전이학습

Transfer Learning

6. 전이학습

학습 목표

- 전이학습이 무잇인지 이해하기
- 사전 학습된 신경망 활용하기
- 신경망 미세 조정 이해하기
- 모델 학습에 오픈 소스 이미지 데이터셋
- 완결된 두 가지 전이학습 프로젝트 완성하기

- **학습 내용**

6. 전이학습

전이학습으로 해결할 수 있는 문제

- 전이학습이란 신경망이 어떤 과제에서 얻은 기존 지식(어떤 데이터셋을 학습하며 익힌 것)을 비슷한 다른 문제를 해결하는데 활용하는 것
- 어떤 과제에서 익힌 기존 지식이란 학습과정을 통해 얻어진 추출된 특징(특징 맵Feature Map)을 말함
- 딥러닝 분야에서 전이학습이 활발히 이용되는 이유는 더 짧은 시간에 적은 양의 데이터로 신경망을 학습할 수 있음
- 전이학습의 가장 큰 장점은 이미지넷 등 일반적인 컴퓨터비전벤치마크용 데이터셋으로
  학습한 모델의 가중치를 간단히 재사용하는 것만으로 다른 문제를 위한 대규모 데이터 수집(레이블링된 데이터)에 드는 수고를 절감할 수 있다는 것
- 전이학습의 또 다른 장점은 일반화 성능을 확보하고 과적합을 방지하는 것

6. 전이학습

전이학습이란

- 신경망이 어떤 과업task을 위해 많은 양의 데이터를 이용해 학습한 지식 (특징 맵)을 학습 데이터가 상대적으로 적은 다른 유사한 과업으로 옮겨오는 것을 말함

사전 학습된 신경망을 특징 추출기로 활용하는 방법

- 예)고양이와 개의 이미지를 분류하는 분류기를 학습하려 할 때 전이학습을 활용해보자.
  - (1) 우리가 해결하려는 문제와 비슷한 특징을 갖는 데이터셋을 물색 (문제와 가장 비슷한 오픈 소스 데이터셋 찾기) ➔ 5장에서 사용한적이 있는 이미지넷을 사용하기로 결정
  - (2) 이미지넷 데이터셋으로 학습되었고 준수한 성능을 보이는 신경망을 선택 (5장에서 배운 VGGNet, GoogLeNet, ResNet을 선택해도 됨)
    ➔ 이미지넷 데이터셋으로 학습된 VGG16 신경망을 활용
  - (3)사전 학습된 가중치를 포함하는 VGG16 모델을 내려받은 다음 분류기 부분을 제거하고 해결하려는 문제의 분류기 부분을 새로 만들어 추가한 후, 새로 구성한 신경망을 학습함

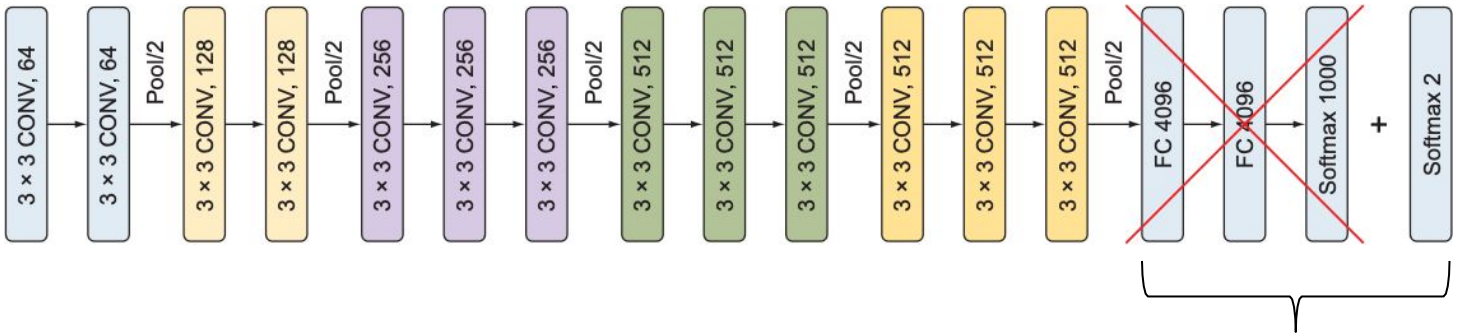
6. 전이학습

사전 학습된 신경망을 특징 추출기로 활용하는 방법

예)고양이와 개의 이미지를 분류하는 분류기를 학습하려 할 때 전이학습을 활용해보자.

- (3)사전 학습된 가중치를 포함하는 VGG16 모델을 내려받은 다음 분류기 부분을 제거하고 해결하려는 문제의 분류기 부분을 새로 만들어 추가한 후, 새로 구성한 신경망을 학습함

사전 학습된 VGG16 신경망을 이용하는 전이학습의 예



특징 추출기 부분은 그대로 두고, 분류기 부분을 제거한 다음,
새로 구현한 분류기 (유닛이 2개인 소프트맥스층)을 추가함

6. 전이학습

케라스로 사전 학습된 신경망을 이용해보자

- (1)가중치를 포함한 VGG16 신경망의 오픈 소스 구현을 내려받아 기반 모델을 만듬
  - 신경망 구현과 가중치를 모두 내려 받은 코드 예시

```
from keras.applications.vgg16 import VGG16

base_model = VGG16(weights = "imagenet", include_top=False,
                   input_shape = (224,224, 3))
base_model.summary()
```

Imports the VGG16 model from Keras

사전 학습된 가중치를 내려받아 변수 base_model에 할당. 이때 이미지넷 데이터셋으로 학습된 가중치 지정. Include_top을 False로 지정해서 분류기 부분의 가중치는 내려받지 않음

- (2) 모델의 개요를 출력해보면 5장에서 구현했던 VGG16의 신경망 구조와 일치하는 것을 알 수 있음
  - 딥러닝 라이브러리에서 제공하는 주요 신경망을 내려받아 이용하면 별도의 모델 학습 시간을 “많이” 절약할 수 있음
  - 주요 신경망 내려 받는 방식
    - (신경망 구현 + 가중치) 혹은(가중치만)

6.전이학습

케라스로 사전 학습된 신경망을 이용해보자

- (2) 모델의 개요를 출력해보면 5장에서 구현했던 VGG16의 신경망 구조와 일치

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

케라스 옵션으로 VGG16 신경망 구조에서 분류기 부분(3개의 전결합층)이 빠져 있음

6. 전이학습

케라스로 사전 학습된 신경망을 이용해보자

- (3) 특징 추출기 부분에 해당하는 층의 가중치는 고정
  - “가중치를 고정시킨다 " 라는 것은 미리 학습된 가중치가 추가 학습으로 변하지 않도록 한다는 것을 의미함 (즉, 기존 지식을 활용하는 것)

```
for layer in base_model.layers:
    layer.trainable = False

base_model.summary()
```

Iterates through layers and locks them to make them non-trainable with this code

```
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
```

6. 전이학습

케라스로 사전 학습된 신경망을 이용해보자

- (4) 새로운 분류기 추가
  - 분류 대상 클래스가 2개 이므로 유닛이 2개인 소프트맥스층 추가

```
from keras.layers import Dense, Flatten
from keras.models import Model

last_layer = base_model.get_layer('block5_pool')
last_output = last_layer.output

    x = Flatten()(last_output)

    x = Dense(2, activation='softmax', name='softmax')(x)
```

Imports Keras modules

get_layer 메서드를 사용해서 신경망의 마지막 층에 접근한다.

마지막 층의 출력을 다음 층의 입력으로 삼는다.

VGG16 모델의 출력을 1차원으로 변환해서 분류기 부분의 입력으로 삼는다.

유닛이 2개인 새 소프트맥스층을 모델에 추가한다

6. 전이학습

케라스로 사전 학습된 신경망을 이용해보자

- (5) base_model의 입력을 입력으로, 소프트맥스층의 출력을 출력으로 사용하는 새로운 모델을 구축

- 새 모델은 VGGNet의 특징 추출기 + 아직 학습되지 않은 새 소프트맥스 층으로 구성됨

```
new_model = Model(inputs=base_model.input, outputs=x)

new_model.summary()        <-- Prints the new_model summary
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_layer (Flatten)	(None, 25088)	0
softmax (Dense)	(None, 2)	50178

```
Total params: 14,789,955
Trainable params: 50,178
Non-trainable params: 14,714,688
```

6. 전이학습

전이학습의 원리

- 학습된 신경망이란?가중치 학습과 하이퍼파라미터 튜닝을 거쳐 신경망이 만족스러운 성능을 보이게 된 상태

- 모델 학습이 끝나면 “신경망 구조”와 “학습된 가중치” 두 가지가 결과물로 남음
- 사전 학습된 신경망을 활용하기 위해서는 신경망 구조와 가중치를 함께 내려받음

- 학습 중에 특징이 학습되려면 훈련 데이터에 포함된 특징이어야 함
- 대규모 모델(인셉션 등)은 대규모 데이터셋(이미지넷 같은)을 대상으로 학습되므로

  <거의 모든 특징이 이미 추출되어 사용할 수 있는 상태>가 됨
- 사전 학습 모델은 학습 데이터(새로운 과업용)에 포함되어 있지 않았던 특징까지도 포함되어 다른 신경망을 학습하는데 도움을 줌

6. 전이학습

전이학습의 원리

- 컴퓨터비전 문제 해결을 위해서는 모서리, 꼭짓점, 다양한 도형 같은 저수준 특징부터

  눈, 원, 사각형, 바퀴 같은 중수준 또는 고수준 특징까지 다양한 특징을 학습해야 함

- 합성곱신경망이 포착할 수 있는 이미지의 세부 사항은 매우 많지만 1000장 또는 25000장 정도의 데이터로는 과업 해결을 위한 특징을 모두 학습하기 어려움

- 따라서 사전학습된 신경망을 사용함으로써 기존에 배운 지식을 모두 내려받아 새로운 신경망에 포함시킬 수 있고, 빠른 학습과 더 높은 성능을 얻을 수 있음

6. 전이학습

신경망이 특징을 학습하는 방법

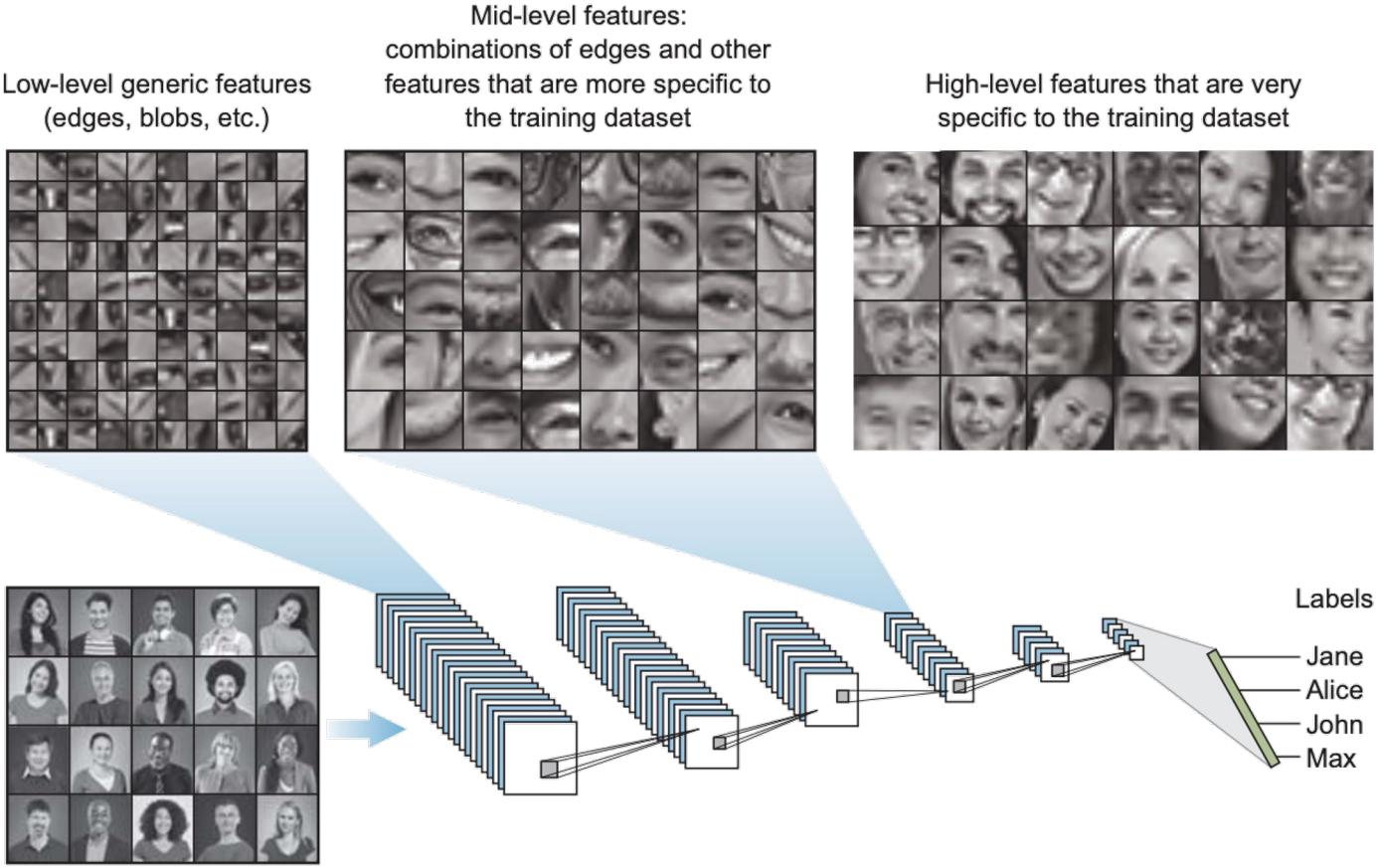
- 신경망은 단순한 것부터 각 층마다 차근차근 복잡도를 올려가며 특징을 학습함
- 이렇게 학습된 특징을 특징 맵이라고 함
- 신경망의 뒤쪽 층으로 갈수록 이미지의 특징적인 특징이 학습됨
- 첫번째 층은 모서리나 곡선 같은 저수준 특징을 포착
- 두번째 층은 원이나 사각형 같은 중수준 특징을 포착
  - 첫번째 층의 출력이 두번째 층의 입력으로 활용
  - 즉, 앞쪽 층의 특징을 모아 대상의 일부를 구성
- 층을 거쳐가며 보다 복잡한 특징을 나타내는 활성화 맵Activation Map이 생성됨
- 동시에 필터의 감수 영역도 점차 커짐
- 고차원 특징을 인식하는 층은 입력된 특징 중 해당 대상을 구별하는데 중요한 특징은

  증폭하고 그렇지 않은 특징은 억제함

6. 전이학습

신경망이 특징을 학습하는 방법

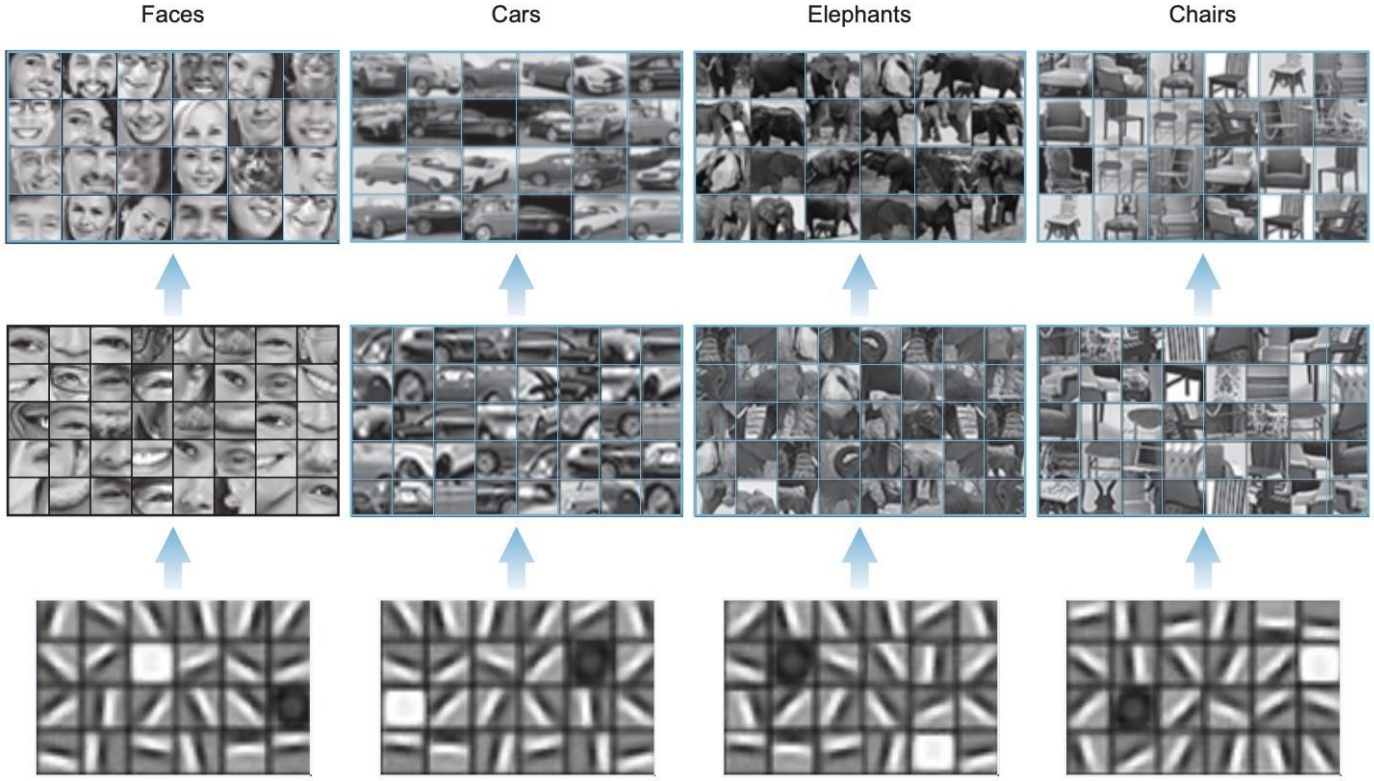
합성곱 신경망의 앞쪽 층에서는 일반적인 저수준 특징이 포착되며, 신경망의 뒷부분으로 갈수록 복잡하고 해당 이미지의 특징적인 특징이 학습됨



6. 전이학습

신경망이 특징을 학습하는 방법

- 저수준 특징: 거의 대부분 다른 어떤 과업에서도 활용할 수 있는 일반적인 정보만 담김
- 고수준 특징: 과업에 특화되며 다른 과업에서 재사용하기 어려워짐



6. 전이학습

뒤쪽 층에서 학습된 특징의 재사용성

- 신경망의 뒤쪽 층에서 학습된 특징의 재사용 가능 여부는 기존 모델 학습에 사용한 데이터셋과 새로운 데이터셋의 유사성과 관계가 깊음
- 모든 이미지에서 모서리나 직선을 찾아볼 수 있으므로 저수준 특징(앞쪽 층)은 서로 다른 과업에서도 재사용 가능함, 그러나 고수준 특징은 과업마다 크게 다름
- 따라서 원 도메인과 목표 도메인의 유사성에 따라 고수준 내지 중수준 특징의 재사용 여부를 판단할 수 있음

6. 전이학습

전이학습의 세 가지 방식

① 사전 학습된 신경망을 **분류기**로 이용하거나

② 사전 학습된 신경망을 **특징 추출기**로 이용하거나

③ **미세 조정**으로 이용하기 등 크게 세 가지로 나뉨

- 전이학습의 세 가지 방식 모두 층수가 많은 CNN 모델을 구축하고 학습하는 과정에서 상당한 시간을 절약 할 수 있으며 뛰어난 성능을 얻을 수 있음

사전 학습된 신경망을 분류기로 이용하기 (유) 견종 분류기

- 사전 학습된 신경망을 분류기로 이용하는 방식은 사전 학습된 신경망의 가중치를 고정하거나 추가적인 학습이 불필요함
- 비슷한 과업에 대해 학습된 신경망을 골라 직접 새로운 과업에 그대로 투입하는 방식
- 원 도메인과 목표 도메인이 매우 유사하고, 사전 학습된 신경망을 즉시 사용할 수 있는 경우에 적용함

6. 전이학습

사전 학습된 신경망을 분류기로 이용하기

- 이미지넷 데이터셋으로 학습된 VGG16 신경망을 그대로 사용할 수 있음
  - 이미지넷 데이터에는 개 이미지가 많이 포함되어 있으므로 견종을 구분하는데 필요한 특징의 상당수가 이미 추출되어 있으리라 생각할 수 있음

- (1) 필요한 라이브러리를 임포트

```
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16
```

- (2) 사전 학습된 VGG16 신경망과 가중치 내려받기

```
model = VGG16(weights = "imagenet", include_top=True, input_shape =
   (224,224, 3))
```

6. 전이학습

사전 학습된 신경망을 분류기로 이용하기

- (3) 입력 이미지를 읽어 들이고 전처리

파일에서 이미지 읽기

```
image = load_img('path/to/image.jpg', target_size=(224, 224))
image = img_to_array(image)
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
image = preprocess_input(image)
```

이미지의 픽셀값을 넘파이 배열로 변환

VGG 모델 입력을 위해 전처리

모델에 입력할 수 있도록 배열 모양을 변경

6. 전이학습

사전 학습된 신경망을 분류기로 이용하기

- (4) 입력 이미지 준비가 끝났으면 예측을 실행

```
yhat = model.predict(image)

label = decode_predictions(yhat)

label = label[0][0]

print('%s (%.2f%%)' % (label[1], label[2]*100))
```

모든 클래스에 대해 예측 확률 계산

확률로부터 예측 클래스 결정

가장 확률이 높은 클래스 값 추출

예측 결과 출력

```
>> German_shepherd (99.72%)
```

6. 전이학습

사전 학습된 신경망을 특징 추출기로 이용하기 유 개/고양이 분류기

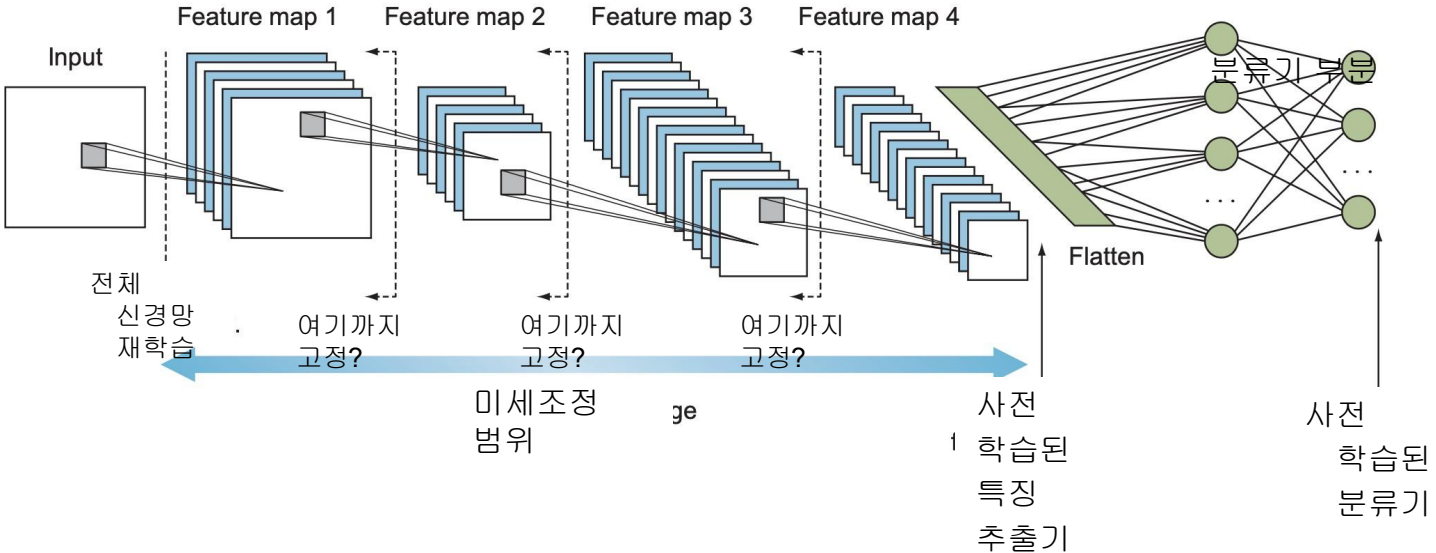
- 이미지넷 데이터셋을 학습한 CNN의 특징 추출기 부분 가중치를 고정하고 분류기 부분을 제거한 다음 새로운 분류기 부분을 추가하여 문제 해결

- 원 도메인의 데이터셋과 새로운 과업이 큰 차이가 없을 때 사용하는 방식
  - 이미지넷 데이터셋에는 고양이나 개 이미지를 상당수 포함하고 있기 때문에 개/고양이 분류기와 같은 새로운 과업에 사용할 수 있음
  - 즉, 이미지넷 데이터셋에서 추출한 고수준 특징을 다른 과업에 재사용할 수 있음
- 사전 학습된 신경망의 가중치를 모두 고정하고 새로 추가한 분류기 부분의 전결합층만 새로운 데이터셋으로 학습 진행
  - 사전 학습된 신경망의 분류기 부분은 원 분류 과업에 특화된 경우가 많고, 이후 모델이 학습된 클래스셋에만 한정되기 때문에 제거해야 함
  - 예를들어 이미지넷은 1000개 이상의 분류 클래스가 있으므로 기존 신경망의 분류기 부분 역시 1000가지로 분류하도록 설계 및 학습되었으므로 개와 고양이만

6. 전이학습

미세 조정하기

- 목표 도메인과 원 도메인이 많이 동떨어진 경우 (전혀 다른 경우 포함), 미세 조정fine- tuning 을 거쳐 문제를 해결할 수 있음
- 미세 조정의 정의는 특징 추출에 쓰이는 신경망의 일부 층을 고정하고 고정하지 않은 층과 새로 추가된 분류기 부분의 층을 함께 학습하는 방식임
- 이 방식을 미세 조정이라 부르는 이유는 특징 추출기 부분을 재 학습하면서 고차원 특징의 표현이 새로운 과업에 적합하게 조정되기 때문임



6. 전이학습

미세 조정하기

- 미세 조정의 특징 맵 보전 범위는 대개 시행착오를 통해 결정됨
- 하지만 직관적으로 따를 수 있는 가이드라인도 존재함
- 학습 데이터 양과 원 도메인과 목표 도메인의 유사성, 두가지 요소의 의해 결정됨
- 6.5절에서 네 가지 시나리오를 상정해 적합한 특징 맵의 보전 범위를 살펴보겠음

미세 조정이 모델을 처음부터 학습시키는 것보다 나은가

- 새로운 모델을 처음부터 학습시키면 가중치를 무작위 값으로 초기화하고 경사 하강법을 이용해서 오차 함수값이 최소가 되게 하는 최적의 가중치를 찾아가게 됨
- 무작위 값으로 초기화된 모델은 우리가 원하는 최적값과 가까이 있을 거라는 보장이 없음. 그리고 이 초기값이 최적값과 거리가 멀다면 가중치가 수렴하는데 오랜 시간이 걸림

6. 전이학습

미세 조정이 모델을 처음부터 학습시키는 것보다 나은가

- 미세 조정의 이점은 사전 학습된 신경망의 가중치는 이미 학습 데이터를 학습하며 최적화된 상태이므로 새로운 문제에서 재사용하면 한번 최적화된 가중치를 대상으로 학습이 진행됨

- 따라서 무작위 값으로 초기화된 가중치와 비교하면 가중치 수렴이 빨리 일어남
- 사전 학습된 신경망 전체를 재 학습 하더라도 처음부터 모델을 학습시키는 것 (스크래치 단계에서부터 학습) 보다 학습 속도가 빠름

미세 조정에서는 학습률을 작게 설정한다

- 미세 조정에서는 합성곱층의 학습률을 무작위 값으로 초기화된 가중치를 가진 분류기 부분보다 작게 설정함

- 이미 최적 값에 가까워서 바르게 수정할 필요가 상대적으로 적기 때문임

6. 전이학습

적합한 전이학습 수준 선택하기

- 전이학습의 적합한 수준을 결정하는 중요한 요소
  - 목표 데이터셋의 크기 (많음 또는 적음): 목표 데이터셋의 크기가 작다면 많은 층을 학습시키기 어렵고 새로운 데이터에 대해 과적합을 일으키기 쉬움. 이런 경우에는 미세 조정 범위를 줄이고 원 데이터셋의 의존도를 높여야 함
  - 원 도메인과 목표 도메인의 유사성: 해결하려는 문제가 자동차와 배를 분류하는 것이라면 비슷한 특징을 다수 포함하는 이미지넷 데이터셋으로도 충분함. 반면 새로운 문제가 엑스레이 사진에서 폐암 병변을 찾아내는 것이라면 도메인이 전혀 달라지므로 미세 조정 범위가 넓어져야 함

- 전이학습의 적합한 수준을 결정하는 네 가지 시나리오
  - 목표 데이터셋의 크기가 작고, 원 도메인과 목표 도메인이 유사
  - 목표 데이터셋의 크기가 크고, 원 도메인과 목표 도메인이 유사
  - 목표 데이터셋의 크기가 작고, 원 도메인과 목표 도메인이 크게 다름
  - 목표 데이터셋의 크기가 크고, 원 도메인과 목표 도메인이 크게 다름

6. 전이학습

시나리오1: 목표 데이터셋의 크기가 작고, 두 도메인이 유사한 경우

- 원-목표 데이터셋이 서로 유사하므로 사전 학습된 고수준 특징도 재사용 가능
- 따라서 특징 추출기 부분의 가중치를 고정하고 분류기 부분만 새로 학습
- 미세 조정이 유리하지 않은 또 다른 이유는 목표 데이터셋의 크기가 작기 때문
- 특징 추출기에 해당하는 층을 미세 조정 범위에 포함시키면 과적합이 발생하기 쉬움
- 데이터 셋의 크기가 작으면 대상의 모든 가능한 특징을 담지 못할 가능성이 높기 때문에 일반화 성능이 떨어짐
- 따라서 미세 조정 범위를 넓힐수록 신경망이 과적합을 일으킬 가능성이 높음
- 예를들어 목표 데이터셋의 개 이미지가 특정 날씨 조건(눈)에 국한되었다면 개의 특징으로 눈이 내린 희 배경을 포착할 것이므로 다른 날씨에 찍힌 개 사진을 제대로 분류하지 못할 것임
- 여기서 도출할 수 있는 원칙은 “목표 데이터셋의 크기가 작다면 사전 학습된 신경망의 미세 조정 범위를 넓히는데 주의해야 한다” 는 것임

6. 전이학습

시나리오2: 목표 데이터셋의 크기가 크고, 두 도메인이 유사한 경우

- 시나리오 1의 해법과 비슷함
- 원-목표 도메인이 유사하므로 특징 추출기 부분의 가중치를 고정하고 분류기

  부분을 재 학습 하면 됨

- 하지만 목표 도메인에 충분한 데이터가 있으므로 미세 조정 범위를 적절히 넓혀도

  과적합에 대한 우려 없이 학습을 진행할 수 있음

- 두 도메인이 비슷하므로 고수준 특징도 서로 비슷하므로 신경망 전체를 미세 조정

  범위로 삼을 필요는 없음
- 사전 학습된 신경망의 60~80% 정도를 고정하고 나머지 부분을 목표

  데이터셋으로 재 학습 하는 것이 적절하다고 보여짐

6. 전이학습

시나리오3: 목표 데이터셋의 크기가 작고, 두 도메인이 크게 다른 경우

- 원-목표 도메인이 크게 다르므로 사전 학습된 신경망을 도메인에 특화된 고수준 특징까지 고정하는 것은 적절하지 않음
- 앞쪽 층의 <일부 저수준 특징을 고정>하거나, 특징 재활용 없이 <전체 신경망을 미세 조정 범위>로 삼는 것이 좋음
- 목표 데이터셋의 크기가 작은 만큼 전체 신경망의 재학습이 어려울 수 있음
- 따라서 이런 경우 중용을 취하도록 함
- 사전 학습된 신경망의 뒷부분 1/3이나 절반 정도의 특징을 고정하는 수준으로 시작하는 것이 적절함
- 도메인이 크게 다르더라도 일반적인 특징 맵(저수준 특징)은 재사용할 수 있음

6. 전이학습

시나리오

- **4: 목표 데이터셋의 크기가 크고, 두 도메인이 크게 다른 경우**
  - 목표 데이터셋의 크기가 크므로 전이학습 없이 전체 신경망을 처음부터 학습할 수 있다는 유혹에 빠지기 쉬움
  - 그러나 실제로는 목표 데이터셋의 크기가 크더라도 사전 학습된 가중치를 재학습하는 것이 더 이로운 경우가 많음
  - 학습 시간이 빠른 것은 물론이고, 목표 데이터셋의 크기가 크므로 과적합에 대한 우려 없이 재학습을 진행할 수 있음

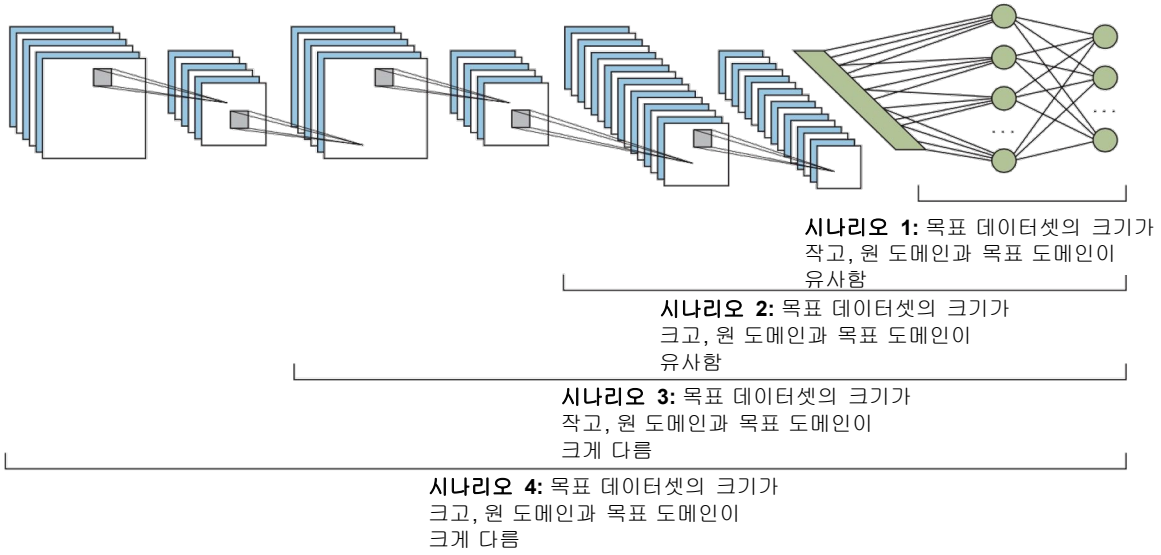
6. 전이학습

네 가지 시나리오 정리

- 네 가지 시나리오를 정리해 보면 다음과 같음

시나리오	목표 데이터셋 크기	원-목표 도메인의 유사성	적합한 접근법
1	작다	유사	사전 학습된 신경망을 특징 추출기로 사용
2	크다	유사	전체 신경망을 미세 조정
3	작다	크게 다르다	신경망의 뒷부분을 미세 조정
4	크다	크게 다르다	전체 신경망을 미세 조정

- 네 가지 시나리오에 적합한 미세 조정 수준을 제시하는 가이드라인



6. 전이학습

■ 6.6 오픈 소스 데이터셋

- 본 절에서 소개하는 데이터셋은 현재 컴퓨터 비전 연구자 커뮤니티에서 가장 널리

  사용되는 것을 정리한 것으로 공개된 모든 데이터셋을 소개하는 것은 아님
- 훌륭한 이미지 데이터셋이 많이 나와 있고, 최근 <paper with code> 라는

  사이트를 통해 연구자들이 공개하는 오픈 소스 데이터셋이 정리되어 소개되고

  있으니, 프로젝트 시작하기 전에 어떤 데이터셋이 사용가능한지 직접

  조사해보는 것을 권장함

6. 전이학습

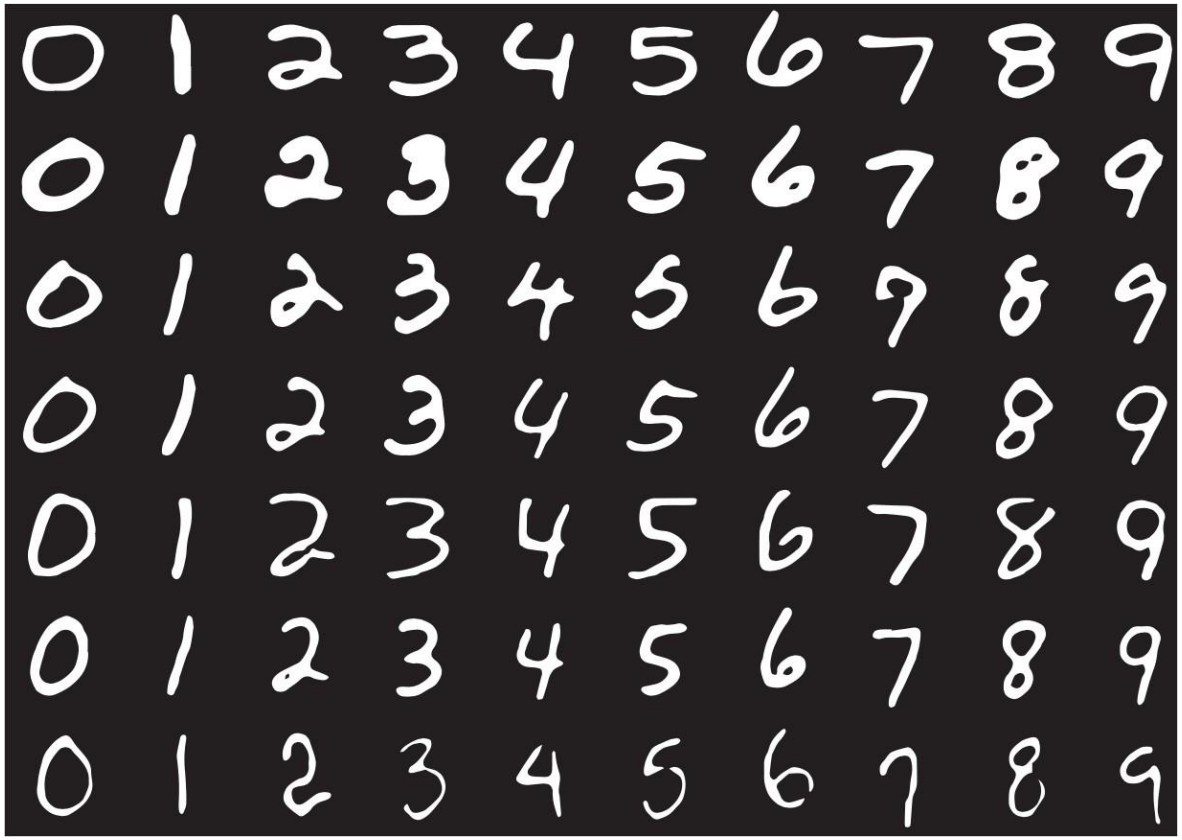
■ 6.6.1 MNIST

- MNIST 는 손글씨 이미지 데이터셋
- 0부터 9까지의 숫자가 쓰인 손글씨 이미지로 구성되어 있음
- 이 데이터셋의 목적은 손글씨 숫자 이미지를 분류하는 것
- 연구자들이 분류 알고리즘의 성능을 가늠하는 벤치마크 테스크로 오랫동안 사용해옴
- 이미지 데이터셋의 “hello world”와도 같은 존재
- 그러나 최근 기본적인 합성곱 신경망으로도 99% 성능을 달성하여 최고 성능을 다투는

  테스트로서의 역할을 상실함
- 6만 장의 학습 데이터와 1만 장의 테스트 데이터로 구성
- 이미지는 모두 회색조(단일채널)고, 이미지 크기는 가로세로 모두 28픽셀

6. 전이학습

- **6.6.1 MNIST**



6. 전이학습

■ 6.6.2 Fashion-MNIST

- Fashion-MNIST 데이터셋은 현재의 기술 수준에서 지나치게 단순해진 MNIST 데이터셋을 대체하려는 목적으로 만들어짐
- 데이터의 형식은 MNIST와 같지만 손글씨 이미지가 아니라 티셔츠, 바지, 풀오버, 드레스, 코트, 샌들, 셔츠, 운동화, 가방, 단화 등 10가지 패션의류 클래스로 나뉨

6. 전이학습

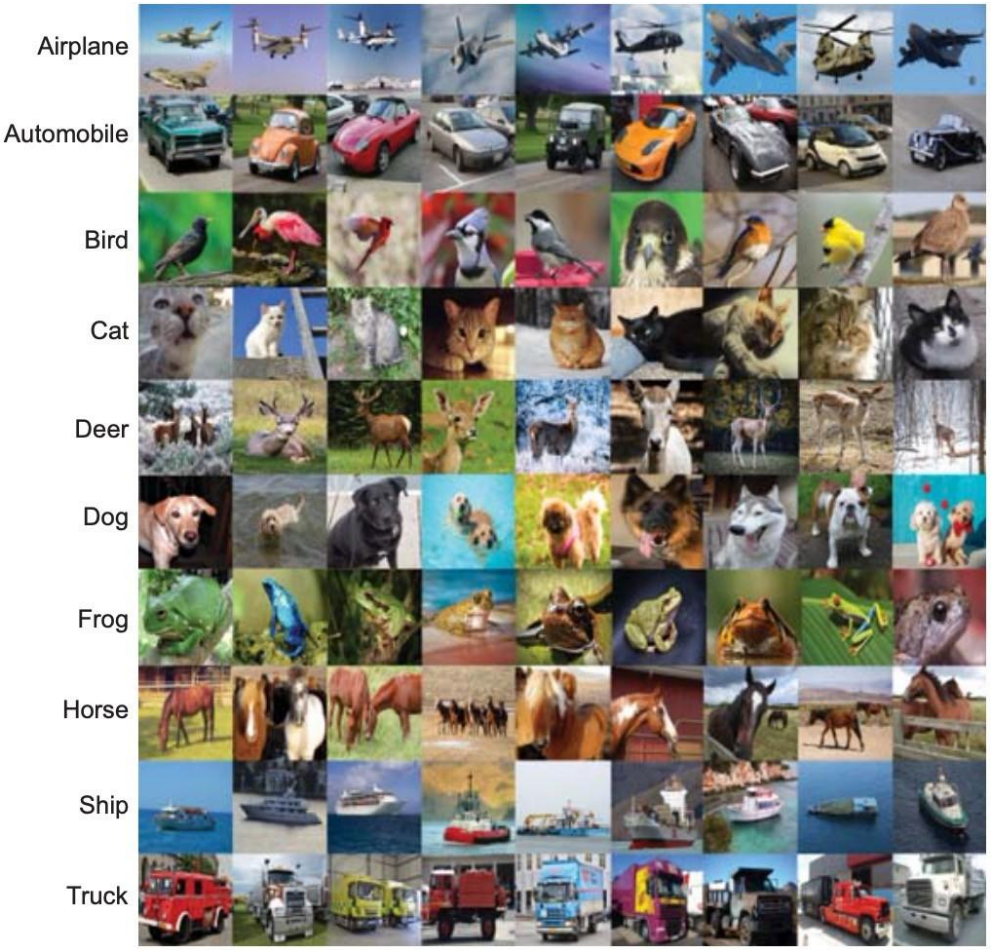
■ 6.6.3 CIFAR

- CIFAR-10 컴퓨터 비전이나 머신러닝 문헌에서 널리 사용되는 벤치마크 데이터셋임
- CIFAR 데이터셋의 이미지는 MNIST 데이터셋의 이미지 보다 더 복잡함
- 우선 MNIST 데이터셋의 이미지가 모두 회색조 이미지고 대상 이미지 중심에 위치했던 것이 비해, CIFAR 데이터셋의 이미지는 컬러 이미지고 이미지에 대상이 나타나는 방식도 제각각임
- 10개 클래스로 나뉜 32x32픽셀 크기의 이미지로 구성되며, 각 클래스마다 6000개의 이미지를 포함하고, 학습 데이터 5만장, 테스트 데이터 1만장으로 구성됨

- CIFAR-100은 CIFAR-10 데이터셋을 확장한 데이터셋임
- 클래스가 100개로 늘어났으며 각 클래스마다 600장의 이미지가 있음
- 100개의 클래스는 다시 20개의 슈퍼클래스에 속하는데, 이미지에는 대분류 레이블과 소분류 레이블이 부여되어 있음

6. 전이학습

- **6.6.3 CIFAR**



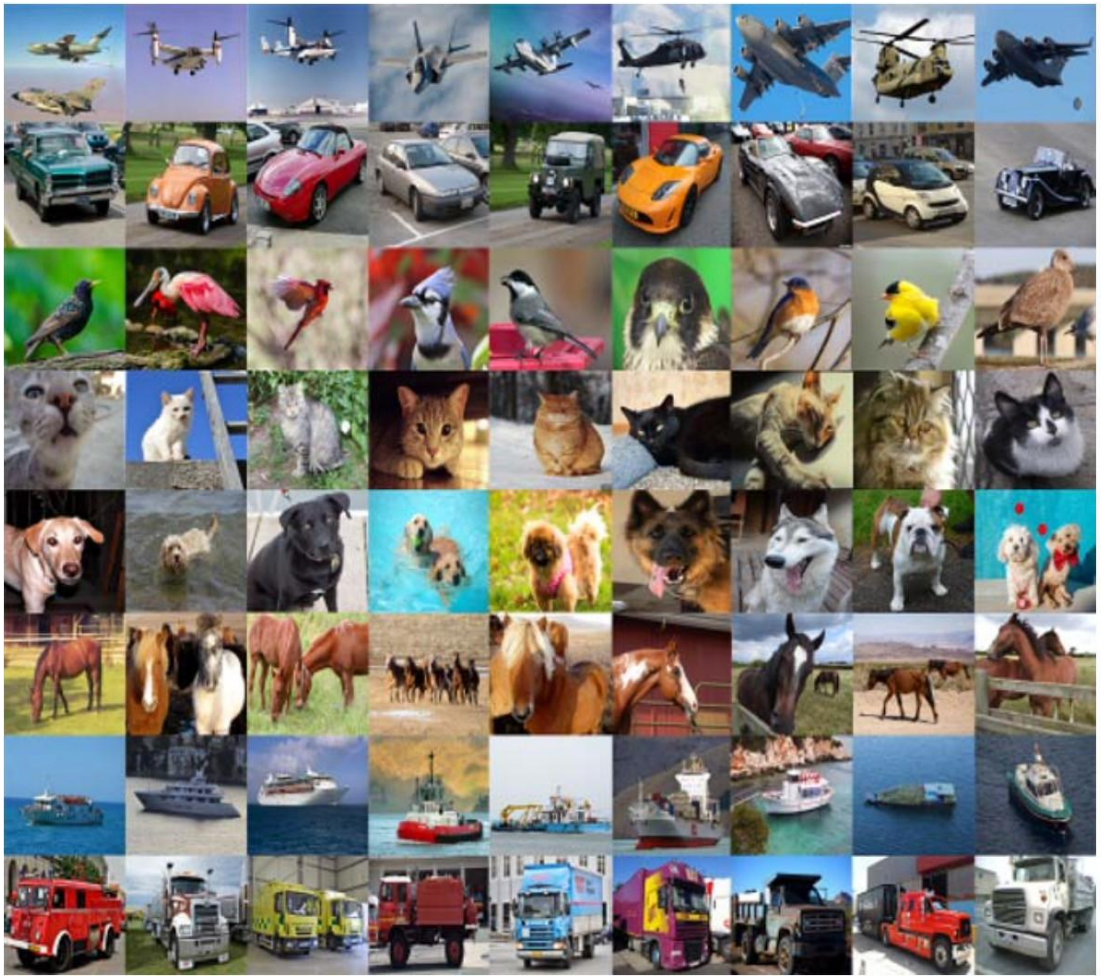
6. 전이학습

■ 6.6.4 이미지넷

- 이미지넷은 컴퓨터 비전 연구자들 사이에서 자신의 분류 알고리즘 성능을 측정하기 위해 널리 사용되는 최신 벤치마크 데이터셋임
- 이미지넷은 본래 물체를 시각적으로 인식하는 소프트웨어를 연구하기 위한 목적으로 만들어진 대규모 이미지 데이터베이스임
- 데이터셋을 구성하는 이미지는 웹에서 수집되어 아마존의 클라우드 소싱 도구인 터크를 이용해 수동으로 레이블이 부여되었음
- 1000개의 클래스로 구성되며, 하나의 클래스에는 1000개 이상의 이미지가 속하도록
- 구성되어 있음
- 이미지넷을 활용한 대규모 물체 인식 경진대회 ImageNet Large Scale Visual Recognition Challenge, ILSVRC 가 존재하며, 이 경진대회에서는 각 참가팀이 만든 소프트웨어로 이미지에서 물체를 인식하고 분류하는 정확도를 겨룸
- 현재 ILSVRC 대회는 종료되었음

6. 전이학습

- **6.6.4**
  이미지넷



6. 전이학습

6.6.5 MS COCO

- MS COCO는 물체 인식, 인스턴스 세그먼테이션, 이미지 캡셔닝, 인체 주요 부위 위치 파악 등의 연구를 위해 만든 오픈 소스 데이터베이스임 (2014년 제안)

- 약 32만 8000장의 이미지를 포함하며, 그중 20만장 이상이 레이블이 부여되어 있음
- 이들 이미지에는 4세 어린이가 쉽게 인식할 수 있는 수준의 80개 물체 카테고

6. 전이학습

6.6.6 구글 오픈 이미지

- 구글 오픈 이미지는 구글이 만든 오픈 소스 이미지 데이터베이스임
- 집필 시점 현재 9백만장의 이미지를 포함함
- 구글 오픈 이미지의 특징은 이미지들이 대부분 수천 가지 물체 클래스에 걸쳐 있는

  복잡한 장면이라는 점
- 특히 이들 중 2백만장 이상의 이미지는 사람이 직접 물체의 위치를 나타내는

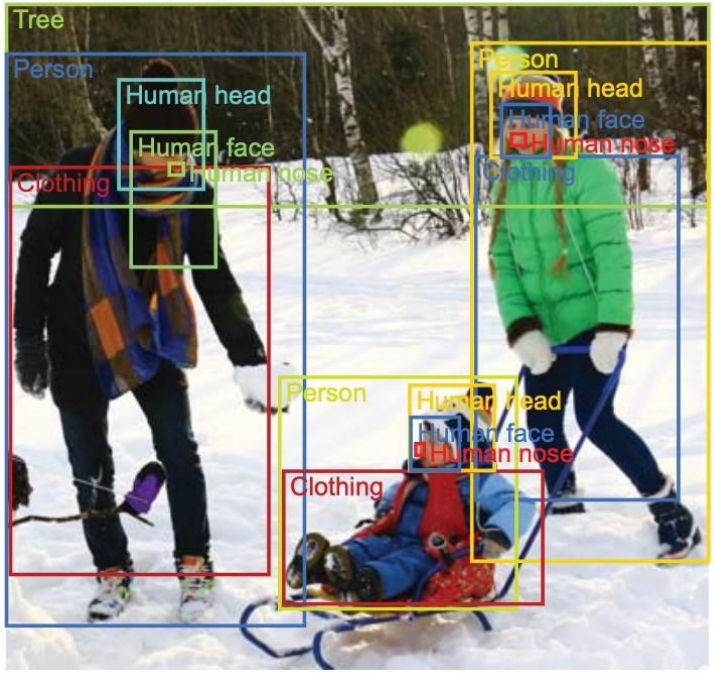
  범위를 태깅한 것으로 그 덕분에 구글 오픈 이미지는 물체 위치가 표시된

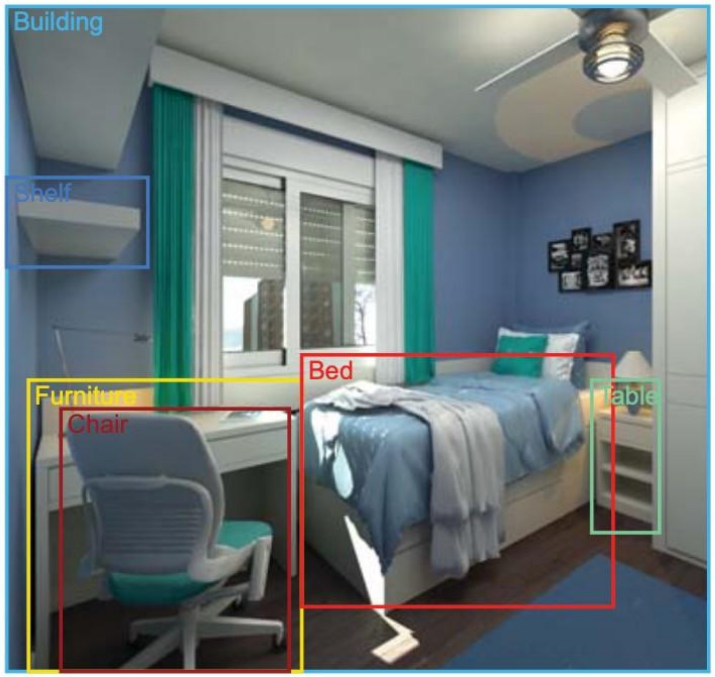
  이미지 데이터베이스 중 가장 규모가 큼
- 이들 이미지에는 약 1500만개의 물체 위치 범위가 태깅되어 있으며,

  태깅된 물체 종류도 600클래스에 이름
- 구글 오픈 이미지에도 오픈 이미지 챌린지라는 경진대회가 있음

6. 전이학습

6.6.6 구글 오픈 이미지





구글 오픈 이미지 데이터셋에 포함된 물체의 위치가 태깅된 이미지

6. 전이학습

6.6.7 캐글 (Kaggle)

- 지금까지 소개한 데이터셋 외에 캐글에서도 좋은 데이터셋을 구할 수 있음
- 캐글은 전 세계사람이 모여 머신러닝이나 딥러닝을 이용한 경진대회를

  주최하거나 참가하는 웹사이트임

- 지금까지 소개한 데이터셋 외에도 매일같이 새로운 오픈소스 데이터셋이 공개되고
  있어서 이들 데이터를 경험하며 각 데이터셋이 가진 클래스나

  유스케이스를 이해하는 능력을 키울 수 있음

6. 전이학습

6.7 프로젝트 1: 사전 학습된 신경망을 특징 추출기로 사용하기

- 개와 고양기 분류기 만들기
  - 본 절에서는 목표 데이터셋의 규모가 작고, 원 도메인과 목표 도메인의 유사성이

    높은 <시나리오1> 타입의 전이학습 구현방법을 익힘
  - 이번 프로젝트에서 중요한 점은 사용자 정의 데이터의 전처리 및

    신경망 학습에 사용하기 위한 준비 과정을 익히는 것

  - VGG16 을 사용하고 약 96%의 정확도를 얻을 수 있음

6. 전이학습

■ 6.7 프로젝트 1: 사전 학습된 신경망을 특징 추출기로 사용하기

- 사전 학습된 신경망을 특징 추출기로 사용하는 절차
  - (1)필요한 라이브러리를 임포트
  - (2)데이터를 학습에 사용할 수 있도록 전처리
  - (3)대규모 데이터셋에서 사전 학습된 VGG16 모델의 가중치 로드
  - (4)합성곱층의 특징 추출기 부분 가중치를 모두 고정
  - (5)사전 학습된 신경망에서 분류기 부분을 제거 및 새로운 분류기 추가
    - 추가할 전결합층 수 및 유닛 수에 특별한 제한은 없음. 이번 프로젝트는 복잡하지 않은 문제이므로 64개 유닛을 가진 전결합층 1개를 추가. 나중에 성능 추이를 보아 과소적합이 일어난다면 유닛 수를 증가시키고, 과적합이 일어난다면 유닛 수를 줄이면 됨. 소프트맥스층의 유닛 수는 분류 대상 클래스 수와 같아야 함
  - (6)신경망을 컴파일하고 새로운 소규모 데이터셋으로 모델 재학습
  - (7)모델의 성능을 평가

6. 전이학습

- **6.7 프로젝트 1: 사전 학습된 신경망을 특징 추출기로 사용하기**
  - (1) 필요한 라이브러리를 임포트
    Import the necessary libraries:

```
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.applications import imagenet_utils
from keras.applications import vgg16
from keras.applications import mobilenet
from keras.optimizers import Adam, SGD
from keras.metrics import categorical_crossentropy
from keras.layers import Dense, Flatten, Dropout, BatchNormalization
from keras.models import Model
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt
%matplotlib inline
```

6. 전이학습

■ 6.7 프로젝트 1: 사전 학습된 신경망을 특징 추출기로 사용하기

- (2) 데이터를 학습에 사용할 수 있도록 전처리
  - 케라스에는 데이터 강화를 수행하는 ImageDataGenerator 라는 클래스가 있음
  - 이번 예제에서는 편의상 데이터 강화는 생략함

ImageDataGenerator 클래스는 실시간으로 데이터 강화를 수행하며 여러 개의 배치로 나뉜 텐서를 생성해준다. 학습은 이렇게 분할된 배치를 차례대로 반복 입력하는 방식으로 진행된다. 다만 이번 예제에서는 데이터 강화를 적용하지 않았다.

```
train_path  = 'data/train'
valid_path  = 'data/valid'
test_path  = 'data/test'

train_batches = ImageDataGenerator().flow_from_directory(train_path,
                                                         target_size=(224,224),
                                                         batch_size=10)

valid_batches = ImageDataGenerator().flow_from_directory(valid_path,
                                                         target_size=(224,224),
                                                         batch_size=30)

test_batches = ImageDataGenerator().flow_from_directory(test_path,
                                                        target_size=(224,224),
                                                        batch_size=50,
                                                        shuffle=False)
```

6. 전이학습

■ 6.7 프로젝트1: 사전 학습된 신경망을 특징 추출기로 사용하기

- (3) 대규모 데이터셋에서 사전 학습된 VGG16 모델의 가중치 로드
  - 분류기 부분을 제거할 것이므로 include_top=False 로 지정

```
base_model = vgg16.VGG16(weights = "imagenet", include_top=False,
                         input_shape = (224,224, 3))
```

- (4) 합성곱층의 특징 추출기 부분 가중치를 모두 고정

```
for layer in base_model.layers:
    layer.trainable = False
```

Iterates through layers and locks them to make them non-trainable with this code

6. 전이학습

■ 6.7 프로젝트 1: 사전 학습된 신경망을 특징 추출기로 사용하기

- (5) 사전 학습된 신경망에서 분류기 부분을 제거 및 새로운 분류기 추가

Get_layer() 메서드를 사용해서 신경망의 마지막 층과 그 출력을 별도의 변수에 따로 저장한다.

```
last_layer = base_model.get_layer('block5_pool')
last_output = last_layer.output

x = Flatten()(last_output)

x = Dense(64, activation='relu', name='FC_2')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(2, activation='softmax', name='softmax')(x)

new_model = Model(inputs=base_model.input, outputs=x)
new_model.summary()
```

VGG16 신경망의 출력을 분류기 부분에 입력할 수 있도록 1차원으로 변환한다.

유닛이 64개인 전결합층.
배치 정규화 층.
드롭아웃층.
소프트맥스층을 각각 하나씩 추가한다.

6. 전이학습

■ 6.7 프로젝트 1: 사전 학습된 신경망을 특징 추출기로 사용하기

- (6) 신경망을 컴파일하고 새로운 소규모 데이터셋으로 모델 재학습
  - CPU만 사용해도 모델의 학습이 굉장히 빨리 진행되는 것을 볼 수 있음
    - 한 에포크당 약 25초에서 29초 정도가 걸리니 20 에포크의 학습도 채 10분이 걸리지 않음

```
new_model.compile(Adam(lr=0.0001), loss='categorical_crossentropy',
                  metrics=['accuracy'])

new_model.fit_generator(train_batches, steps_per_epoch=4,
                        validation_data=valid_batches, validation_steps=2,
                        epochs=20, verbose=2)
```

6. 전이학습

■ 6.7 프로젝트 1: 사전 학습된 신경망을 특징 추출기로 사용하기

- (7) 모델의 성능을 평가
  - 먼저 데이터셋을 텐서로 변환하는 load_dataset() 메서드를 정의

```
from sklearn.datasets import load_files
from keras.utils import np_utils
import numpy as np

def load_dataset(path):
    data = load_files(path)
    paths = np.array(data['filenames'])
    targets = np_utils.to_categorical(np.array(data['target']))
    return paths, targets

test_files, test_targets = load_dataset('small_data/test')
```

6. 전이학습

■ 6.7 프로젝트 1: 사전 학습된 신경망을 특징 추출기로 사용하기

- (7) 모델의 성능을 평가
  - 를 만듦

```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from tqdm import tqdm

def path_to_tensor(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    x = image.img_to_array(img)
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in
tqdm(img_paths)]
    return np.vstack(list_of_tensors)

test_tensors = preprocess_input(paths_to_tensor(test_files))
```

PIL.Image.Image 타입의 이미지를 모양이 (244,244,3) 인 3차원 텐서로 변환한다.

3차원 텐서를 모양이 (1,244,244,3) 인 4차원 텐서로 변환한다.

RGB 이미지를 PIL.Image.Image 타입으로 읽어 들인다.

6. 전이학습

■ 6.7 프로젝트 1: 사전 학습된 신경망을 특징 추출기로 사용하기

- (7) 모델의 성능을 평가
  - 이제 케라스의 evaluate() 메서드를 실행해서 모델의 정확도를 측정할 수 있음

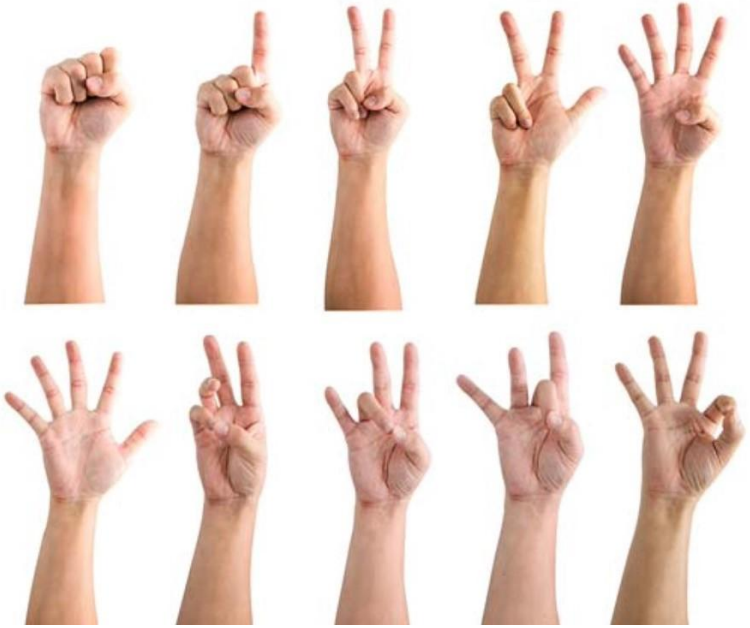
```
print('\nTesting loss: {:.4f}\nTesting accuracy:
  {:.4f}'.format(*new_model.evaluate(test_tensors, test_targets)))
```

```
Testing loss: 0.1042
Testing accuracy: 0.9579
```

6. 전이학습

■ 6.8 프로젝트2: 미세 조정

- 목표 데이터셋의 규모도 작고, 원 도메인과 목표 도메인이 매우 다른 경우 (시나리오3)
- 10가지 수화를 구별할 수 있는 분류기 구축



분류대상 클래스의 가짓수 = 10 이미지 크기 = 100x100
색 공간 = RGB
훈련 데이터 1712건 검증 데이터 300건
테스트 데이터 50건

6. 전이학습

■ 6.8 프로젝트 2: 미세조정

- 사전 학습된 신경망을 특징 추출기로 사용하는 절차
  - (1)필요한 라이브러리를 임포트
  - (2)데이터를 학습에 사용할 수 있도록 전처리
  - (3)대규모 데이터셋에서 사전 학습된 VGG16 모델의 가중치 로드
  - (4)합성곱층의 특징 추출기 일부 가중치를 고정
  - (5) 새로운 분류기 추가
  - (6)신경망을 컴파일하고 새로운 소규모 데이터셋으로 모델 재학습
  - (7)모델의 성능을 평가

6. 전이학습

■ 6.8 프로젝트2: 미세조정

- (1)필요한 라이브러리를 임포트

```
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.applications import imagenet_utils
from keras.applications import vgg16
from keras.optimizers import Adam, SGD
from keras.metrics import categorical_crossentropy
from keras.layers import Dense, Flatten, Dropout, BatchNormalization
from keras.models import Model
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt
%matplotlib inline
```

6. 전이학습

■ 6.8 프로젝트 2: 미세조정

- (2) 데이터를 학습에 사용할 수 있도록 전처리
  - 케라스에는 데이터 강화를 수행하는 ImageDataGenerator 라는 클래스가 있음
  - 이번 예제에서는 편의상 데이터 강화는 생략함

ImageDataGenerator 클래스는 실시간으로 데이터 강화를 수행하며 여러 개의 배치로 나뉜 텐서를 생성해준다. 학습은 이렇게 분할된 배치를 차례대로 반복 입력하는 방식으로 진행된다. 다만 이번 예제에서는 데이터 강화를 적용하지 않았다.

```
train_path  = 'dataset/train'
valid_path  = 'dataset/valid'
test_path  = 'dataset/test'

train_batches = ImageDataGenerator().flow_from_directory(train_path,
                                                         target_size=(224,224),
                                                         batch_size=10)

valid_batches = ImageDataGenerator().flow_from_directory(valid_path,
                                                         target_size=(224,224),
                                                         batch_size=30)

test_batches = ImageDataGenerator().flow_from_directory(test_path,
                                                        target_size=(224,224),
                                                        batch_size=50,
                                                        shuffle=False)
```

6. 전이학습

■ 6.8 프로젝트2: 미세조정

- (3) 대규모 데이터셋에서 사전 학습된 VGG16 모델의 가중치 로드
  - 분류기 부분을 제거할 것이므로 include_top=False 로 지정

```
base_model = vgg16.VGG16(weights = "imagenet", include_top=False,
                         input_shape = (224,224, 3), pooling='avg')
```

- (4) 합성곱층의 특징 추출기 중 일부 가중치를 고정

```
for layer in base_model.layers[:-5]:
    layer.trainable = False

base_model.summary()
```

모델의 각 층을 순회하며 가중치를 고정한다.

6. 전이학습

■ 6.8 프로젝트 2: 미세조정

- (5) 새로운 분류기 추가하여 모델 완성

base_model의 출력을 다음 층에 입력한다.

유닛이 10개인 소프트맥스층을 추가한다.

```
last_output = base_model.output

x = Dense(10, activation='softmax', name='softmax')(last_output)

new_model = Model(inputs=base_model.input, outputs=x)

new_model.summary()
```

케라스 Model 클래스의 객체를 new_model이라는 이름으로 생성한다.

모델 객체 new_model의 개요를 출력한다.

6. 전이학습

■ 6.8 프로젝트 2: 미세조정

- (6) 신경망을 컴파일하고 새로운 소규모 데이터셋으로 모델 재학습

- CPU만 사용해도 모델의 학습이 굉장히 빨리 진행되는 것을 볼 수 있음
  - 한 에포크당 약 40초가 걸리니 20 에포크의 학습도 채 15분이 걸리지 않음

```
new_model.compile(Adam(lr=0.0001), loss='categorical_crossentropy',
                  metrics=['accuracy'])

from keras.callbacks import ModelCheckpoint

checkpointer = ModelCheckpoint(filepath='signlanguage.model.hdf5',
                               save_best_only=True)

history = new_model.fit_generator(train_batches, steps_per_epoch=18,
                    validation_data=valid_batches, validation_steps=3,
                    epochs=20, verbose=1, callbacks=[checkpointer])
```

6. 전이학습

6.8 프로젝트 2: 미세조정

- (7) 모델의 성능을 평가
  - 먼저 데이터셋을 텐서로 변환하는 load_dataset() 메서드를 정의

```
def load_dataset(path):
    data = load_files(path)
    paths = np.array(data['filenames'])
    targets = np_utils.to_categorical(np.array(data['target']))
    return paths, targets
```

```
from sklearn.datasets import load_files
from keras.utils import np_utils
import numpy as np

test_files, test_targets = load_dataset('dataset/test')
```

6. 전이학습

6.8 프로젝트
2: 미세조정

- (7) 모델의 성능을 평가

```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from tqdm import tqdm

def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    return np.vstack(list_of_tensors)

test_tensors = preprocess_input(paths_to_tensor(test_files))
```

6. 전이학습

6.8 프로젝트 2: 미세조정

- (7) 모델의 성능을 평가
  - 이제 케라스의 evaluate() 메서드를 실행해서 모델의 정확도를 측정할 수 있음

```
print('\nTesting loss: {:.4f}\nTesting accuracy:
   {:.4f}'.format(*new_model.evaluate(test_tensors, test_targets)))

Testing loss: 0.0574
Testing accuracy: 0.9800
```

6. 전이학습

6.9 마치며

- 전이학습은 분류 또는 물체 인식 프로젝트를 시작하는 출발점으로 유용하다. 특히 학습 데이터를 충분히 확보하지 못했다면 더욱 유용하다.
- 전이학습은 원 데이터셋에서 학습된 지식을 목표 데이터셋으로 옮기는 방법으로 학습에 소요되는 계산 자원이나 학습 시간을 절약하는 효과가 있다.
- 신경망은 층수가 깊어질수록 더욱 복잡한 특징을 학습한다. 뒤쪽에 위치한 층일수록 특정 이미지에 가까운 특징이 학습된다.
- 신경망의 앞쪽 층은 직선, 모서리 등의 저수준 특징을 학습한다. 첫 번째 층의 출력이 두
  번째 층의 입력이 되며 점차 고수준 특징을 학습한다. 다음 층은 이전 층의 출력을 입력받아 대상의 일부로 조합하고 그 뒤로 이어지는 층이 대상을 탐지한다.
- 전이학습 방식은 사전 학습된 신경망을 분류기로 사용하는 방식, 특징 추출기로 사용하는 방식, 미세 조정 이렇게 크게 세 가지다.

6. 전이학습

6.9 마치며

- 사전 학습된 신경망을 분류기로 사용하는 방식은 사전 학습된 신경망을 가중치 고정이나 재학습 없이 그대로 사용하는 방식이다.
- 사전 학습된 신경망을 특징 추출기로 사용하는 방식은 사전 학습된 신경망의 특징 추출기 부분의 가중치를 고정하고 그 외 부분을 재학습해서 사용하는 방식이다.
- 미세 조정은 특징 추출기 부분 중 일부 층의 가중치를 고정하고, 가중치를 고정하지 않은 층과 새로 추가한 분류기 부분을 함께 재학습하는 방식이다.
- 학습된 특징을 다른 신경망으로 옮기는 과정의 성공 가능성은 목표 데이터셋의 규모, 원 도메인과 목표 도메인의 유사성에 따라 결정된다.
- 일반적으로 미세 조정 중의 재학습에는 신경망 부분별로 학습률을 달리 설정한다. 기존 신경망의 가중치를 고정하지 않은 부분은 학습률을 작게 설정하고, 새로 추가한 분류기 부분은 학습률을 크게 설정한다.