니다.

텐서플로 허브

- ImageNetV2-TopImages 불러오기

```
import os
import pathlib

content_data_url = '/content/sample_data'
data_root_orig = tf.keras.utils.get_file('imagenetV2', 'https://s3-us-west-2.amazonaws.com/imagenetv2public/imagenetv2-top-images.tar.gz',
cache_subdir='datasets',cache_dir=content_data_url, extract=True)
data_root = pathlib.Path(content_data_url + '/datasets/imagenetv2-top-images-format-val')
print (data_root )
```

텐서플로 허브

- tf.keras.utils.get_file() 함수로 ImageNetV2 데이터를 불러올 수 있습니다.
- 함수의 인수 중 extract=True로 지정했기 때문에 tar.gz 형식의 압축 파일이 자동으로 해제되어 구글 코랩 가상 머신에 저장됩니다.
- 이제 데이터를 잘 불러왔는지 확인하기 위해 하위 디렉터리를 순회하며 디렉터리의 경로를 출력 해봅니다.

텐서플로 허브

- 디렉터리 출력

```
for idx, item in enumerate(data_root.iterdir()):
    print( item )
    if idx == 9 :
        break
```

- 데이터 디렉터리 밑에 각 라벨에 대한 숫자 이름으로 하위 디렉터리가 만들어져 있는 것을 확인할 수 있습니다.
- 하위 디렉터리는 0 ~999까지 총 1,000개입니다.
- 라벨에 대한 숫자가 어떤 데이터를 뜻하는 지에 대한 정보인 라벨 텍스트는 따로 불러와야 합니다.
- MobileNet에서 사용된 라벨은 다음 예제에서 역시 tf.keras.utils.get_file() 함수로 불러옵니다.

텐서플로 허브

- ImageNet 라벨 텍스트 불러오기

```
# 8.4 ImageNet 라벨 텍스트 불러오기
label_file =
tf.keras.utils.get_file('label',
'https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLab
els.txt')
label_text = None
with open(label_file, 'r') as f:
    label_text = f.read().split('\n')[:-1]
print(len(label_text))
print(label_text[:10])
print(label_text[-10:])
```

텐서플로 허브

- 그럼 이제 이미지를 확인해보겠습니다.
- 지금까지 배운 것처럼 matplotlib.pyplot을 이용해 이미지를 출력할 수 있습니다.

```
# 8.5 이미지 확인
import PIL.Image as Image
import matplotlib.pyplot as plt
import random

all_image_paths = list(data_root.glob('*/*'))
all_image_paths = [str(path) for path in all_image_paths]
# 이미지를 랜덤하게 섞습니다.
random.shuffle(all_image_paths)

image_count = len(all_image_paths)
print('image_count:', image_count)
```

텐서플로 허브

- 그럼 이제 이미지를 확인해보겠습니다.
- 지금까지 배운 것처럼 matplotlib.pyplot을 이용해 이미지를 출력할 수 있습니다.

```
plt.figure(figsize=(12,12))
for c in range(9):
    image_path = random.choice(all_image_paths)
    plt.subplot(3,3,c+1)
    plt.imshow(plt.imread(image_path))
    idx = int(image_path.split('/')[-2]) + 1
    plt.title(str(idx) + ', ' + label_text[idx])
    plt.axis('off')
plt.show()
```

텐서플로 허브

- 출력에서는 10,000장의 이미지 중 랜덤하게 뽑은 9장의 이미지를 확인할 수 있습니다.
- 한 가지 유의해야 할 점은 라벨 텍스트는 background가 포함돼 있어서 총 1,001개의 텍스트가 있지만 실제 테이터의 라벨은 0에서 999까지의 1,000개라는 점입니다.
- 이 차이를 무마하기 위해 idx = int(image_path.split(' /') [-2]) + 1 에서는 파일 경로의 라벨 디렉터리에 해당하는 부분을 정수로 변환한 다음 1을 더해서 첫 번째부터 1,000번째 까지의 라벨 텍스트와 동일한 값을 가리키게 합니다.

텐서플로 허브

- 이제 이 이미지들을 MobileNet이 얼마나 잘 분류하는지 확인해 보겠습니다.
- 전통적으로 ImageNet 대회에서는 네트워크가 예측한 값 중 상위 5개 이내에 데이터의 실제 분류가 포함돼 있으면 정답으로 인정하는 Top-5 정확도를 분류 정확도로 측정했습니다.
- 그런데 요즘은 네트워크의 성능이 높아져서 Top-l 정확도, 즉 가장 높은 값이 데이터의 실제 분류와 일치하는 비율을 놓고 경쟁하는 추세입니다.
- 다음 예제에서도 Top- 5 정확도와 Top-l 정확도를 측정해보겠습니다.

텐서플로 허브

- MobileNet의 분류 성능 확인

```
# 8.6 MobileNet의 분류 성능 확인
import cv2
import numpy as np

top_1 = 0
top_5 = 0
for image_path in all_image_paths:
    img = cv2.imread(image_path)
    img = cv2.resize(img, dsize=(224, 224))
    img = img / 255.0
    img = np.expand_dims(img, axis=0)
    top_5_predict = model.predict(img)[0].argsort()[::-1][:5]
    idx = int(image_path.split('/')[-2])+1
```

텐서플로 허브

- MobileNet의 분류 성능 확인

```
if idx in top_5_predict:
        top_5 += 1
        if top_5_predict[0] == idx:
            top_1 += 1

print('Top-5 correctness:', top_5 / len(all_image_paths) * 100, '%')
print('Top-1 correctness:', top_1 / len(all_image_paths) * 100, '%')
```

텐서플로 허브

- Top- 5 정확도는 83.84%. Top- l 정확도는 59.45%가 나옵니다.
- 논문에서는 정확도를 높이기 위해 이미지에 여러 가지 전처리 기법을 사용하지만 여기서는 특별한 방법을 사용하지 않았기 때문에 정확도가 약간 낫게 나옵니다.
- 참고로 MobileNet의 처음 버전은 Top - 5 정확도가 89.9%, Top- 1 정확도가 70.9%가 나옵니다.
- 첫 줄에서 임포트한 cv2는 OpenCV(Open Source Computer Vision) 라이브러리입니다.
- 이미지를 메모리에 불러오고 크기를 조정하는 등의 작업을 편하게 해주기 때문에 사용했습니다.
- 자주 쓰이는 라이브러리이기 때문에 구글 코랩에서는 별도의 설치 없이 불러올 수 있도록 미리 설치돼 있습니다

텐서플로 허브

- 다음 코드에서 Top- 5 정확도를 측정하기 위한 top_5_predict를 구합니다.

```
top_5_predict = model.predict(img)[0].argsort()[::-1][:5]
```

- 여기서 numpy.argsort ( )는 값을 정렬하는 대신 인덱스를 정렬합니다.
- 즉, [99, 32, 5, 64]라는 넘파이 array가 있다면 sort( ) 와 argsort( )를 사용했을 때 각각 다음과 같은 값을 얻게 됩니다.

```
# numpy.argsort() 설명 코드
a = np.array([99,32,5,64])
arg = np.argsort(a)
print(arg)
print(np.sort(a))
print(a[arg])
```

텐서플로 허브

- argsort( ) 결과를 다시 원래의 넘파이 array에 넣으면 정렬된 넘파이 array, 즉 sort( )를 사용한 것과 같은 결과를 얻게 됩니다.
- 그런데 우리가 원하는 것은 예측 확률이 높은 순서이기 때문에 오름차순으로 정렬된 array를 array[::-1]로 반전시킵니다.
- 그 다음에 앞에서부터 5번째까지의 값을 array [: 5] 로 잘라서 top_5_predict에 저장합니다.

텐서플로 허브

- 다음 예제에서는 MobileNet이 분류하는 라벨을 실제로 확인하고 Top- 5 예측을 표시합니다.

```
# 8.7 MobileNet의 분류 라벨 확인
plt.figure(figsize=(16,16))

def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum(axis=0)

for c in range(3):
    image_path = random.choice(all_image_paths)

    # 이미지 표시
    plt.subplot(3,2,c*2+1)
    plt.imshow(plt.imread(image_path))
    idx = int(image_path.split('/')[-2]) + 1
    plt.title(str(idx) + ', ' + label_text[idx])
    plt.axis('off')
```

텐서플로 허브

```
# 예측값 표시
plt.subplot(3,2,c*2+2)
img = cv2.imread(image_path)
img = cv2.resize(img, dsize=(224, 224))
img = img / 255.0
img = np.expand_dims(img, axis=0)

# MobileNet을 이용한 예측
logits = model.predict(img)[0]
prediction = softmax(logits)

# 가장 높은 확률의 예측값 5개를 뽑음
top_5_predict = prediction.argsort()[::-1][:5]
labels = [label_text[index] for index in top_5_predict]
color = ['gray'] * 5
if idx in top_5_predict:
    color[top_5_predict.tolist().index(idx)] = 'green'
color = color[::-1]
plt.barh(range(5), prediction[top_5_predict][::-1] * 100, color=color)
plt.yticks(range(5), labels[::-1])
```

텐서플로 허브

- 랜덤하게 뽑은 세 개의 이미지 중 2개는 Top-1 예측을 맞췄고, 나머지 하나도 Top-5 예측 안에 이미지의 라벨이 들어 있습니다.
- 그 밖에 높은 값을 보이는 예측도 헷갈리기 쉽다고 생각할 만한 것들입니다
- 이렇게 별도의 훈련 과정 없이 마리 훈련된 모델을 텐서플로 허브에서 불러오는 것만으로 네트워크를 그대로 사용할 수 있습니다.

전이 학습

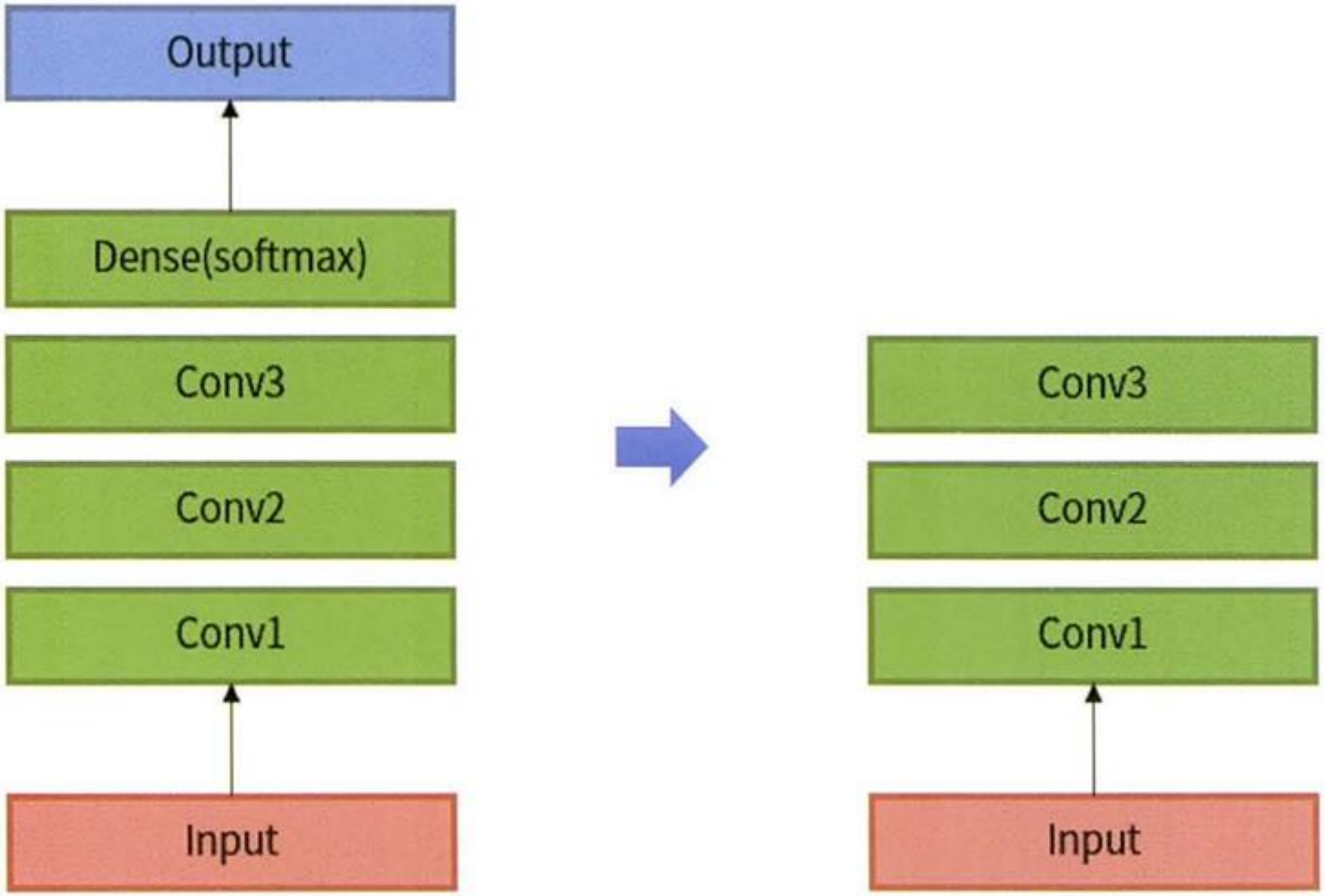
- 전이 학습(Transfer Learning)은 미리 훈련된 모델을 다른 작업에 사용하기 위해 추가적인 학습을 시키는 것입니다.
- 이때 훈련된 모델은 데이터에서 유의미한 특징(feature)을 뽑아내기 위한 특징 추출기 (Feature Extractor)로 쓰이거나, 모델의 일부를 재학습시키기도 합니다.

전이 학습

- 모델의 일부를 재학습시키기
  - 전이 학습을 설명하기 위해 가장 많이 쓰이는 컨볼루션 신경망의 전이 학습을 예로 들겠습니다.
  - 미리 훈련된 컨볼루션 신경망을 불러올 때 가장 마지막의 Dense 레이어를 제외합니다.
  - 이미지를 분류하는 컨볼루션 신경망에서 이 레이어는 소프트맥스 활성화함수로 실제 분류 작업을 수행하는 레이어입니다.
  - ImageNet 데이터로 학습했다면 이 레이어의 뉴런 수는 ImageNet 문제의 이미지 분류 숫자와 같은 1,000일 것입니다.

전이 학습

- 모델의 일부를 재학습시키기

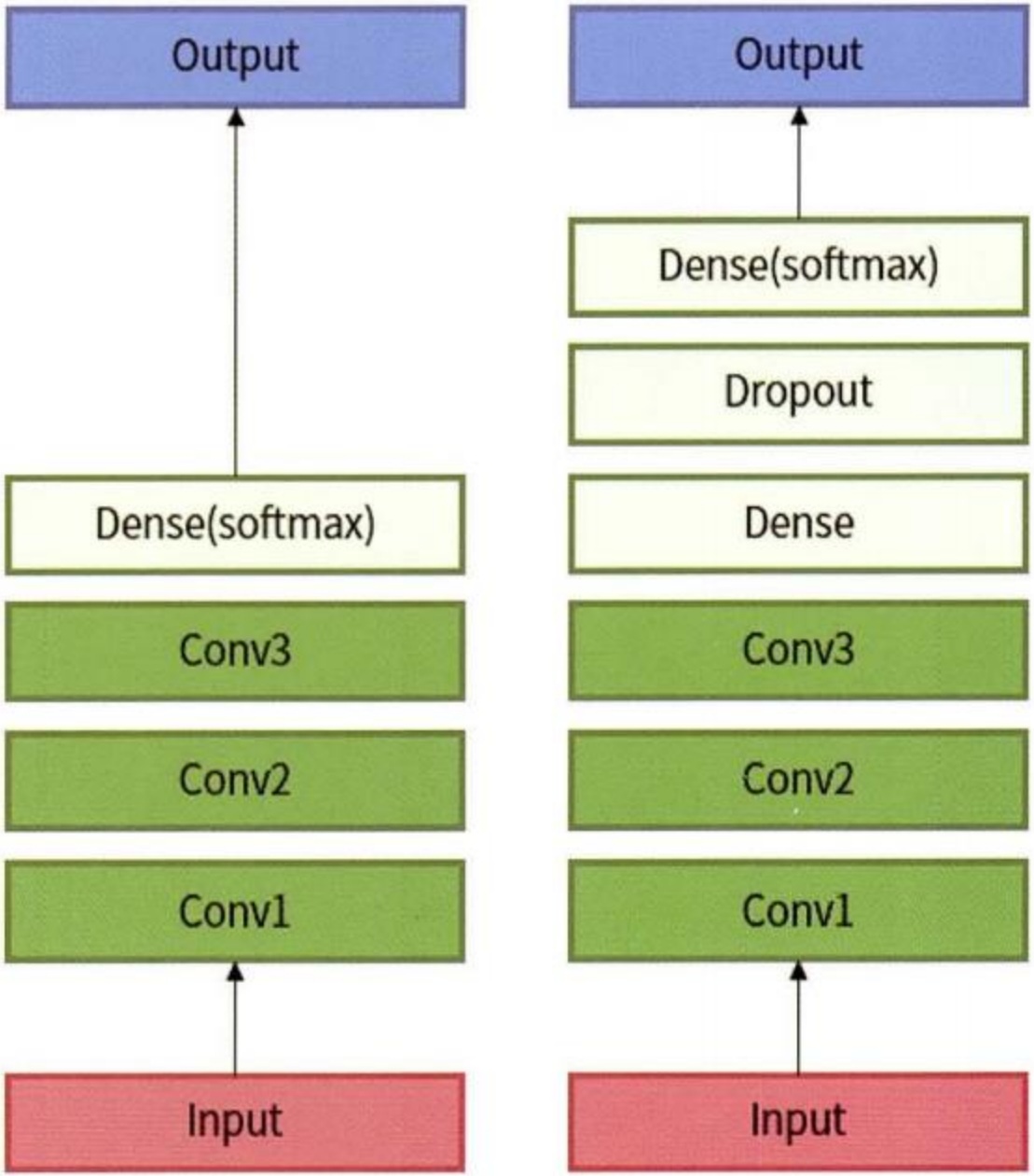


전이 학습

- 모델의 일부를 재학습시키기
  - 그 다음에 해야 할 일은 새로운 분류 작업을 위한 레이어를 추가하는 것입니다.
  - 마지막에 그림의 왼쪽처럼 Dense 레이어 하나만 추가하거나, 오른쪽처럼 좀 더 복잡한 분류 작업을 위해 여러 개의 Dense 레이어와 드롭아웃 레이어를 추가하기도 합니다.
  - 마지막에 추가되는 레이어의 뉴런 수는 새로운 분류 작업의 범주 수와 같습니다.
  - 새로운 작업에 쓰이는 데이터가 5장에 나온 Fashion_MNIST라면 마지막 레이어의 뉴런 수는 10이 될 것입니다.

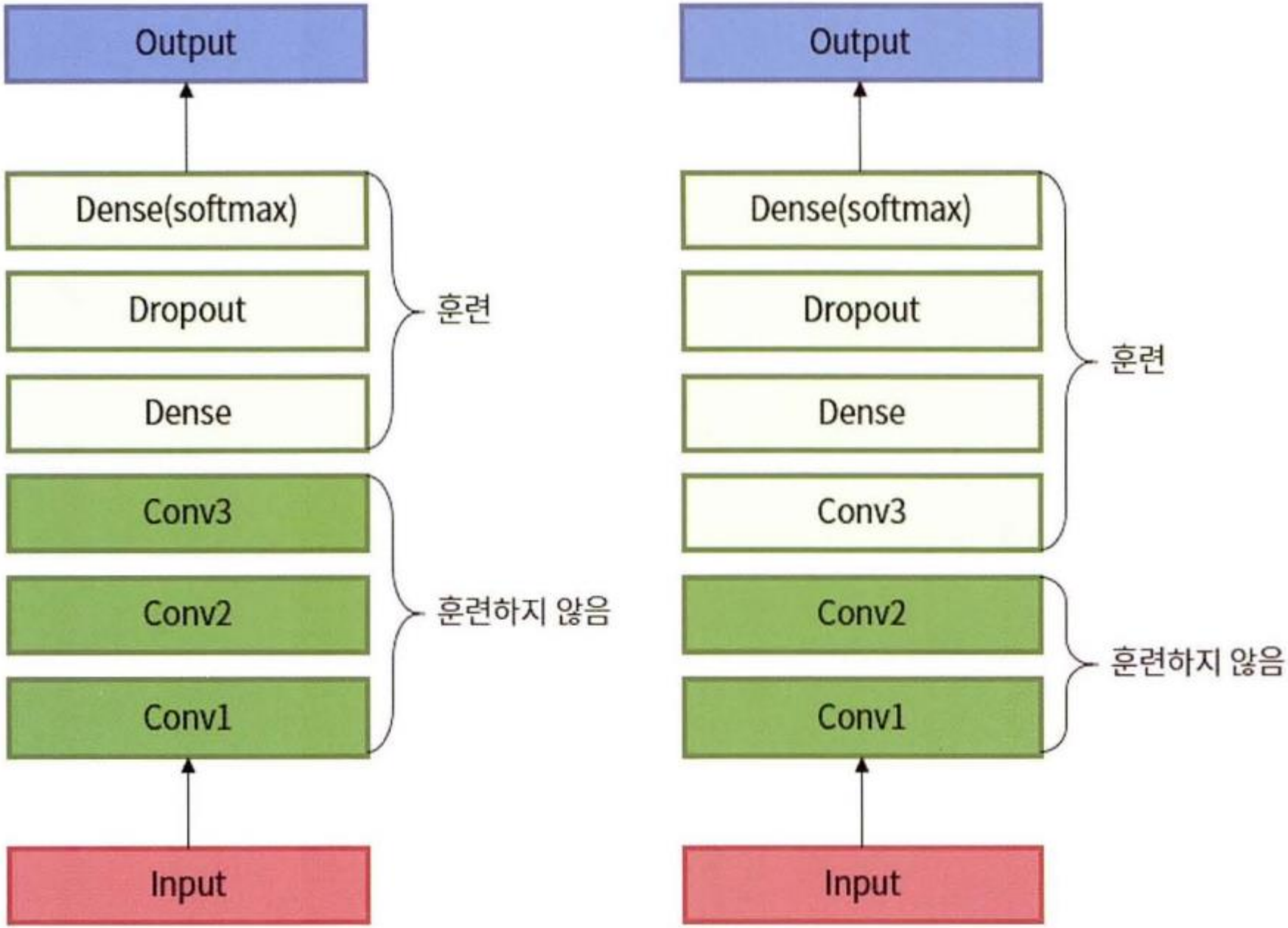
전이 학

- 모델의


Output
Dense(softmax)
Conv3
Conv2
Conv1
Input
Output
Dense(softmax)
Dropout
Dense
Conv3
Conv2
Conv1
Input

전이 학습

- 모델의 일부를 재학습시키기
  - 그 다음에는 네트워크를 훈련시킵니다.
  - 이때 새로 추가된 레이어의 가중치만 훈련시킬 수도 있고, 미리 훈련된 모델의 일부 레이어를 훈련시킬 수도 있습니다.
  - 이때 훈련하지 않는 레이어를 '얼린다(freeze)'라고 표현합니다.
  - 레이어를 얼마나 얼릴지는 새로운 직업을 위한 데이터의 양에 의해 결정됩니다.
  - 새로운 작업을 위한 데이터의 양이 많을수록 기존에 훈련된 데이터와 차이가 많이져서 다시 학습할 필요가 생기기 때문에 얼리는 레이어의 양을 줄이게 됩니다.


Output
Dense(softmax)
Dropout
Dense
훈련
Conv3
Conv2
Conv1
훈련하지 않음
Input
Output
Dense(softmax)
Dropout
Dense
Conv3
훈련
Conv2
Conv1
훈련하지 않음
Input

전이 학습

- 모델의 일부를 재학습시키기
  - 그럼 이제 본격적으로 전이 학습을 예제 코드와 함께 배워보겠습니다.
  - 여기서 사용할 새로운 데이터는 스탠퍼드 대학의 Dogs Dataset입니다.
  - 이 데이터는 2만여 개의 사진으로 구성돼 있으며 120가지 견종에 대한 라벨이 붙어 있습니다.
  - ImageNet보다는 훨씬 적은 양의 데이터이면서 분류의 수도 아주 적지는 않기 때문에 전이 학습을 실습하기에 적당한 데이터입니다.

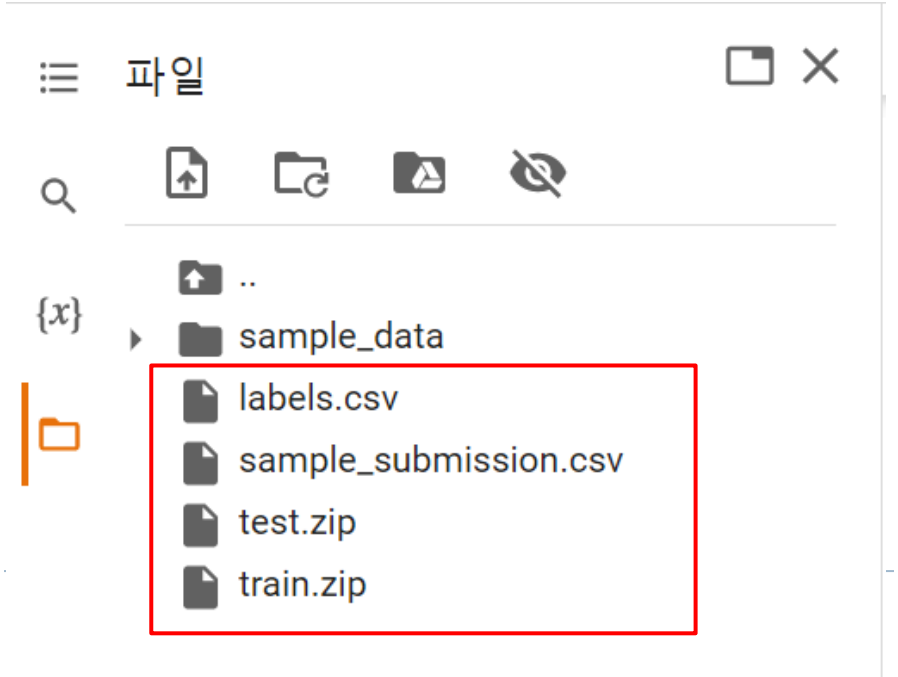
전이 학습

- 모델의 일부를 재학습시키기
  - Stanford Dog Dataset를 불러오기

```
# 8.8 Stanford Dog Dataset을 Kaggle에서 불러오기

# 2020.02.01 현재 kaggle의 Stanford Dog Dataset 파일 구조가 변경되었습니다.
# kaggle API를 사용하는 대신에 아래 링크에서 파일을 직접 받아오도록 수정되었습니다.
tf.keras.utils.get_file('/content/labels.csv', 'http://bit.ly/2GDxsYS')
tf.keras.utils.get_file('/content/sample_submission.csv', 'http://bit.ly/2GGnMNd')
tf.keras.utils.get_file('/content/train.zip', 'http://bit.ly/31nIyel')
tf.keras.utils.get_file('/content/test.zip', 'http://bit.ly/2GHEsnO')
```

전이 학습

- 모델의 일부를 재학습시키기
  - 다운로드가 완료되면 구글 코랩 좌측 상단에 있는 파일 메뉴를 클릭해서 열어봅니다.
  - 보통 구글 코랩에서 사용자의 작업 디렉터리는 /content 폴더이고, 사용자가 올리거나 외부에서 내려받는 파일은 이곳에 들어가게 됩니다.
  - 그림에서 새로 추가된 네 개의 파일을 확인할 수 있습니다.



전이 학습

- 모델의 일부를 재학습시키기
  - labels.csv.zip에는 훈련 데이터의 실제 분류값이 저장돼 있습니다.
  - sample_submission.csv.zip은 테스트 데이터를 예측한 값을 쓰기 위한 양식 파일입니다.
  - 이 양식 파일을 캐글에 올려서 점수를 평가받게 됩니다.
  - test.zip과 train.zip에는 테스트 데이터와 훈련 데이터에 해당하는 이미지 파일이 저장돼 있습니다.
  - 모두zip 파일로 압축돼 있기 때문에 먼저 압축을 풀겠습니다.
  - 다음과 같은간단한 unzip 명령어로 각 파일의 압축을 풀 수 있습니다
  - 여기서는 train.zip과 labels.csv zip의 압축만 풀어보겠습니다.

```
!unzip train.zip
!unzip labels.csv.zip
```

전이 학습

- 모델의 일부를 재학습시키기
  - 이제 정답 라벨을 담고 있는 csv 파일의 내용을 살펴보겠습니다.

```
import pandas as pd

label_text = pd.read_csv('labels.csv')
print(label_text.head())
```

  - 판다스의 pd.read_csv() 함수를 이용해 csv 파일을 데이터프레임으로 불러올 수 있습니다.
  - id 컬럼에 있는 값은 각 사진 파일의 이름입니다.
  - Id가 파일 이름인지 확인해 보기 위해서는 왼쪽에 생긴 train 폴더를 알아보면 됩니다.
  - breed 칼럼에는 각 사진이 어느 견종인지 분류돼 있습니다.

전이 학습

- 모델의 일부를 재학습시키기
  - 좀 더 요약된 정보를 보기 위해 다음 예제에서는 info() 함수를 실행합니다.

```
label_text.info()
```

  - 총 10,222장의 사진이 훈련 데이터에 포함돼 있음을 확인할 수 있습니다.
  - 그럼 이 사진들은 몇 개의 견종으로 분류할 수 있을까요?
  - 판다스에서는 nunique( )라는 함수를 사용하면 해당 값의 겹치지 않는 숫자를 구할 수 있습니다.

```
label_text['breed'].nunique()
```

전이 학습

- 모델의 일부를 재학습시키기
  - 견종에 대한 정보만 궁금하기 때문에 breed 컬럼만 선택해서 nunique() 함수를 사용한 결과, 120이라는 결괏값을 얻었습니다.
  - 이는 처음에 소개했던 데이터의 정보와 일치합니다.
  - 훈련 데이터에는 전체 데이터에 존재하는 모든 견종이 다 존재하고 있다고 생각할 수 있습니다.
  - 실제로 어떤 사진들로 구성돼 있는지 이미지와 라벨을 함께 출력해서 확인해보겠습니다

전이 학습

- 모델의 일부를 재학습시키기
  - 이미지 확인

```
# 8.13 이미지 확인
import PIL.Image as Image
import matplotlib.pyplot as plt

plt.figure(figsize=(12,12))
for c in range(9):
    image_id = label_text.loc[c, 'id']
    plt.subplot(3,3,c+1)
    plt.imshow(plt.imread('/content/train/' + image_id + '.jpg'))
    plt.title(str(c) + ', ' + label_text.loc[c, 'breed'])
    plt.axis('off')
plt.show()
```

전이 학습

- 모델의 일부를 재학습시키기
  - 각 견종은 섹깔, 무늬, 털의 길이, 귀의 크기 등 여러 가지 차이점이 있고, 사진은 정면, 측면, 후방 등 다양한 각도에서 찍혔습니다.
  - 이 문제가 얼마나 어려운지 판단해보기 위해, 전이 학습을 사용하기 전에 먼저 MobileNet V2의 모든 레이어의 가중치를 초기화한 상태에서 학습을 시켜보겠습니다.
  - 레이어 구조는 같지만 ImageNet의 데이터로 미리 훈련된 이미지 분류에 대한 지식은 전혀 없는 상태에서 학습시켜보는 것입니다.
  - 앞 절에서 소개한 텐서플로 허브를 이용하는 방법 외에 tf.keras에서도 MobileNet V2를 불러올 수 있습니다.
  - KerasLayer로 불러오는 방법과 딜리 네트워크를 각 레이어별로 분리해서 사용할 수 있기 때문에 좀더 편리합니다.

```
# 8.14 tf.keras에서 MobileNetV2 불러오기
from tensorflow.keras.applications import MobileNetV2
mobilev2 = MobileNetV2()
```

전이 학습

- 모델의 일부를 재학습시키기
  - 다음 예제에서는 메모리에 불러온 네트워크의 가중치를 초기화합니다.

```
# 8.15 MobileNetV2의 가중치 초기화
for layer in mobilev2.layers[:-1]:
    layer.trainable = True

for layer in mobilev2.layers[:-1]:
    if 'kernel' in layer.__dict__:
        kernel_shape = np.array(layer.get_weights()).shape
        # weight를 평균이 0, 표준편차가 1인 random 변수로 초기화
        layer.set_weights(tf.random.normal(kernel_shape, 0, 1))
```

  - 첫 번째 for 문에서는 각 레이어의 훈련 가능 여부를 모두 True로 바꿉니다.
  - 다만 마지막 레이어인 소프트맥스 Dense 층은 사용하지 않을 것이기 때문에 mobilev2.1ayers[:-1] 명령으로 제외합니다.

전이 학습

- 모델의 일부를 재학습시키기
  - 두 번째 for 문에서는 각 레이어에 kernel이 있는지를 확인합니다.
  - 뉴런에는 가중치와 편향이 있습니다.
  - 이 때 kernel은 가중치 w에 해당하고, bias는 편향 b입니다.
  - bias는 MobileNet V2에 존재하지 않기 때문에 kernel이 있는지만 감시해서 있을 경우 그 값을 모두 random 변수로 초기화합니다 .
  - 초기화가 끝나면 이제 실제로 학습시켜 볼 차례입니다.
  - 학습을 시키기 위해 훈련 데이터를 준비해보겠습니다.
  - 다음 예제 코드는 훈련 데이터의 모든 사진을 메모리에 올립니다.
  - 여기서 주의해야 할 점은 구글 코랩은 고용량 램 모드로 사용할 경우 약 25GB의 메모리를 사용할 수 있는데, 이 훈련 데이터를 모두 메모리에 올릴 경우 그 절반 정도를 사용하게 됩니다.
  - 테스트 데이터까지 동시에 올릴 경우 사용 기능한 메모리를 모두 사용했다는 메시지와 함께 코랩이 다운됩니다.

전이 학습

- 모델의 일부를 재학습시키기
  - 메모리가 부족할 때도 학습을 시키기 위해서는 ImageDataGenerator 등을 사용해 파일을 디스크에서 읽어와서 학습시킬 수 있지만 속도가 매우 느립니다.
  - 지금은 일단 메모리 위에 훈련 데이터를 올려서 학습시키겠습니다.

```
# 8.16 train 데이터를 메모리에 로드
import cv2

train_X = []
for i in range(len(label_text)):
    img = cv2.imread('/content/train/' + label_text['id'][i] + '.jpg')
    img = cv2.resize(img, dsize=(224, 224))
    img = img / 255.0
    train_X.append(img)
train_X = np.array(train_X)
print(train_X.shape)
print(train_X.size * train_X.itemsize, ' bytes')
```

전이 학습

- 모델의 일부를 재학습시키기
  - 예제 8.16은 고용량 RAM 모드를 지원하지 않는 무료 버전의 경우 OOM(Out Of Memory) 문제를 일으키기 때문에 학습이 진행되지 않습니다.
  - 따라서, ImageDataGenerator 등을 사용해보겠습니다.

```
# 8.16 train 데이터를 메모리에 로드
# ImageDataGenerator가 처리할 수 있는 하위 디렉토리 구조로 데이터 복사
import os
import shutil

os.mkdir('/content/train_sub')

for i in range(len(label_text)):
    if os.path.exists('/content/train_sub/' + label_text.loc[i]['breed']) == False:
        os.mkdir('/content/train_sub/' + label_text.loc[i]['breed'])
    shutil.copy('/content/train/' + label_text.loc[i]['id'] + '.jpg',
'/content/train_sub/' + label_text.loc[i]['breed'])
```

전이 학습

- 모델의 일부를 재학습시키기
  - 예제 8.16은 고용량 RAM 모드를 지원하지 않는 무료 버전의 경우 OOM(Out Of Memory) 문제를 일으키기 때문에 학습이 진행되지 않습니다.
  - 따라서, ImageDataGenerator 등을 사용해보겠습니다.

```
# ImageDataGenerator를 이용한 train/validation 데이터 분리, Image Augmentation
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.inception_resnet_v2 import preprocess_input

image_size = 224 # 이미지 사이즈가 299에서 224로 바뀌었습니다.
batch_size = 32

train_datagen = ImageDataGenerator(rescale=1./255., horizontal_flip=True, shear_range=0.2, zoom_range=0.2, width_shift_range=0.2, height_shift_range=0.2, validation_split=0.25)
valid_datagen = ImageDataGenerator(rescale=1./255., validation_split=0.25)
```

전이 학습

- 모델의 일부를 재학습시키기
  - 예제 8.16은 고용량 RAM 모드를 지원하지 않는 무료 버전의 경우 OOM(Out Of Memory) 문제를 일으키기 때문에 학습이 진행되지 않습니다.
  - 따라서, ImageDataGenerator 등을 사용해보겠습니다.

```
train_generator = train_datagen.flow_from_directory(directory="/content/train_sub/", subset="training", batch_size=batch_size, seed=42, shuffle=True, class_mode="categorical", target_size=(image_size, image_size))
valid_generator = valid_datagen.flow_from_directory(directory="/content/train_sub/", subset="validation", batch_size=1, seed=42, shuffle=True, class_mode="categorical", target_size=(image_size, image_size))
```

전이 학습

- 모델의 일부를 재학습시키기
  - 그 다음으로는 Y에 해당하는 라벨 데이터를 작성합니다.

```
# 8.17 train 라벨 데이터를 메모리에 로드
unique_Y = label_text['breed'].unique().tolist()
train_Y = [unique_Y.index(breed) for breed in label_text['breed']]
train_Y = np.array(train_Y)

print(train_Y[:10])
print(train_Y[-10:])
```

  - 여기서 라벨 데이터는 'boston_bu11', 'dingo'와 같은 텍스트로 돼 있기 때문에 먼저 숫자로 바꾸는 과정이 필요합니다.
  - 첫 줄에서는 nunique()와 비슷한 기능을 하는 unique() 함수를 사용해 label_text[' breed ']를 구성하는 겹치지 않는 유일한 원소들을 구하고, 그것을 넘파이 array에서 파이썬 리스트로 변환합니다.

전이 학습

- 모델의 일부를 재학습시키기
  - 2번째 줄에서는 이 리스트를 이용해 전체 train_Y를 숫자로 만들고, 학습을 위해 다시 넘파이 array로 바꿉니다.
  - 출력에서는 train_Y의 처음 값 10개와 마지막 값 10개를 확인합니다.
  - 이제는 전이학습을 위한 네트워크를 만들어야 합니다.
  - 엄밀히 말하면 이번에는 랜덤한 기중치를 사용하기 때문에 가져오는 지식이 없어서 전이 학습이라고 하기는 힘들지만 바로 다음 예제에서 미리 훈련된 가중치를 사용해 학습시킬 것입니다.
  - 다음 예제에서는 전이 학습을 위한 모델을 정의합니다.

전이 학습

- 모델의 일부를 재학습시키기
  - Dogs Dataset 학습을 위한 모델 정의

```
# 8.18 Dogs Dataset 학습을 위한 모델 정의
x = mobilev2.layers[-2].output
predictions = tf.keras.layers.Dense(120, activation='softmax')(x)
model = tf.keras.Model(inputs=mobilev2.input, outputs=predictions)

model.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy']) # 라벨이 원-핫 인코딩을 사용하기 때문에 sparse가 아닌 categorical_crossentropy를 사용합니다.
model.summary()
```

  - 첫 번째 줄에서는MobileNet V2에서 마지막 Dense 레이어를 제외하기 위해 뒤에서 두 번째 레이어를 지정해서 그 레이어의 outpu을 x라는 변수에 저장했습니다.

전이 학습

- 모델의 일부를 재학습시키기
  - 그 다음에는 Dogs Dataset의 분류를 위해 120개의 뉴런을 가진 Dense 레이어를 새롭게 만들었습니다

```
predictions = tf.keras.layers.Dense(120, activation='softmax')(x)
```

  - 그런데 이 문장은 처음 나온 구문입니다.
  - 지금까지는 tf.keras.Sequential 모델만 사용해왔습니다.
  - 각 레이어는 모델 정의 안에 들어가 있었고 밖으로 나온 적이 없었습니다.
  - 반면 위에서는 레이어가 함수처럼 밖에 나오고 x라는 인수를 받습니다.
  - 그 계산은 predictions라는 변수에 다시 저장됩니다.

전이 학습

- 모델의 일부를 재학습시키기
  - 이처럼 레이어를 함수처럼 사용하는 구문을 케라스의 함수형 (functional) API라고 합니다.
  - 입력부터 출력까지 일직선의 구조라면 시퀀셜 모델을 사용해도 되지만, 그렇지 않은 경우에는 함수형 모델을 사용해 모델의 입력, 출력 구조를 정의해야 합니다.
  - 함수형 모델을 정의하기 위해서는 연결에 필요한 모든 레이어를 준비한 다음에 tf.keras.Model () 의 인수인 inputs, outputs에 각각 입력과 출력에 해당하는 부분을 넣으면 tf.keras가 그 사이의 연결을 알아서 찾아서 모델을 만듭니다.
  - 물론 입력과 출력 사이에 연결이 되지 않을 경우에는 에러가 발생합니다.

```
model = tf.keras.Model(inputs=mobilev2.input, outputs=predictions
```

전이 학습

- 모델의 일부를 재학습시키기
  - 그럼 이제 모델을 실제로 학습시켜보겠습니다.

```
# 8.19 랜덤한 가중치를 가진 상태에서의 학습
# history = model.fit(train_X, train_Y, epochs=10, validation_split=0.25, batch_size=32)
steps_per_epoch = int(7718/32) # generator를 사용하기 때문에 1epoch 당 학습할 step수를 정합니다. batch_size인 32로 train_data의 크기를 나눠주면 됩니다.
history = model.fit_generator(train_generator, validation_data=valid_generator, epochs=10, steps_per_epoch=steps_per_epoch) # model.fit_generator() 를 사용합니다.
```

전이 학습

- 모델의 일부를 재학습시키기
  - 결과는 좋지 않습니다.
  - 훈련 데이터의 정확도인 accuracy는 조금 나아지고 있지만 그중 검증을 위해 따로 분리한 검증 데이터의 정확도인 val_accurac는 학습의 시작과 끝부분에서 거의 나아지지 않은 모습을 보입니다.
  - 견종의 수가 120이기 때문에 랜덤하게 120개 중에 하나를 선택한다면 정답을 맞출 확률은 0.83% = 0.0083이 나올 텐데, val_accuracy에는 그에 근접한 숫자가 보이고 있습니다.

전이 학습

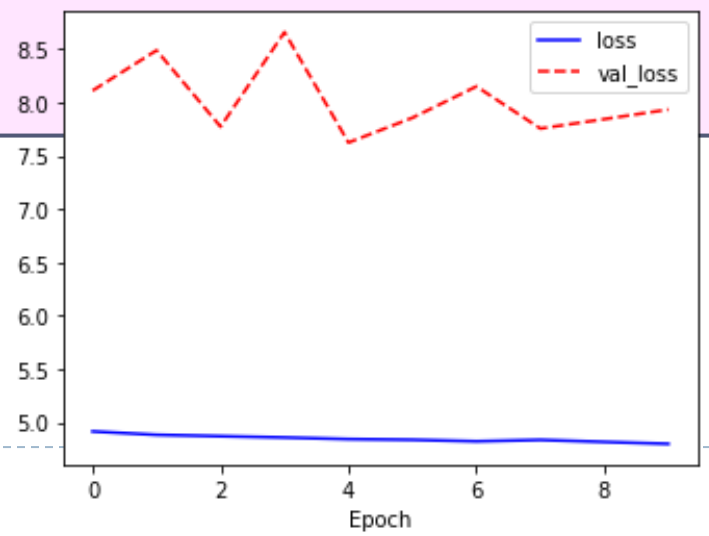
- 모델의 일부를 재학습시키기
  - 학습결과 확인

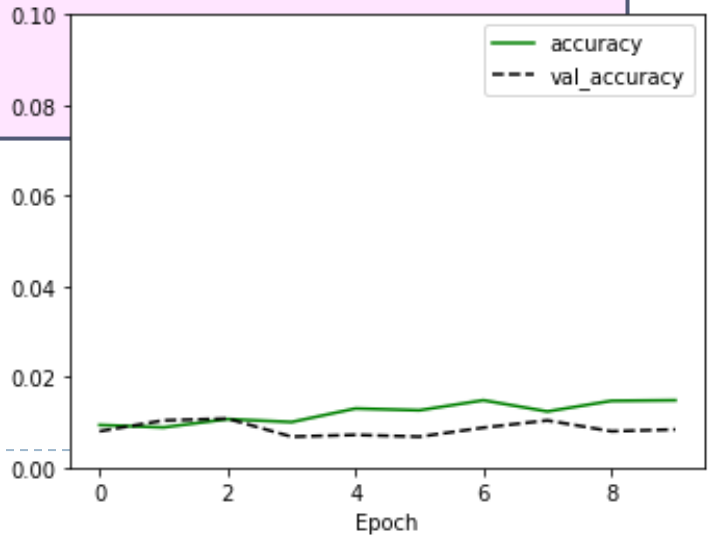
```
# 8.20 학습 결과 확인
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
```

전이 학습

- 모델의 일부를 재학습시키기
  - 학습결과 확인

```
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0, 0.1)
plt.legend()

plt.show()
```





전이 학습

- 모델의 일부를 재학습시키기
  - loss와 va1_loss 값은 4를 넘는 매우 큰 값이고, 매우 천천히 줄어들고 있습니다.
  - 정확도도 천천히 증가하고 있지만 학습이 잘 되고 있다고 말하기는 어려운 그래프입니다.
  - 그럼 같은 네트워크로 전이 학습을 시도해보겠습니다.
  - 예제 8.18과 동일한 네트워크 구조지만 기존의 가중치를 그대로 사용하고 일부 레이어의 가중치를 고정시킨 상태로 학습시킬 것입니다.

전이 학습

- 모델의 일부를 재학습시키기
  - Dogs Dataset 학습을 위한 전이 학습 모델 정의

```
# 8.21 Dogs Dataset 학습을 위한 Transfer Learning 모델 정의
from tensorflow.keras.applications import MobileNetV2
mobilev2 = MobileNetV2()

x = mobilev2.layers[-2].output
predictions = tf.keras.layers.Dense(120, activation='softmax')(x)
model = tf.keras.Model(inputs=mobilev2.input, outputs=predictions)

# 뒤에서 20개까지의 레이어는 훈련 가능, 나머지는 가중치 고정
for layer in model.layers[:-20]:
    layer.trainable = False
for layer in model.layers[-20:]:
    layer.trainable = True
```

전이 학습

- 모델의 일부를 재학습시키기
  - Dogs Dataset 학습을 위한 전이 학습 모델 정의

```
# model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy']) # 라벨이 원-핫 인코딩을 사용하기 때문에 sparse가 아닌 categorical_crossentropy를 사용합니다.
model.summary()
```

  - 함수형 API를 이용해 모델을 정의하는 과정은 예제 8.18과 같습니다.
  - 그리고 뒤에서 20개까지의 레이어를 훈련 가능하게 하고 나머지 레이어의 가중치는 고정시키는 작업을 합니다.
  - model.summary() 의 출력 결과에서 훈련 가능한 가중치의 수와 고정된 값의 가증치가 반반 정도로 비슷해진 것을 확인할 수 있습니다.
- 그럼 학습을 시키고 그래프로 결과를 확인해보겠습니다

전이 학습

- 모델의 일부를 재학습시키기
  - 모델 학습 및 결과 확인

```
# 8.22 모델 학습 및 결과 확인
# history = model.fit(train_X, train_Y, epochs=10, validation_split=0.25, batch_size=32)
steps_per_epoch = int(7718/32) # generator를 사용하기 때문에 1epoch 당 학습할 step수를 정합니다. batch_size인 32로 train_data의 크기를 나눠주면 됩니다.
history = model.fit_generator(train_generator, validation_data=valid_generator, epochs=10, steps_per_epoch=steps_per_epoch) # model.fit_generator()를 사용합니다.

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
```

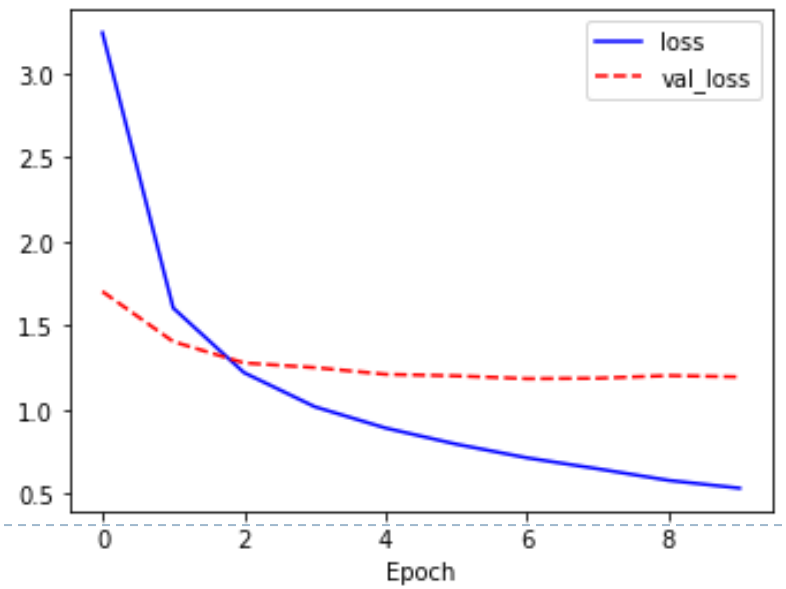
전이 학습

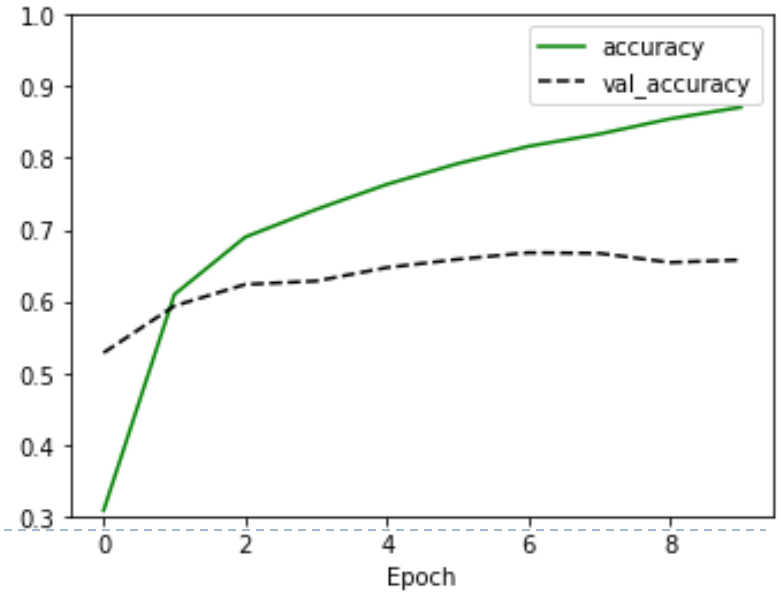
- 모델의 일부를 재학습시키기
  - 모델 학습 및 결과 확인

```
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.3, 1)
plt.legend()

plt.show()
```

전이 학습

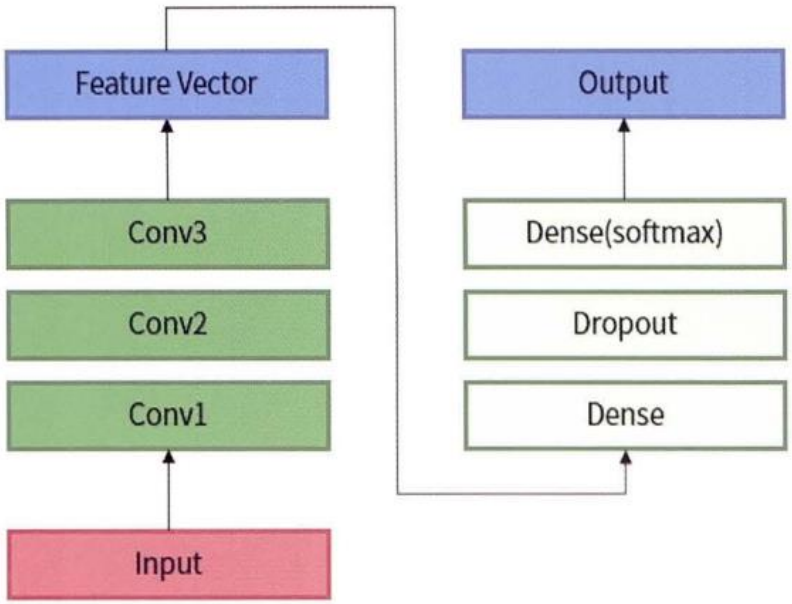
- 모델의 일부를 재학습시키기
  - 같은 네트워크 구조를 사용했지만 va1_accuracy는 56.73%로 1% 정도에 머물던 위의 예제와는 전혀 다른 결과기 나왔습니다.
  - 특히 val_loss는 감소하는 추세이고 va1_accuracy는 증가 추세여서 학습을 좀 더 시켜도 네트워크의 성능이 향상될 것으로 기대됩니다.
  - 또 학습시킬 가중치의 숫자가 줄어들었기 때문에 학습속도도 평균 42초에서 13초로 약 1/3이 된 것을 확인할 수 있습니다.





전이 학습

- 특징 추출기
  - 미리 훈련된 모델에서 데이터의 특징만 추출하고, 그 특징을 작은 네트워크에 통과시켜 정답을 예측하는 방법도 있습니다.
  - 미리 추출된 특징은 파일로 저장했다가 필요할 때 불러서 사용합니다.
  - 학습할 때마다 전체 네트워크의 계산을 반복할 필요가 없기 때문에 계산 시간은 획기적으로 줄어들고, 메모리도 절약됩니다.



전이 학습

- 특징 추출기
  - 텐서플로 허브에서 Inception v3를 불러오겠습니다.
  - Inception은 2014년에 구글이 ImageNet 대회를 위해 GoogLeNet이라는 이름으로 발표한 CNN으로, V3은 세번째로 개선된 버전입니다.
  - 텐서플로 허브에는 소프트맥스 Dense 레이어를 포함한 전체 네트워크와 소프트맥스 Dense 레이어가 없는 특징 추출기 네트워크가 있는데, 여기서는 특징 추출기 네트워크를 KerasLayer로 불러오겠습니다.

전이 학습

- 특징 추출기
  - 텐서플로 허브에서 사전 훈련된 Inception V3의 특징 추출기 불러오기

```
# 8.23 텐서플로우 허브에서 사전 훈련된 Inception V3의 특징 추출기 불러오기
import tensorflow_hub as hub

inception_url = 'https://tfhub.dev/google/tf2-preview/inception_v3/feature_vector/4'
feature_model = tf.keras.Sequential([
    hub.KerasLayer(inception_url, output_shape=(2048,), trainable=False)
])
feature_model.build([None, 299, 299, 3])
feature_model.summary()
```

전이 학습

- 특징 추출기
  - 1 절과 마찬가지로 KerasLayel 형식으로 모델 안에 들어가는 하나의 레이어로 전체 네트워크를 불러 올 수 있습니다.
  - Inception V3에서 마지막 Dense 레이어에 넘겨주는 출력의 크기는 2048이고 여기서 불러오는 특징 추출기는 마지막 Dense 레이어를 포함하지 않기 때문에 output_shape=(2048,)로 출력 크기를 지정합니다.
  - 이 설정에 의해 lnception V3 특징 추출기는 이미지에서 2048 크기의 특징 벡터 (Feature Vector)를 추출할 수 있습니다.
  - summary() 함수로 모델을 확인하기 위해서는 보통 compile( ) 함수를 먼저 실행해야 하지만 7번째 줄처럼 build () 함수를 사용할 수도 있습니다.
  - 네트워크의 input_shape이 특정되지 않았기 때문에 build () 함수의 인수로 [None , 299 , 299 , 3] 이라는 입력 데이터의 차원을 정의해서 넣습니다.

전이 학습

- 특징 추출기
  - 첫 번째 차원은 배치 차원이기 때문에 입력이 몇 개가 들어와도 상관없습니다.
  - None으로 설정하면 어떤 숫자도 받을 수 있도록 제한을 두지 않는다는 뜻이 됩니다.
  - 두 번째와 세 번째의 299는 이미지의 높이와 너비를 의미합니다.
  - 네 번째의 3은 컬러 이미지를 받을 것이기 때문에 RGB 차원 수인 3이 됩니다.
  - Inception V3 네트워크의 피라미터 수는 MobileNet V2의 350만 개보다 훨씬 많은 2,100만 개입니다.
  - 이미지의 입력 크기도 MobileNet V2의 기본 입력 크기인 224 대신에 299를 받고 있습니다.

전이 학습

- 특징 추출기
  - 학습을 위해 이미지의 가로 세로 크기를 224로 조정하는 대신에 299로 조정해야 하지만, 훈련 데이터 전체를 299로 조정해서 메모리에 올리면 메모리 부족으로 코랩이 다운돼 버립니다.
  - 이를 방지하기 위해서는 이미지 보강에서 나왔던 ImageDataGenerator를 사용해 이미지를 한꺼번에 메모리에 올리는 것이 아니라 필요할 때마다 디스크에서 배치 크기만큼 조금씩 읽어 들이는 방법을 사용할 수 있습니다.
  - 그런데 ImageDataGenerator는 라벨이 있는 데이터를 처리할 때 각 라벨의 이름을 하위 디렉터리로 가지고 있는 디렉터리를 받아서 그 데이터를 처리합니다.
  - 반면 캐글에서 내려받은 데이터들은 하위 디렉터리의 구분 없이 train 폴더에 모든 이미지 하일이 저장돼 있습니다.
  - 따라서 ImageDataGenerator가 처리할 수 있는 방식으로 데이터를 복사하는 작업이 필요합니다.

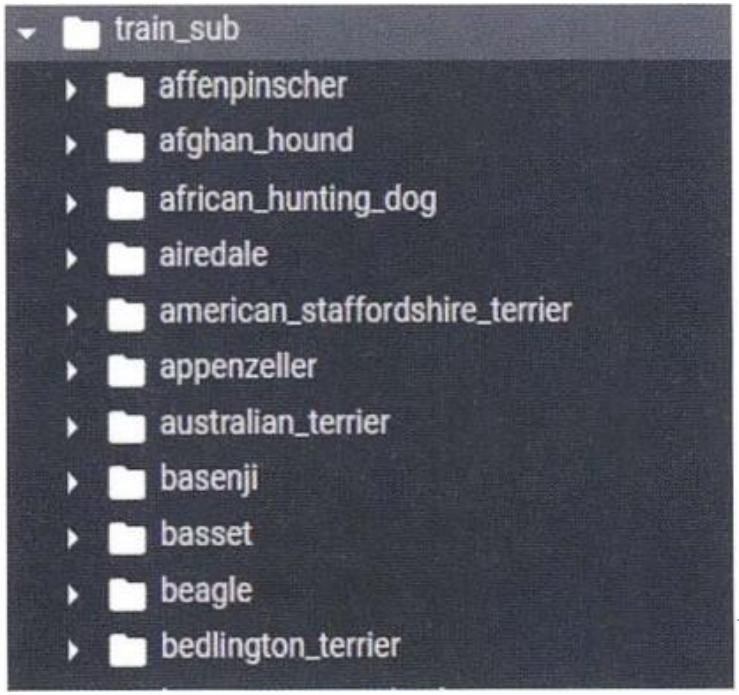
전이 학습

- 특징 추출기
  - ImageDataGenerator가 처리할 수 있는 하위 디렉터리 구조로 데이터를 복사

```
# 8.24 ImageDataGenerator가 처리할 수 있는 하위 디렉토리 구조로 데이터 복사
import os
import shutil

os.mkdir('/content/train_sub')

for i in range(len(label_text)):
    if os.path.exists('/content/train_sub/' + label_text.loc[i]['breed']) == False:
        os.mkdir('/content/train_sub/' + label_text.loc[i]['breed'])
    shutil.copy('/content/train/' + label_text.loc[i]['id'] + '.jpg',
'/content/train_sub/' + label_text.loc[i]['breed'])
```

전이 학습

- 특징 추출기
  - 먼저 content 폴더 아래에 train_sub라는 디렉터리를 만든 다음, 그 아래에 label_text의 각 데이터의 breed에 해당하는 하위 디렉터리를 만들고 해당 파일을 복사합니다.
  - 복사를 끝내면 그림처럼 train_sub 폴더 아래에 각breed에 대한 하위 디렉터리가 생깁니다.



전이 학습

- 특징 추출기
  - 이제 ImageDataGenerator를 통해 train_sub 디렉터리의 이미지들을 이용해 훈련 데이터와 검증 데이터를 분리하고, 훈련 데이터에는 이미지 보강(Image Augmentation)을 적용해 네트워크의 성능을 높여보겠습니다.

전이 학습

- 특징 추출기
  - ImageDataGenerator를 이용한 훈련 및 검증 데이터 분리, 이미지 보강

```
# 8.25 ImageDataGenerator를 이용한 train/validation 데이터 분리, Image Augmentation
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from keras.applications.inception_resnet_v2 import preprocess_input

image_size = 299
batch_size = 32

train_datagen = ImageDataGenerator(rescale=1./255., horizontal_flip=True, shear_range=0.2, zoom_range=0.2, width_shift_range=0.2, height_shift_range=0.2, validation_split=0.25)
valid_datagen = ImageDataGenerator(rescale=1./255., validation_split=0.25)
```

전이 학습

- 특징 추출기
  - ImageDataGenerator를 이용한 훈련 및 검증 데이터 분리, 이미지 보강

```
train_generator = train_datagen.flow_from_directory(directory="/content/train_sub/", subset="training", batch_size=batch_size, seed=42, shuffle=True, class_mode="categorical", target_size=(image_size, image_size))
valid_generator = valid_datagen.flow_from_directory(directory="/content/train_sub/", subset="validation", batch_size=1, seed=42, shuffle=True, class_mode="categorical", target_size=(image_size, image_size))
```

전이 학습

- 특징 추출기
  - 먼저 ImageDataGenerator를 생성합니다.
  - 그런데 train_datagen과 valid_datagen의 2개를 각각 생성해야 합니다.
  - train_datagen은 픽셀 정규화(1/255로 rescale), 좌우 반전, 기울이기, 줌, 죄우/상하 평행 이동을 수행하는 ImageDataGenerator이고, valid_datagen은 픽셀 정규화만 실행합니다.
  - 양쪽 모두 validation_split을 0. 25로 지정해서 25%의 데이터는 검증 데이터로 뺍니다.
  - 그 다음에 양쪽의 ImageDataGenerator 를 각각 사용하는 train_generator, valid_generator를 flow_from_directory() 함수를 사용해서 만듭니다.

전이 학습

- 특징 추출기
  - 이렇게 ImageDataGenerator 2개를 사용하는 이유는 네트워크를 학습시킬 훈련 데이터에는 이미지 보강을 위한 ImageDataGenerator를 사용하고, 검증 데이터에는 별도의 이미지 보강 없이 원본 데이터를 그대로 사용하기 위해서입니다.
  - 양쪽 train_generator와 valid_generator의 seed 값을 같은 값으로 맞춰주기 때문에 두 ImageDataGenerator에는 동일한 training, validation 부분집합이 생깁니다.
  - 이 가운데 train generator는 training 부분집합을 사용하고 valid_generator는 validation 부분집합을 사용합니다.
  - 이제 위에서 불러왔던 lnception V3 특징 추출기를 이용해 훈련 데이터와 검증 데이터를 특징 벡터로 변환하는 작업이 필요합니다.
  - 이렇게 변환한 특징 벡터를 작은 시퀀셜 모델에 넣어서 실제 라벨을 예측한 것입니다.

전이 학습

- 특징 추출기
  - 훈련 데이터를 특징 벡터로 변환

```
# 8.26 train 데이터를 특징 벡터로 변환
batch_step = (7718 * 3) // batch_size
train_features = [ ]
train_Y = [ ]
for idx in range(batch_step):
    if idx % 100 == 0:
        print(idx)
    x, y = train_generator.next()
    train_Y.extend(y)

    feature = feature_model.predict(x)
    train_features.extend(feature)

train_features = np.array(train_features)
train_Y = np.array(train_Y)
print(train_features.shape)
print(train_Y.shape)
```

전이 학습

- 특징 추출기
  - 첫 줄에서는 batch_step을 지정합니다.
  - training 부분집합의 크기인 7718에 3을 곱해서 충분한 이미지 보강이 되게 하고, batch_size(32)로 나눠서 training 부분집합을 3번 정도 반복해서 특징 벡터를 뽑아냅니다.
  - ImageDataGenerator에 next( ) 함수를 사용하면 다음에 올 값을 반환받을 수 있습니다.
  - 훈련 데이터에는 이미지의 분류에 해당하는 y 값이 있기 때문에 식의 좌변에서 x, y를 함께 받게 됩니다.
  - y값은 바로 train_Y에 저장해서 뒤에서 쓸 수 있게 합니다.

```
x, y = train_generator.next()
train_Y.extend(y)
```

전이 학습

- 특징 추출기
  - x 값은 이미지 데이터에 해당하는 부분으로, 특징 추출기를 통과하면 특징 벡터가 됩니다.
  - 이미 학습이 완료된 특징 추출기를 사용하기 때문에 predict()로 특징 벡터를 추출합니다.
  - 특징 벡터는 feature라는 변수에 저장한 뒤 역시 뒤에서 쓸 수 있게 train_features에 저장합니다

```
feature = feature_model.predict(x)
train_features.extend(feature)
```

전이 학습

- 특징 추출기
  - 최종 출력되는 값의 Shape은 train_features가 (23084, 2048) 이고 train Y가 (23084, 120) 입니다.
  - 특징 벡터는 2,048차원의 벡터인 것을 알 수 있습니다.
  - 가로 세로 299 픽셀, RGB 3차원의 이미지가 2048개의 실수로 변환됐으니 데이터의 크기는 원본에 비해 2048/(299 x 299 x 3) x 100=0.76%로 크게 줄어들었습니다.
  - 검증 데이터를 특징 벡터로 변환하는 과정은 다음과 같습니다.

전이 학습

- 특징 추출기
  - 검증 데이터를 특징 벡터로 변환

```
# 8.27 validation 데이터를 특징 벡터로 변환
valid_features = []
valid_Y = []

for idx in range(valid_generator.n):
    if idx % 100 == 0:
        print(idx)
    x, y = valid_generator.next()
    valid_Y.extend(y)

    feature = feature_model.predict(x)
    valid_features.extend(feature)

valid_features = np.array(valid_features)
valid_Y = np.array(valid_Y)
print(valid_features.shape)
print(valid_Y.shape)
```

전이 학습

- 특징 추출기
  - 검증 데이터는 한 번만 계산하기 위해 for 문의 range ( )안에 ImageDataGenerator의 데이터의 크기를 나타내는 valid_generator.n을 넣었습니다.
  - 그 밖에는 예제 8.26과 거의 같습니다.
  - 이제 특징 벡터가 준비됐으니 분류를 위한 네트워크를 만들어 보겠습니다.
  - 작은 시퀀설 모델이면 충분합니다.

전이 학습

- 특징 추출기
  - 분류를 위한 작은 시퀀셜 모델 정의

```
# 8.28 분류를 위한 작은 Sequential 모델 정의
model = tf.keras.Sequential([
    tf.keras.layers.Dense(256, activation='relu', input_shape=(2048,)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(120, activation='softmax')
])

model.compile(tf.optimizers.RMSprop(0.0001),  loss='categorical_crossentropy',
metrics=['accuracy'])
model.summary()
```

전이 학습

- 특징 추출기
  - 첫 번째 Dense 레이어에 input_shape=(2048,)을 지정해서 특징 벡터를 믿을 수 있게 했습니다.
  - 마지막 Dense 레이어는 분류를 위해 softmax 활성화 함수를 저장하고 뉴런의 수는 견종의 수와 같은 120으로 지정했습니다.
  - 손실에는 앞 절에서 사용했던 sparse_categorical_crossentropy 가 아닌 categorical_crossentropy를 사용했는데 train_Y의 마지막 차원이 1이 아닌 원-핫 펙터인 120 이기 때문입니다.
  - 정답의 인덱스만 기록된 희소 행렬 (sparse matrix) 을 Y로 사용할 때 는 sparse를 넣은 categorical crossentropy를 사용하고 one-hot 벡터를 사용할 때는 sparse가 없는 버전을 사용하면 됩니다.

전이 학습

- 특징 추출기
  - 그럼 지금까지 해왔던 것처럼 모델을 학습시키고 그래프로 결과를 확인해 보겠습니다.

```
# 8.29 분류를 위한 작은 Sequential 모델 학습
history = model.fit(train_features, train_Y, validation_data=(valid_features, valid_Y), epochs=10, batch_size=32)

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
```

전이 학습

- 특징 추출기
  - 그럼 지금까지 해왔던 것처럼 모델을 학습시키고 그래프로 결과를 확인해 보겠습니다.

```
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.8, 1)
plt.legend()

plt.show()
```

전이 학습

- 특징 추출기
  - model.fit()의 인수로 train features를 넣었고 검증을 위한 validation_data를 직접 지정하고 여기에도 valid_ features를 넣어서 특징 추출기의 출력을 직접 사용하고 있습니다.
  - 이미지 보강과 크기가 커진 이미지 데이터를 사용해서인지 학습은 전반적으로 잘 되는 모습이고 val_accuracy는 90%를 넘기고 있습니다.
  - 앞 절의 55 ~56%에 비교하면 크게 향상된 수치입니다.
  - 각 에포크당 약 3초가 걸릴 정도로 학습 속도는 매우 빨리졌습니다.
  - 위에서 MobileNet V2를 이용한 모델 재학습은 에포크당 42 초, 가중치의 일부를 학습시키지 않도록 얼렸을 때는 에포크당 13초가 걸렸으니 처음과 비교하면 1 /13로 실행 시간이 줄어들었습니다.
  - 특징 추출기를 사용하면 더 많은 파라미터와 큰 이미지를 사용하면서도 학습 속도를 획기적으로 줄일 수 있습니다.

전이 학습

- 특징 추출기
  - 모델이 예측을 얼마나 잘하는지 알아보기 위해 검증 데이터의 이미지에 대한 분류를 시각화해보겠습니다.
  - 그 전에 먼저 라벨의 이름을 따로 저장해야 합니다.
  - 앞 절에서 견종의 이름을 지정한 unique_y를 만들었지만 이것은 csv 파일 에서 나오는 순서대로 인덱스를 저장한 것이었습니다.
  - 이와 달리 ImageDataGenerator에서는 라벨을 인덱스로 저장할 때 알파벳 순으로 정렬된 순서로 저장하기 때문에 여기서도 그에 맞게 unique_y를 정렬하겠습니다.

```
# 8.30 라벨 텍스트를 알파벳 순으로 정렬
unique_sorted_Y = sorted(unique_Y)
print(unique_sorted_Y)
```

전이 학습

- 특징 추출기
  - 그럼 이제 검증 데이터의 이미지에 대한 분류를 시각화해 보겠습니다.

```
# 8.31 InceptionV3 특징 추출기-Sequential 모델의 분류 라벨 확인
import random
plt.figure(figsize=(16,16))

for c in range(3):
    image_path = random.choice(valid_generator.filepaths)

    # 이미지 표시
    plt.subplot(3,2,c*2+1)
    plt.imshow(plt.imread(image_path))
    real_y = image_path.split('/')[3]
    plt.title(real_y)
    plt.axis('off')
    idx = unique_sorted_Y.index(real_y)
```

전이 학습

- 특징 추출기
  - 그럼 이제 검증 데이터의 이미지에 대한 분류를 시각화해 보겠습니다.

```
# 예측값 표시
    plt.subplot(3,2,c*2+2)
    img = cv2.imread(image_path)
    img = cv2.resize(img, dsize=(299, 299))
    img = img / 255.0
    img = np.expand_dims(img, axis=0)

    # Inception V3를 이용한 특징 벡터 추출
    feature_vector = feature_model.predict(img)

    # Sequential 모델을 이용한 예측
    prediction = model.predict(feature_vector)[0]
```

전이 학습

- 특징 추출기
  - 그럼 이제 검증 데이터의 이미지에 대한 분류를 시각화해 보겠습니다.

```
# 가장 높은 확률의 예측값 5개를 뽑음
    top_5_predict = prediction.argsort()[::-1][:5]
    labels = [unique_sorted_Y[index] for index in top_5_predict]
    color = ['gray'] * 5
    if idx in top_5_predict:
        color[top_5_predict.tolist().index(idx)] = 'green'
    color = color[::-1]
    plt.barh(range(5), prediction[top_5_predict][::-1] * 100, color=color)
    plt.yticks(range(5), labels[::-1])
```

전이 학습

- 특징 추출기
  - 결과부터 살펴보면 임의로 뽑은 검증 데이터 이미지 3개 중에서 2개는 정확하게 에측하고, 나머지 1개는 2 번째로 높은 확률로 정답을 예측했습니다.
  - 첫 번째 데이터에서 모델은 그레이트 피레니즈(Great Pyrenees), 쿠바츠(Kuvasz)를 헷갈리는 것 같은데, 실제 데이터에서 뽑은 두 견종의 사진을 그림에 표시했습니다.

great_pyrenees

kuvasz

전이 학습

- 특징 추출기
  - 두 견종의 차이는 육안으로도 판단하기 쉽지 않아 보입니다.
  - 모델이 분류를 어려워하는 것도 이해가 됩니다.
  - 이제 이 모델을 계산한 결과를 실제로 케글 competition 에 올려서 지금까지 한 번도 본 적이 없는 테스트 데이터에 대한 예측 성적이 어떻게 나오는지를 알아보겠습니다.
  - 캐글 대회에서는 테스트 데이터에 대한 정답이 주어지지 않기 때문에 모델의 성능을 최종적으로 평가하기 위해서는 캐글에 모델의 계산 결과를 csv 파일로 올렸을 때 나오는 점수로 판단해야 합니다.

전이 학습

- 특징 추출기
  - 테스트 데이터는 예제 8.8에서 미리 내려받았던 파일을 압축 해제하면 사용할 수 있습니다.
  - 그리고 예측 결과를 캐글에 올리기 위한 submission.csv 파일의 압축도 풉니다.

```
!unzip test.zip
!unzip sample_submission.csv.zip
```

전이 학습

- 특징 추출기
  - 그럼 먼저 submission.csv 파일의 내용을 확인해보겠습니다.

```
# 8.33 submission.csv 파일 내용 확인
import pandas as pd
submission = pd.read_csv('sample_submission.csv')
print(submission.head())
print()
print(submission.info())
```

  - 데이터 프레임의 첫 열은 labels.csv와 같은 id 입니다.
  - 나머지 열은 120 개의 견종의 이름이 있고, 각 id 에 대한 각 견종의 예측 값은 랜덤한 선택을 했을 때의 0 .008333으로 채워져 있습니다.
  - 이대로 캐글에 제출하면 4.78749 라는 점수를 얻게 됩니다.
  - 이 점수는 Multiclass Logloss로 산정되며 낮을수록 좋습니다.

전이 학습

- 특징 추출기
  - Multiclass logloss는 바로 범주에 대한 크로스 엔트로피 (categorical crossentropy) 입니다.
  - 우리가 사용하고 있는 시퀀셜 모델에서도 Multiclass logloss를 찾아볼 수 있습니다.
  - 모델을 컴파일할 때 loss를 categorica1_crossentropy로 설정하면 fit() 함수의 출력 창에서 훈련 데이터의 Multiclass logloss 인 loss와 검증 데이터의 Multiclass logloss 인 va1_loss를 확인할 수 있습니다.
  - 예제 8.29 의 마지막 에포크에서 loss는 0.2988, va1_loss는 0.3065였습니다.
  - Val_loss가 학습 과정에서 한 번도 보지 못한 데이터에 대한 Multiclass logloss이기 때문에 페스트 데이터에 대한 Multiclass logloss도 이와 비슷한 값이 나올 것이라고 예측해 볼 수 있습니다.
  - 물론 검증 데이터와 테스트 데이터가 아주 다른 성질과 분포를 가지고 있다면 이런 예측은 빗나갈 수도 있습니다.

전이 학습

- 특징 추출기
  - 테스트 데이터도 훈련 데이터와 마찬가지로 가로 × 세로 299 픽셀의 데이터를 사용할 것이기 때문에 ImageDataGenerator를 사용하겠습니다.
  - 그런데 ImageDataGenerator가 f1ow_from_directory( ) 함수로 이미지를 읽어 들이기 위해서는 하위 디렉터리가 꼭 필요합니다.
  - 현재 테스트 데이터는 각 사진이 어떤 범주에 속하는지 알 수 없기 때문에 unknown 이라는 디렉터리를 하나 만들고 모든 데이터를 이곳에 복사하겠습니다 .

전이 학습

- 특징 추출기
  - ImageDataGenerator가 처리할 수 있는 하위 디렉터리 구조로 데이터 복사

```
# 8.34 ImageDataGenerator가 처리할 수 있는 하위 디렉토리 구조로 데이터 복사
import os
import shutil

os.mkdir('/content/test_sub/')
os.mkdir('/content/test_sub/unknown/')

for i in range(len(submission)):
    shutil.copy('/content/test/' + submission.loc[i]['id'] + '.jpg', '/content/test_sub/unknown/')
```

전이 학습

- 특징 추출기
  - 파일 복사가 끝나면 test_sub 디렉터리 하위에 unknown이리는 디렉터리가 생기고, 모든 파일이 이 안에 들어있음을 확인할 수 있습니다.
  - 이제 테스트 데이터를 불러오는 ImageDataGenerator를 정의하겠습니다.
  - 이미지 보강 등을 할 필요가 없기 때문에 코드는훈련 데이터와 검증 데이터를 처리할 때보다 간단합니다.

전이 학습

- 특징 추출기
  - ImageDataGenerator를 이용한 테스트 데이터 불러오기

```
# 8.35 ImageDataGenerator를 이용한 test 데이터 불러오기
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator

test_datagen=ImageDataGenerator(rescale=1./255.)
test_generator=test_datagen.flow_from_directory(directory="/content/test_sub
/",batch_size=1,seed=42,shuffle=False,target_size=(299, 299))
```

전이 학습

- 특징 추출기
  - 정상적으로 ImageDataGenerator가 만들어지면 이를 이용해 특징 벡터를 추출할 수 있습니다.

```
# 8.36 test 데이터를 특징 벡터로 변환
test_features = []

for idx in range(test_generator.n):
    if idx % 100 == 0:
        print(idx)

    x, _ = test_generator.next()
    feature = feature_model.predict(x)
    test_features.extend(feature)

test_features = np.array(test_features)
print(test_features.shape)
```

전이 학습

- 특징 추출기
  - 각 이미지 데이터는 2,048의 길이를 가진 특징 벡터 로 변환됐습니다.
  - 이제 이 특징 벡터를 예제 8.29에서 학습시킨 모델에 넣어서 정답을 예측합니다.

```
# 8.37 특징 벡터로 test 데이터의 정답 예측
test_Y = model.predict(test_features, verbose=1)
```

  - model.predict() 함수는 verbose 인수의 기본 값이 0으로 설정돼 있기 때문에 진행 과정을 보기 위해서는 verbose를 1로 지정해야 합니다.
  - 실제로 예측이 잘 됐는지 시각화로 확인해보겠습니다.

전이 학습

- 특징 추출기
  - Inception V3 특징 추출기 시퀀셜 모델의 테스트 데이터 분류 라벨 확인

```
# 8.38 Inception V3 특징 추출기-Sequential 모델의 test 데이터 분류 라벨 확인
import random
plt.figure(figsize=(16,16))

for c in range(3):
    image_path = random.choice(test_generator.filepaths)

    # 이미지 표시
    plt.subplot(3,2,c*2+1)
    plt.imshow(plt.imread(image_path))
    real_y = image_path.split('/')[3]
    plt.title(real_y)
    plt.axis('off')
```

전이 학습

- 특징 추출기
  - Inception V3 특징 추출기 시퀀셜 모델의 테스트 데이터 분류 라벨 확인

```
# 예측값 표시
    plt.subplot(3,2,c*2+2)
    img = cv2.imread(image_path)
    img = cv2.resize(img, dsize=(299, 299))
    img = img / 255.0
    img = np.expand_dims(img, axis=0)

    # Inception V3를 이용한 특징 벡터 추출
    feature_vector = feature_model.predict(img)

    # Sequential 모델을 이용한 예측
    prediction = model.predict(feature_vector)[0]
```

전이 학습

- 특징 추출기
  - Inception V3 특징 추출기 시퀀셜 모델의 테스트 데이터 분류 라벨 확인

```
# 가장 높은 확률의 예측값 5개를 뽑음
    top_5_predict = prediction.argsort()[::-1][:5]
    labels = [unique_sorted_Y[index] for index in top_5_predict]
    color = ['gray'] * 5
    plt.barh(range(5), prediction[top_5_predict][::-1] * 100, color=color)
    plt.yticks(range(5), labels[::-1])
```

전이 학습

- 특징 추출기
  - 모델은 꽤 확신을 가지고 테스트 데이터에 대해서 답을 예측하고 있습니다.
  - 첫 번째 사진은 치와와, 두 번째 사진은 블랙앤탄 쿤하운드, 세 번째 사진은 휘펫 견종으로 예측했는데 구글 이미지에서 검색한 결과, 비교적 정확한 것 같습니다.
  - 그럼 이제 실제로 캐글에 결과를 업로드하기 위해 submission이라는 이름으로 불러온 데이터 프레임에 test_Y로 저장된 예측값을 넣겠습니다.

```
# 8.39 submission 데이터프레임에 예측값 저장
for i in range(len(test_Y)):
    for j in range(len(test_Y[i])):
        breed_column = unique_sorted_Y[j]
        submission.loc[i, breed_column] = test_Y[i, j]
```

전이 학습

- 특징 추출기
  - 예측값이 잘 저장됐는지 확인하기 위해 데이터의 일부를 출력해 보겠습니다.

```
# 8.40 submission 데이터 확인
print(submission.iloc[:5, :5])
```

  - iloc는 데이터 프레임에 인덱스로 접근합니다.
  - Iloc[행/ 열]의 형식으로 원하는 데이터를 뽑아 낼 수 있습니다.
  - 여기서는 처음 다섯 행과 다섯 열에 해당하는 데이터를 확인했습니다.
  - 각 값들은 더 이상 0.00083이 아니고 뭔가 의미있는 값이 된 것 같습니다.
  - 이제 submission 데이터 프레임을 CSV 파일로 저장합니다

전이 학습

- 특징 추출기
  - submission 데이터프레임을 csv 파일로 저장

```
# 8.41 submission 데이터프레임을 csv 파일로 저장
submission.to_csv('dogbreed_submission_inceptionV3_epoch10_299.csv',
index=False)
```

  - to_csv() 함수로 파일을 저장할 수 있습니다.
  - 파일의 이름은 나중에 알아보기 쉽게 비교적 구체적으로 작성하는편이 좋습니다.
  - 그런데 이 파일은 코랩 가상 환경에 저장되기 때문에 코랩과의 연결이 사라지면 파일도 함께사라집니다.
  - 파일을 보관하기 위해 로컬 환경에 저장하려면 파일 위에서 마우스 오른쪽 버튼을 클릭 했을 때 표시되는 컨텍스트 메뉴에서 '다운로드'를 누르면 파일을 로컬 환경에 내려받을 수 있습니다.

전이 학습

- 특징 추출기
  - 그럼 이제 이 파일을 캐글 환경에 실제로 올려보겠습니다.
  - 먼저 Dog breed competition의 submit 페이지로 이동합니다.
  - 그 다음 Step1 - Up load submission file이라고 돼 있는 부분에 submission csv 파일을 끌어 다 놓거나 그 부분을 클릭해서 파일을 찾아서 넣으면 됩니다.
  - 파일 업로드가 완료된 후에 마지막으로 최하단의 Make Submission 버튼을 누르면 몇 초 동안 서버에서 점수가 계산되고, 계산된 점수를 즉시 확인할 수 있습니다.
  - 테스트 데이터에 대한 점수는 0.33135 점이 나왔습니다.
  - 검증 데이터의 val_loss 보다는 쪼금 높은 값이 나왔습니다.
  - 리더 보드에서의 순위는 516 위 정도로 상위 40%입니다.
  - 특징 추출기로 ResNet 등 다른 네트워크를 사용하거나 분류모델에 여러 개의 레이어를 추가하는 등 다양한 시도를 통해 테스트 데이터에 대한 점수를 더 높일 수 있을 것입니다.

Q&A