5장. 컨볼루션 신경망

Wrap Up

1. 특정 분야가 아닌 이상, 문제에 사용되는 대표적인 데이터셋은 분명히 존재합니다. 어느 부분부터 접근해야 할지 모르겠다면, **해당 문제에 사용되는 대표적인 데이터셋과 문제에 적용된 모델을 벤치마킹**하는 것이 가장 빠른 접근 방법일 수 있습니다.
2. 신경망은 **스케일에 매우 민감하므로 적절한 전처리 과정은 필수**입니다.
3. 이진 분류: sigmoid + binary_crossentropy
   다중 분류: softmax + categorical_crossentropy
   회귀 문제: mse + mae
   다중 레이블 분류: sigmoid + binary_crossentropy
4. 모델의 **History 객체**를 활용하면 학습 과정을 더욱 직관적으로 관찰할 수 있습니다.
5. **데이터가 복잡하지 않고 충분하지도 않을 때, 모델을 깊게 구성하면 과대적합에 크게 노출될 수 있습니다.**
6. 데이터가 충분하지 않을 때, **교차 검증**은 이를 보완할 좋은 방법입니다.
7. 모델의 성능을 극적으로 향상시킬 수 있는 방법은 **데이터의 특성을 잘 파악하는 것**입니다.
8. **캐글**은 이러한 모든 과정을 경험할 수 있는 최고의 공간입니다.

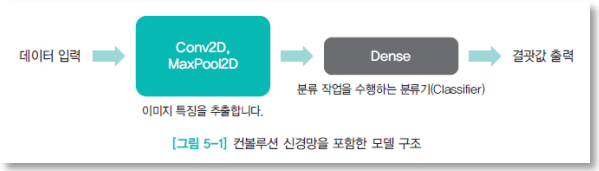
5장의 내용?

- 이미지 데이터를 다루는 **컴퓨터 비전(Computer Vision) 분야**에서 매우 활발하게 사용되고 있는 컨볼루션 신경망
  - '컴퓨터 비전 분야에서'라는 수식어는 이제 옛말
  - 텍스트, 시계열 데이터 등에서도 월등한 성능을 보여주고 있음
  - 그럼에도 컨볼루션 신경망이 가장 빛나는 순간은 이미지 데이터를 다룰 때

- 이번 장에서는 다음 내용을 다뤄봅니다.
  - 컨볼루션 신경망: 컨볼루션과 풀링의 개념
  - 과대적합 예방하기: 규제화 함수, 드롭아웃, 배치 정규화
  - 모델을 견고하게 만들기: 데이터 증식
  - 딥러닝의 매우 강력한 장점: 전이 학습

일단 사용해보기

- 컨볼루션과 풀링 개념을 배우기 전에 먼저 사용해보자
  - 데이터셋: Fashion-MNIST, 전처리 과정 동일
  - 모델은 Conv2D, MaxPool2D, Dense 층으로 구성



[그림 5-1] 컨볼루션 신경망을 포함한 모델 구조

[함께 해봐요] **모델 구성하기** fashion_mnist_cnn.ipynb

```
01 from tensorflow.keras.models import Sequential
02 from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten
03
04 model = Sequential([
05     Conv2D(filters = 16, kernel_size = 3, padding = 'same',
               activation = 'relu', input_shape = (28, 28, 1)),
06     MaxPool2D(pool_size = (2, 2), strides = 2, padding = 'same'),
07     Conv2D(filters = 32, kernel_size = 3, padding = 'same', activation = 'relu'),
08     MaxPool2D(pool_size = (2, 2), strides = 2, padding = 'same'),
09     Conv2D(filters = 64, kernel_size = 3, padding = 'same', activation = 'relu'),
10     MaxPool2D(pool_size = (2, 2), strides = 2, padding = 'same'),
11     Flatten(),
12     Dense(64, activation = 'relu'),
13     Dense(10, activation = 'softmax')
14 ])
```

  - 위와 같이 Sequential()에 리스트 형태로 층을 제공하여 모델을 구성할 수 있음
    - 제공되는 코드는 add()를 주로 사용

일단 사용해보기

- 결과 비교
  - 4.1.5 Fashion-MNIST에서 Dense 층으로 구성한 모델 정확도: **88~89%**
  - 컨볼루션 신경망을 활용하여 구성한 모델 정확도: **약 92%**

```
Epoch 30/30
42000/42000 [==============================] - 3s 83us/sample - loss: 0.0595 -
acc: 0.9781 - val_loss: 0.3235 - val_acc: 0.9178
```

- 결과 비교를 통해 Fashion-MNIST 데이터셋에서는 컨볼루션 신경망으로 높은 성능을 얻기가 더 수월하다는 것을 알 수 있음
  - 주의! 이번 실험을 통해 컨볼루션 신경망의 높은 성능을 알 수 있지만,
    Dense 층이 이미지 데이터에서 잘 작동하지 않는다는 편협한 사고를 가지면 안됨!
    (ViT, Vision Transformer가 그 예시로 대표적)

컨볼루션 층

- **컨볼루션층**과 풀링층

- 컨볼루션층 사용 이유
  - 완전연결층과의 차이
  - 컨볼루션 필터

1. 완전연결층(Fully-Connected Layer)과의 차이

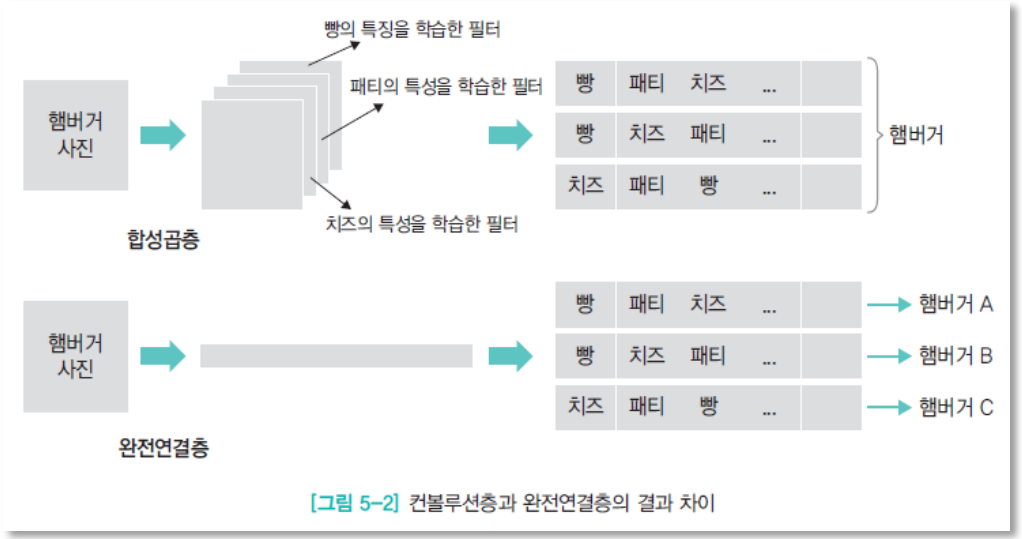
☞ 완전연결층은 1차원 배열 형태를 통해 학습함

- 단순히 데이터를 펼쳐서 사용하기 때문에 **이미지 픽셀 사이의 관계를 고려하지 않음**
- 2차원 형태 데이터를 1차원 형태로 변환하면서, **본래 데이터 특성을 잃어버리게 됨**

컨볼루션 층

- 완전연결층은 햄버거 패티, 치즈, 양상추 등을 학습하는 것이 아닌 햄버거 전체(**전역 특징**)를 학습
  - 패티의 위치나 내용물이 달라지는 경우 서로 다른 햄버거로 인식함
  - **" 공간 정보를 손실한다"**
  - 은닉 유닛 수를 늘려 해결 → But, 과대적합 문제 노출

- 컨볼루션층은 이미지 픽셀 사이의 관계를 고려하기 때문에 햄버거 패티, 치즈, 양상추 등의 **지역적 특징**을 학습
  - 지역적 특징? 빵 밑에 패티가 있고, 다시 그 밑에 패티가 있거나 또는 사람 얼굴에서 눈 옆에 코가 있고, 그 밑에 입이 있는 것
  - **"공간 정보를 유지한다"**
  - 완전연결층에 비해 적은 파라미터 수를 요구



[그림 5-2] 컨볼루션층과 완전연결층의 결과 차이

컨볼루션 층

- 요구되는 파라미터 수 차이
  - 단편적 관점으로 각각 1개씩의 극단적인 Dense, Conv2D 층을 보았을 때

```
model = Sequential()
model.add(Dense(128,
          activation = 'relu',
          input_shape = (784, )))
```

→ 128개의 은닉 유닛을 사용한 Dense층의 파라미터 수: 100,480개

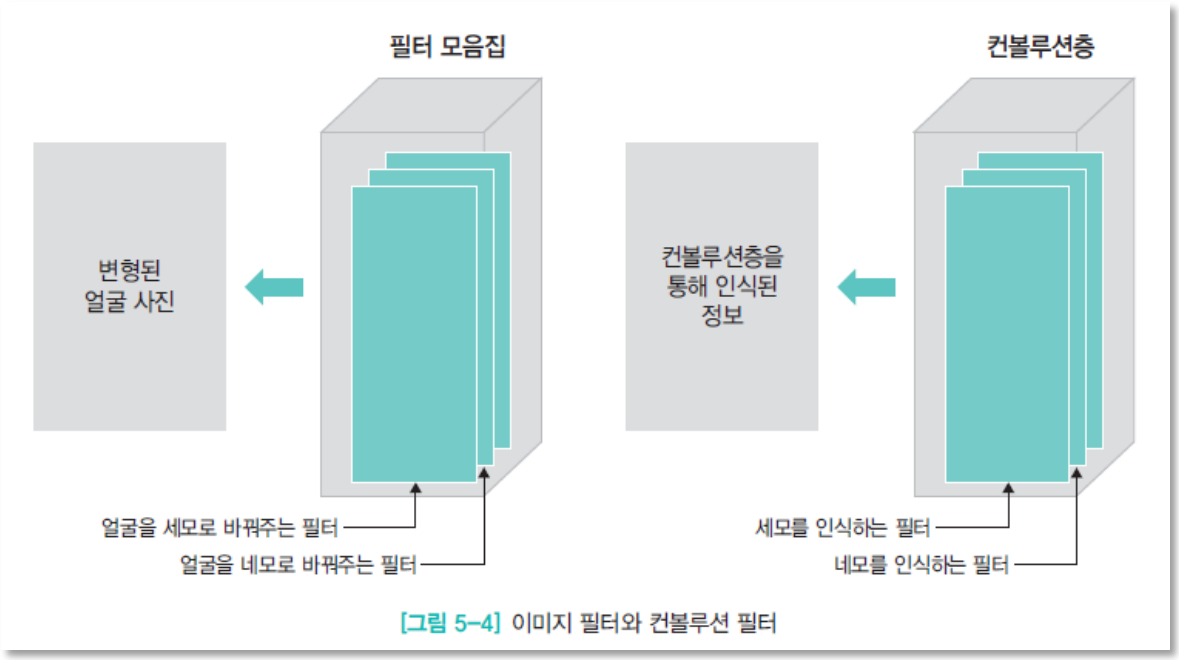
```
model = Sequential()
model.add(Conv2D(1000, 3,
          activation = 'relu',
          input_shape = (28, 28, 1)))
```

→ 1000개의 합성곱 필터를 사용한 Conv2D층의 파라미터 수: 1,280개

[그림 5-3] 파라미터 수: Dense층 vs. Conv2D층

컨볼루션 층

2. 컨볼루션 필터
   - "필터(Filter)" 개념 사용
   - '셀카를 찍고 난 후, 특정 사진 앱을 이용하여 얼굴을 변형해주는 그런 필터?'
   - 사진 앱은 얼굴을 변형하기 위한 특정 필터를 사용했을 것 → **이미지 필터(Image Filter)**
   - 이와 같이 컨볼루션층은 여러 개의 컨볼루션 필터를 활용하여 이미지 내에 존재하는 다양한 정보를 인식할 수 있음



[그림 5-4] 이미지 필터와 컨볼루션 필터

컨볼루션 층

- 이미지 필터와 컨볼루션 필터
  - 공통점: 필터가 가지고 있는 파라미터를 통해 목적을 달성
  - 차이점: 이미지 필터는 **직접 정의**, 컨볼루션 필터는 **학습을 통해 파라미터가 조정됨**

- 얼굴 변형의 목적을 가진 이미지 필터는 이미지 선명함을 목적으로 하는 필터로써 사용할 수 없음
  - 두 가지 목적을 동시에 달성하려면 독립적으로 각각 필터를 직접 정의

- **컨볼루션 필터는 모델 학습을 통해 필터의 파라미터가 목적에 맞게끔 조정**
  - 학습된 모델을 사용하면 얼굴 변형, 선명함 등 목적을 동시에 사용할 수 있음
  - Feature Extraction

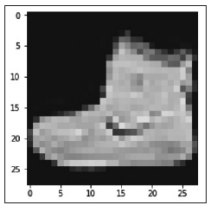
- 이미지 필터를 실습해보자!
  - 가장자리 검출(Edge-Detection)에서 매우 유명한 소벨 필터(Sobel-Filter)를 사용
  - 소벨 필터의 파라미터를 직접 정의할 것

이미지 필터 사용해보기

- 사용 데이터 선정
  - Fashion-MNIST 데이터셋의 첫 번째 데이터: 신발

[함께 해봐요] 이미지 필터 사용해보기 use_image_filter.ipynb

```
01 from tensorflow.keras.datasets import fashion_mnist
02
03 # 데이터를 다운받습니다.
04 (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
05
06 import matplotlib.pyplot as plt
07 plt.imshow(x_train[0])  # 첫 번째 데이터를 그려봅니다.
```



- 신발 가장자리를 검출해볼 것, 이를 위해 이미지 필터를 정의

1	2	1
0	0	0
–1	–2	–1

가로선 필터

1	0	–1
2	0	–2
1	0	–1

세로선 필터

[그림 5-5] 가로선 필터와 세로선 필터

```
04 # 가로선을 추출하기 위한 필터
05 horizontal_filter = np.array([[1., 2., 1.],
06                              [0., 0., 0.],
07                              [-1., -2., -1.]])
```

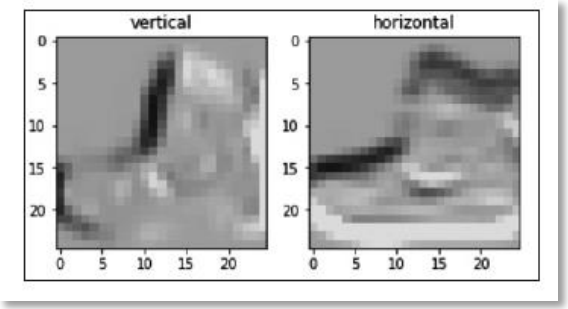
```
09 # 세로선을 추출하기 위한 필터
10 vertical_filter = np.array([[1., 0., -1.],
11                            [2., 0., -2.],
12                            [1., 0., -1.]])
```

이미지 필터 사용해보기

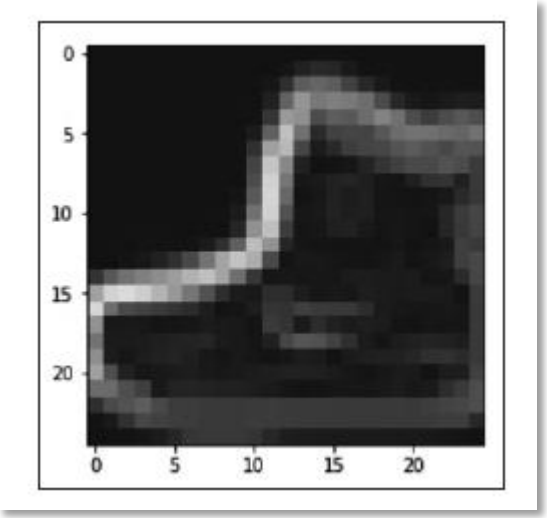
- 컨볼루션 연산을 사용하여 가장자리를 검출

```
07              # 컨볼루션 연산
08              indice_image = test_image[i:(i + filter_size),
                                          j:(j + filter_size)] * filter
```

- 가로선과 세로선을 검출



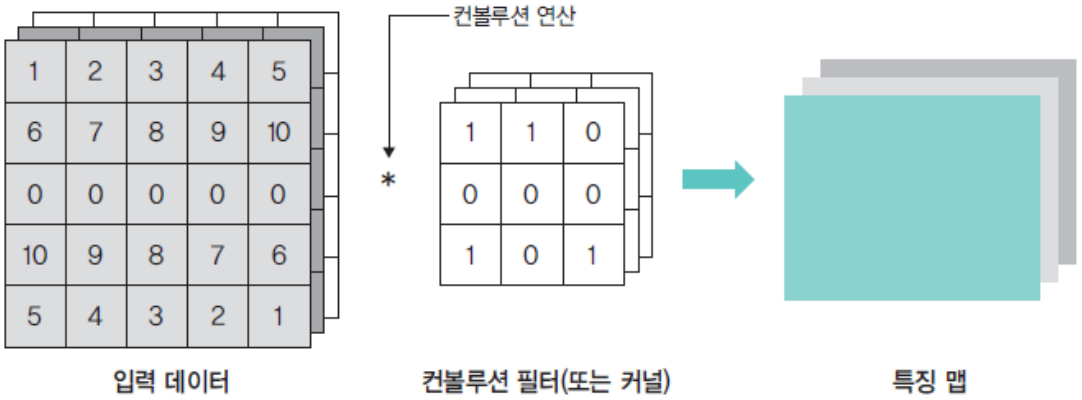
- 둘을 더하면 우리가 원했던 신발의 전체 가장자리가 검출!
  - 파라미터를 직접 정의했음
  - 또 다른 특징을 검출하고 싶다면, 이를 위한 필터를 다시 정의해주어야 함
  - 엄청나게 많은 수의 조합 → 엄청나게 많은 실험
  - 필터를 잘못 정의한다면? → 전혀 다른 결과 문제 발생



컨볼루션 알아보기

- 컨볼루션층에서 주로 사용되는 용어
  - 컨볼루션 연산: Convolution
  - 스트라이드: Stride
  - 패딩: Padding

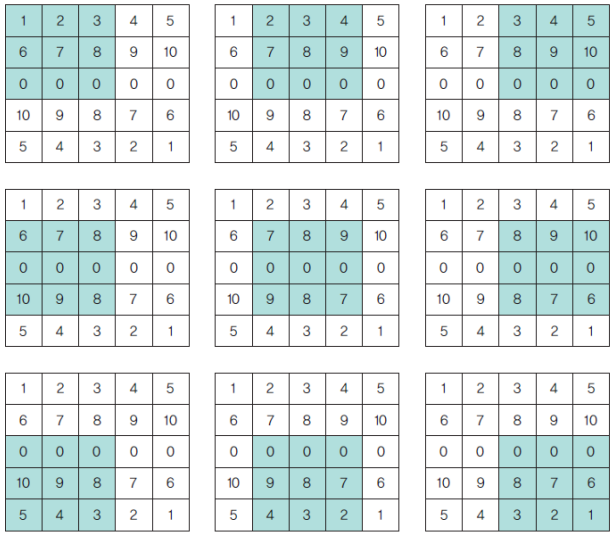
- **컨볼루션 연산과 스트라이드**
  - 컨볼루션층은 주어진 입력 데이터에서 **컨볼루션 필터**를 활용하여 **원소별 곱과 윈도우 슬라이딩(Window Sliding)**을 행하는 **컨볼루션 연산**을 통해 **특징맵(Feature Map)**을 만듦



[그림 5-6] 입력 데이터 → (컨볼루션 연산 + 컨볼루션 필터) → 특징맵

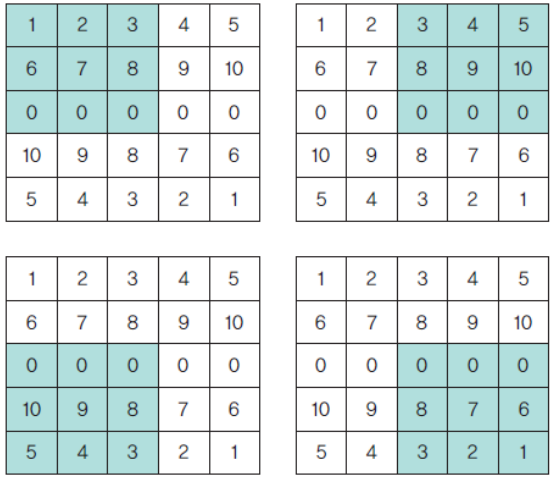
컨볼루션 알아보기

- 아래 예에서 사용한 필터 크기는 5x5, 7x7보다 **대표적으로 사용되는 3x3 크기**
  - ex) 1x1 스트라이드, 3x3 필터 크기(좌)와 2x2 스트라이드, 3x3 필터 크기(우)
  - 스트라이드 크기는 오른쪽 그림처럼 다양한 크기(1x2, 2x1 등)을 사용할 수 있고, 1x1보다 큰 크기의 스트라이드를 사용하여 다운샘플링 효과를 볼 수 있지만, **다운샘플링을 위해서는 이후에 살펴볼 풀링 연산을 주로 활용함**
  - **다운샘플링(Down Sampling): 신경망의 파라미터 수를 감소시키는 효과**



[그림 5-7] 1×1 스트라이드와 3×3 필터를 사용하는 컨볼루션 연산

입력 크기: (5,5)
출력 크기: (3,3)



[그림 5-8] 2×2 스트라이드와 3×3 필터 형태를 사용하는 컨볼루션 연산

입력 크기: (5,5)
출력 크기: (2,2)

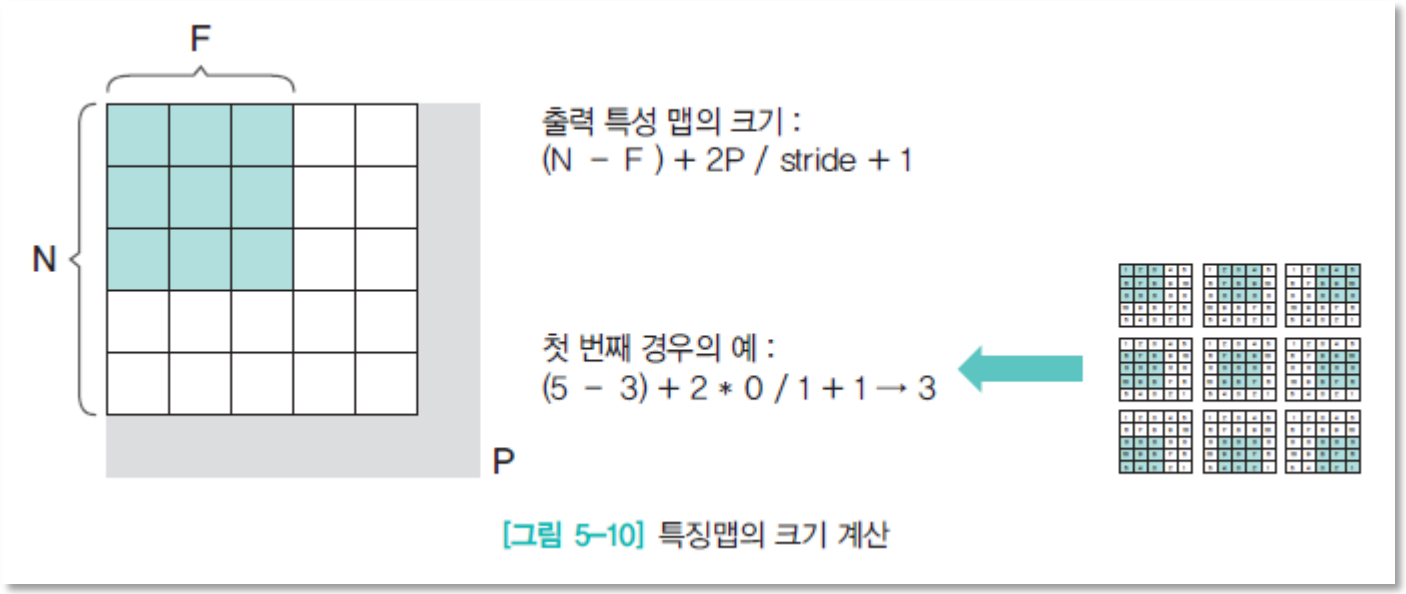
컨볼루션 알아보기

- 패딩(Padding)
  - 특징맵의 크기가 감소하지 않고, **입력 데이터의 형태와 동일한 형태를 출력값으로 얻고 싶은 경우**
  - **이미지의 가장자리에 해당하는 정보 손실 방지**

[그림 5-9] 1 크기의 패딩을 추가한 예

컨볼루션 알아보기

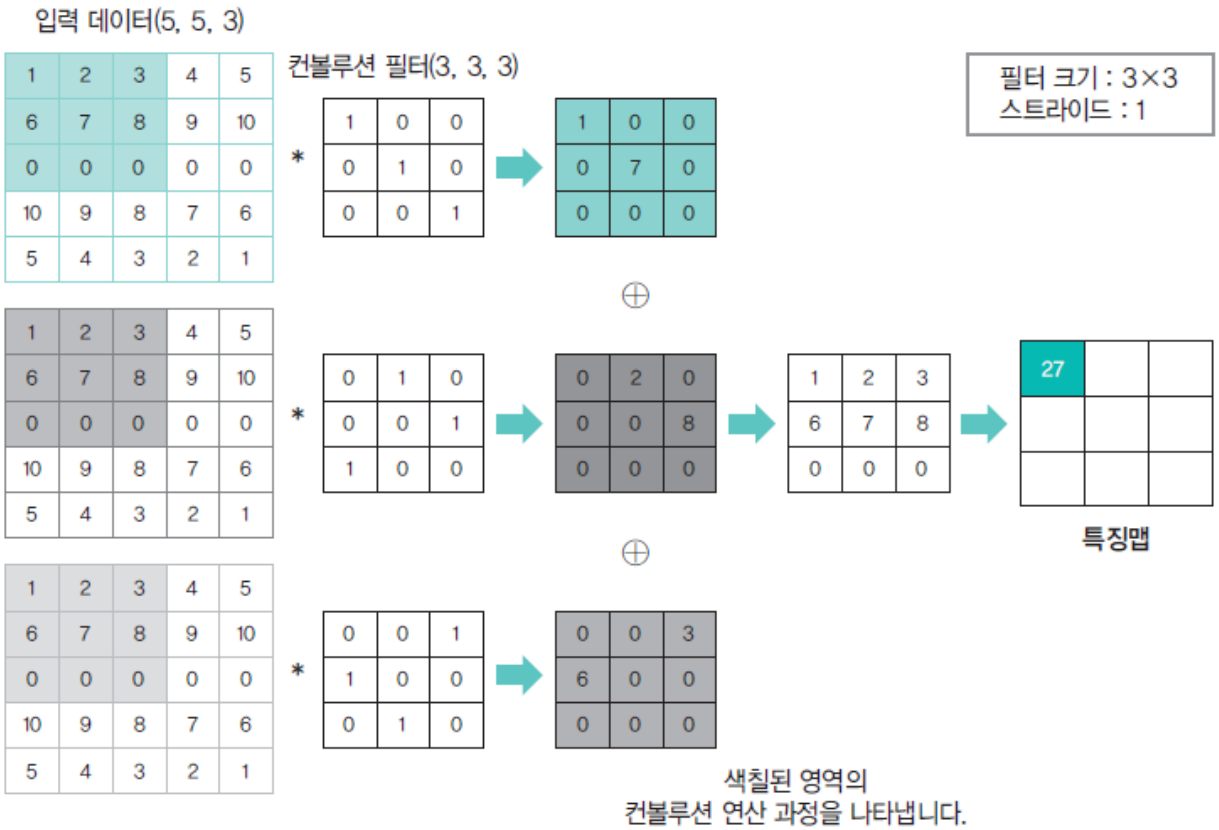
- 컨볼루션층 결괏값, 특징맵 크기 계산해보기
  - 컨볼루션층이 출력하는 특징맵 크기를 확인해보고 싶을 때
  - But, 케라스는 모델을 구성하는 각 층의 입력과 출력을 직접 계산해줌
  - N(입력 데이터 크기), F(필터 크기), P(패딩 크기)



[그림 5-10] 특징맵의 크기 계산

컨볼루션 알아보기

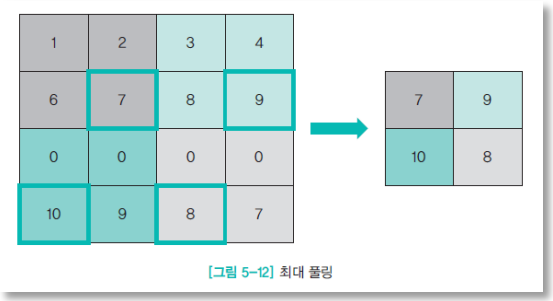
- 컨볼루션 연산을 통해 특징맵의 파라미터가 어떻게 생성되는지 직접 계산해보세요!



[그림 5-11] 컨볼루션 연산

풀링 알아보기

- 풀링(Pooling)
  - 평균 풀링(Average Pooling)
  - **최대 풀링(Max Pooling)** ← 주로 사용



[그림 5-12] 최대 풀링

- 일반적으로 2x2 스트라이드, 2x2 윈도우 크기를 사용하여 특징맵 크기를 절반으로 감소시킴 (다운샘플링)
  - 최대 풀링 연산은 해당 윈도우에서 가장 큰 값을 특징값으로 선택
  - 모델이 물체의 주요한 특징을 학습할 수 있도록 도와주며, 컨볼루션 신경망이 **이동 불변성(Translation or Shift invariant)**을 가지도록 함
    - 이동 불변성은 물체가 어느 위치에 있어도 해당 물체를 인식할 수 있다는 것을 의미
  - **모델 파라미터 수 감소**
    - 계산 속도 향상
    - 과대 적합 노출 가능성 감소

[그림 5-12] 이동 불변성

풀링 알아보기

- 풀링 연산에는 최대(Max) 풀링뿐만 아니라 평균(Average) 풀링도 존재
  - 평균 풀링은 각 윈도우에 속하는 값들을 평균내어 이를 특징값으로 사용
  - 하지만 특정 물체의 존재 여부를 알기 위해서는 가장 명확한 특징을 사용하는 것이 좋다고 알려져 있음

- **최대 풀링층을 사용한다면, 컨볼루션 층에서는 1x1 스트라이드 사용을 권장**
  - 최대 풀링층이 강한 특징값을 뽑아냄과 동시에 다운샘플링을 수행하기 때문에, 컨볼루션 층에서는 가급적 이면 정보를 보존하는 것이 좋음

- 풀링 연산 구현

```
14  for x in range(0, image_x, 2):
15        for y in range(0, image_y, 2):
16            pooled_image[int(x/2), int(y/2)] = np.max(image[x:x + 2, y:y + 2])
```

모델 다시 살펴보기

- 앞서 구성한 모델은 [Conv2D → MaxPool2D] 패턴을 사용하였음

예시: 컨볼루션 신경망

```
01  model = Sequential([
02      # 항상 모델의 첫 번째 층은 입력의 형태를 명시해주어야 합니다.
03      Conv2D(filters = 16, kernel_size = 3, strides = (1, 1),
               padding = 'same', activation = 'relu', input_shape = (28, 28, 1)),
04      MaxPool2D(pool_size = (2, 2), strides = 2, padding = 'same'),
05      Conv2D(filters = 32, kernel_size = 3, strides = (1, 1),
               padding = 'same', activation = 'relu'),
06      MaxPool2D(pool_size = (2, 2), strides = 2, padding = 'same'),
07      Conv2D(filters = 64, kernel_size = 3, strides = (1, 1),
               padding = 'same', activation = 'relu'),
08      MaxPool2D(pool_size = (2, 2), strides = 2, padding = 'same'),
09      Flatten(),  # Dense층에 입력하기 위해 데이터를 펼쳐줍니다.
10      Dense(64, activation = 'relu'),
11      Dense(10, activation = 'softmax')  # 열 개의 출력을 가지는 신경망
12  ])
```

모델 다시 살펴보기: Conv2D

- **Conv2D(filters = 16, kernel_size = 3, strides = (1, 1), padding = 'same', activation = 'relu', input_shape = (28, 28, 1))**
  - **filters:** 특징맵 차원을 결정, filters = 16은 16개의 컨볼루션 필터를 사용한다는 의미이며, 특징맵 형태는 (batch_size, rows, cols, filters)가 됩니다.
  - **kernel_size:** (3,3)과 같이 튜플 형태로 필터 크기를 설정. 위 경우처럼 하나의 숫자 k를 전달할 경우 자동으로 (k, k) 필터 크기로 설정됩니다.
  - **strides:** 스트라이드 크기를 지정합니다. 기본값은 (1, 1)으로 kernel_size와 같이 하나의 숫자 형태로 제공할 수 있습니다.
  - **padding:** 패딩에 대한 결정 여부를 지정합니다. 'same'은 패딩을 사용하여 입출력 형태가 동일하도록 하며, 'valid'는 패딩을 사용하지 않습니다. 기본값은 'valid'입니다.
  - **activation:** 사용할 활성화 함수를 문자열 또는 클래스 형태로 제공합니다.

모델 다시 살펴보기: MaxPool2D

- **MaxPool2D(pool_size = (2, 2), strides = 2, padding = 'same')**
  - **pool_size:** 풀링층에서 사용할 커널의 크기를 설정합니다. Conv2D의 kernel_size처럼 하나의 숫자 형태로 제공할 수 있습니다.
  - **strides:** 스트라이드 크기를 지정합니다. 기본값은 None입니다. 이 값이 주어지지 않는 경우, pool_size 크기와 동일한 크기로 지정됩니다. 예를 들어, pool_size = (2,2)이고 스트라이드가 None 이라면, 실제 최대 풀링층의 스트라이드는 (2,2)로 적용됩니다.
  - **padding:** Conv2D층의 내용과 동일합니다.

모델 구조를 확인하는 방법

- summary()와 plot_model() 함수

[함께 해봐요] plot_model() 함수 사용하기

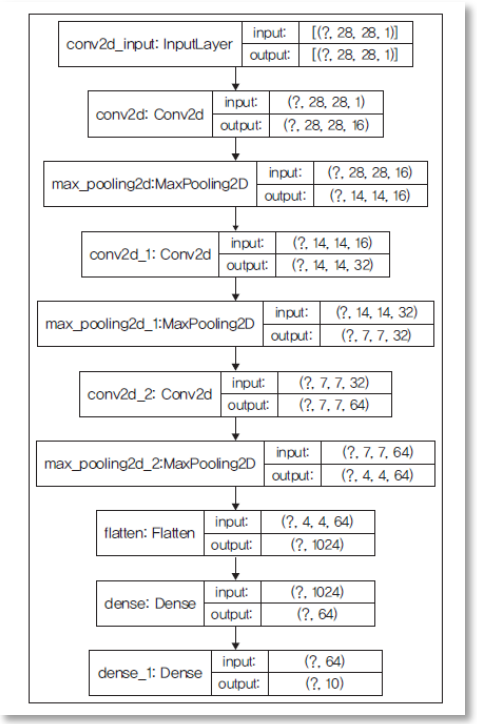
```
# 6번 각주의 해결 방법을 써도 문제가 해결되지 않는다면, 다음 코드의 주석을 풀어 실행시키세요.
# import os
# os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin'

from tensorflow.keras.utils import plot_model
plot_model(model, './model.png', show_shapes=True)
```

[함께 해봐요] model.summary() 함수 사용하기

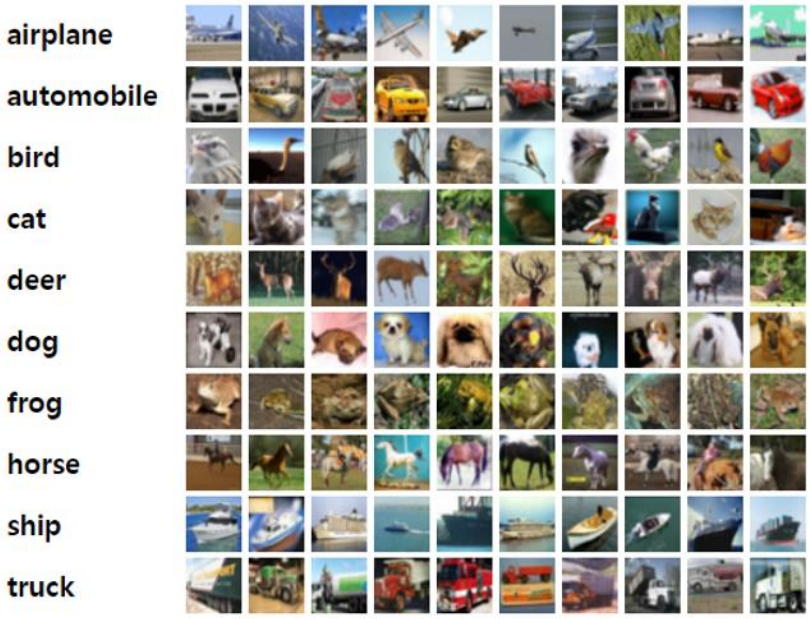
```
model.summary()  # 모델의 구조를 확인합니다.
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 64)          18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 89,546
Trainable params: 89,546
Non-trainable params: 0
_________________________________________________________________
```



CIFAR-10 살펴보기

- CIFAR-10 데이터셋
  - **10개 클래스**로 이루어져 있으며, **CIFAR-100은 100개의 클래스**로 이루어져 있음
  - MNIST 데이터셋과 함께 기본적으로 사용되는 데이터셋이지만, MNIST 데이터셋만큼의 성능을 기대하기 어려움
  - 50,000개 학습 데이터와 10,000 테스트 데이터



그림·5-15·CIFAR-10의·클래스·유형

CIFAR-10 살펴보기

- 신경망 모델의 입력으로 사용하기 위한 전처리 과정 수행
  - 채널별로 평균과 표준편차를 구함

```
01  # 평균과 표준 편차는 채널별로 구해줍니다.
02  x_mean = np.mean(x_train, axis = (0, 1, 2))
03  x_std = np.std(x_train, axis = (0, 1, 2))
04
05  x_train = (x_train - x_mean) / x_std
06  x_test = (x_test - x_mean) / x_std
```

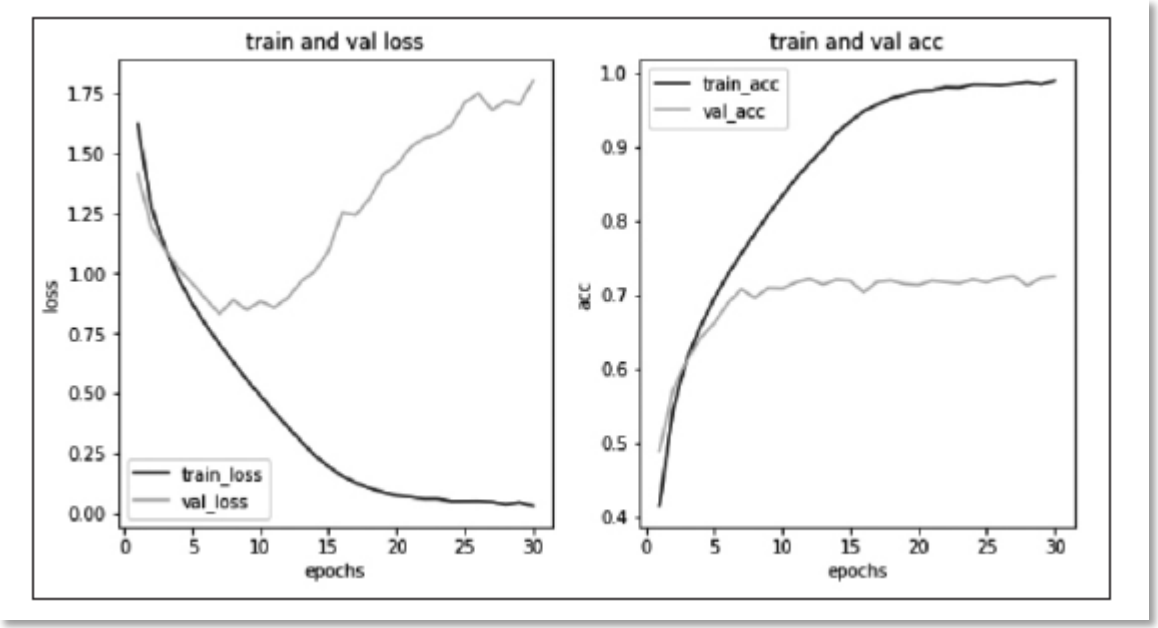
- 모델 구성
  - Conv2D, MaxPool2D
  - Adam(0.0001), sparse_categorical_crossentropy
  - **sparse_categorical_crossentropy**는 0~9 형태로 되어있는 레이블을 그대로 사용할 수 있게 해줌

```
08  model.add(Conv2D(filters = 32, kernel_size = 3, padding = 'same',
                     activation = 'relu'))
09  model.add(MaxPool2D(pool_size = (2, 2), strides = 2, padding = 'same'))
```

CIFAR-10 살펴보기(실습)

- 제공되는 코드를 통해 모델을 학습시켜보세요.

- 결과 확인
  - 과대적합 문제 발생!
  - 예방할 수 있는 방법을 알아보자



설명 가능한 AI(XAI)

- 신경망의 가장 큰 단점 중 하나는 **블랙박스인 모델을 쉽게 해석할 수 없다는 것**
  - **(고민)** 굳이 해석해야 하는가?
  - **(고민)** 자연의 이치와 같이 해석하지 않아도 좋은 성능 그대로를 사용한다면?
  - 그래도 해석해보자!

- 아무리 모델 성능이 좋다고 할지라도 왜 성능이 좋은지를 알지 못하면, 향후 모델의 견고함과 일반화를 위한 실험 방향 설정과 실제 서비스나 연구에서 신뢰성이 떨어지는 결과를 제공할 수 있음
  - ex) 환자가 병원에서 AI를 통해 진료받을 때, 의사가 결과를 설명하지 않고 단순히 AI의 판독 결과만 통보한다면?
  - 신뢰할 수 없는 결과: 이를 설명하지 못한 의사의 잘못? 해석 불가능한 AI의 잘못?
  - 이러한 문제를 해결하기 위해 연구되고 있는 분야: **설명 가능한 AI(XAI; Explainable AI)**

    ☞ **하지만 지금은? 잘되면 Good! 꼭 해석할 수 있는 모델만이 정답은 아님.**

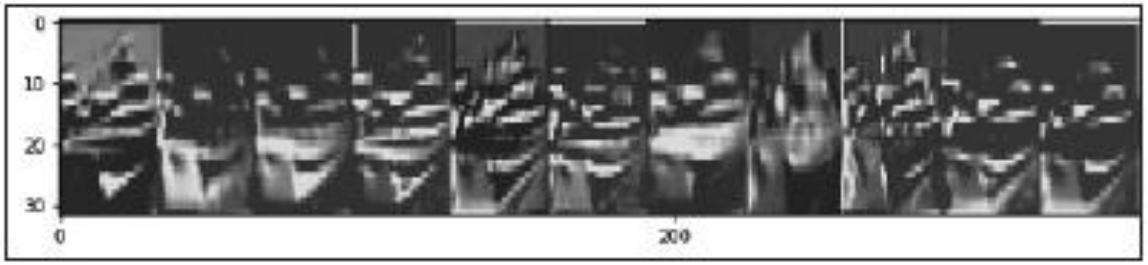
신경망 시각화

- 이미지에서 모델이 인식하는 특징을 확인하는 방법
  - 구체적인 것은 코드를 참고

```
06  # 모델 전체에서 output을 가져올 수 있습니다.
07  visual_model = tf.keras.models.Model(inputs = model.input, outputs = get_output)
```

  - 모델이 포함하고 있는 Conv2D, MaxPool2D층에서 특징을 뽑아서 확인할 수 있음

    ex) 배

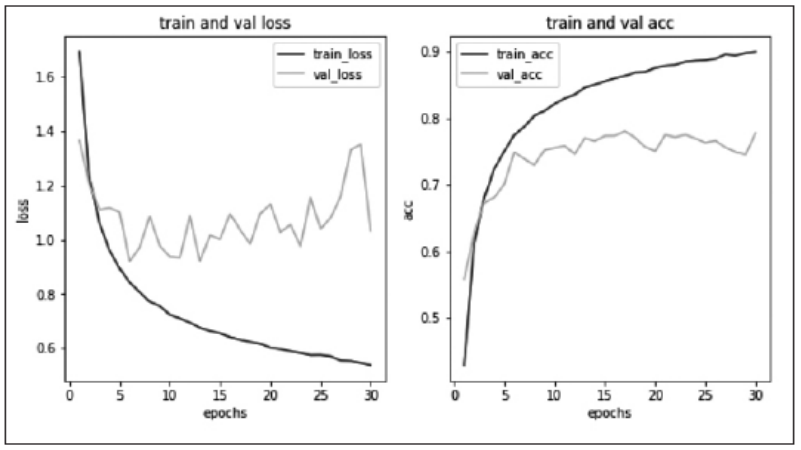


과대적합 피하기

- 과대적합을 방지할 수 있는 2+1가지 방법
  - **여기서 설명할 방법은 예방책일 뿐, 100% 해결해주지 않음**
  1. 규제화 함수(Regularizer)
  2. 드롭아웃(Dropout)
  3. 배치정규화(BatchNormalization)

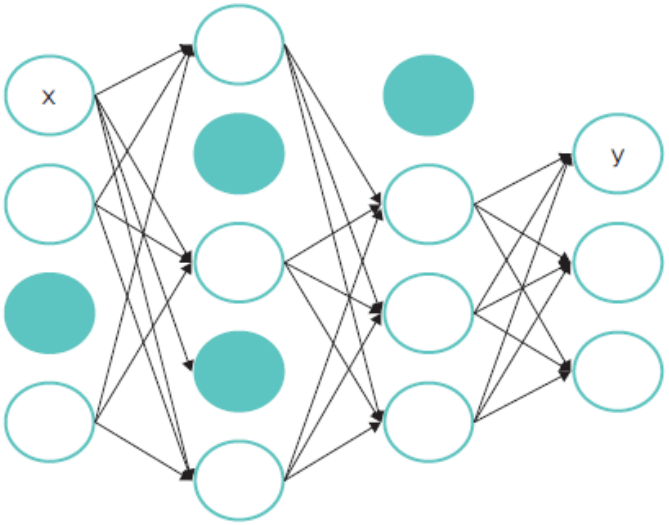
- 규제화 함수 (Regularizer)
  - **임의로 모델 복잡도를 제한**시키는 방법
  - L1 노름, L2 노름, 엘라스틱넷(ElasticNet)이 존재, 가중치 감쇠(Weight Decay)라고도 표현
  - 규제화 함수는 기능에 맞게 가중치의 합을 구하여 손실함수에 더해줌
  - 사용하지 않은 것보다 안정적으로 그래프가 그려짐

```
08  model.add(Conv2D(filters = 32, kernel_size = 3, padding = 'same',
                     activation = 'relu', kernel_regularizer = l2(0.001)))
```



과대적합 피하기

- 드롭아웃 (Dropout)
  - **학습이 진행되는 동안 신경망의 일부 유닛을 제외(드롭)**
  - 테스트 시 드롭아웃이 작동하지 않고 모든 유닛이 활성화되는 대신, 출력값을 드롭아웃 비율만큼 감소시킴
  - 드롭아웃 비율(Dropout Rate)는 일반적으로 0.2~0.5를 사용



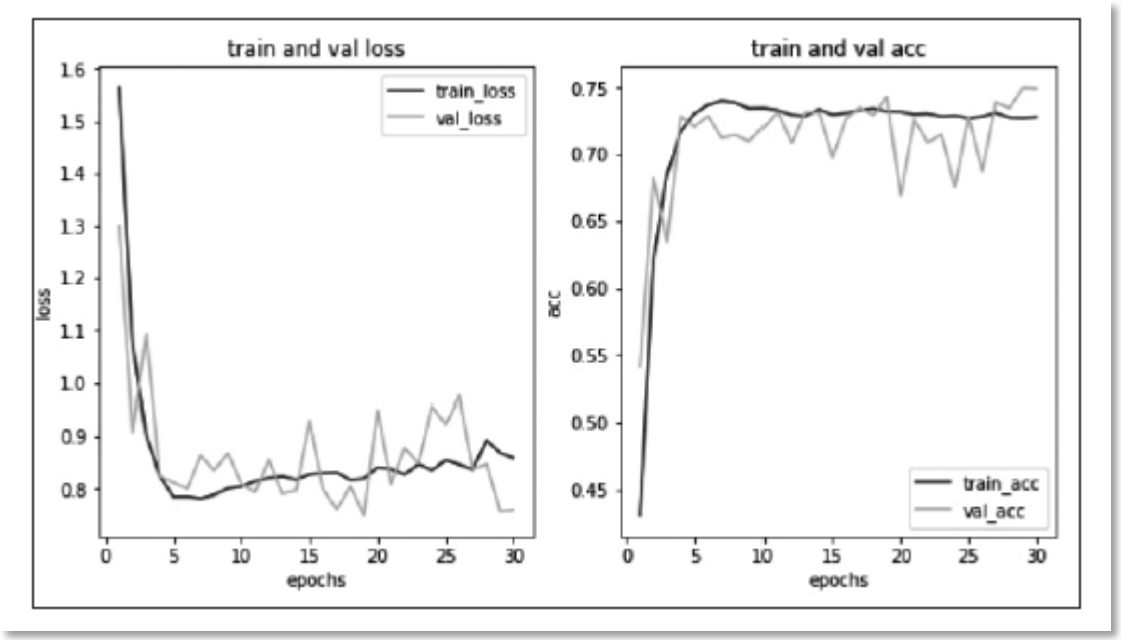
[그림 5-15] 드롭아웃

과대적합 피하기

- 드롭아웃 (Dropout)
  - 제공되는 코드 참고

```
09  model.add(Dropout(0.2))  # 드롭아웃을 추가합니다.
```

  - 결과 확인
    - 효과가 매우 강력해보임
    - 학습 속도를 느리게 하는 단점이 존재



과대적합 피하기

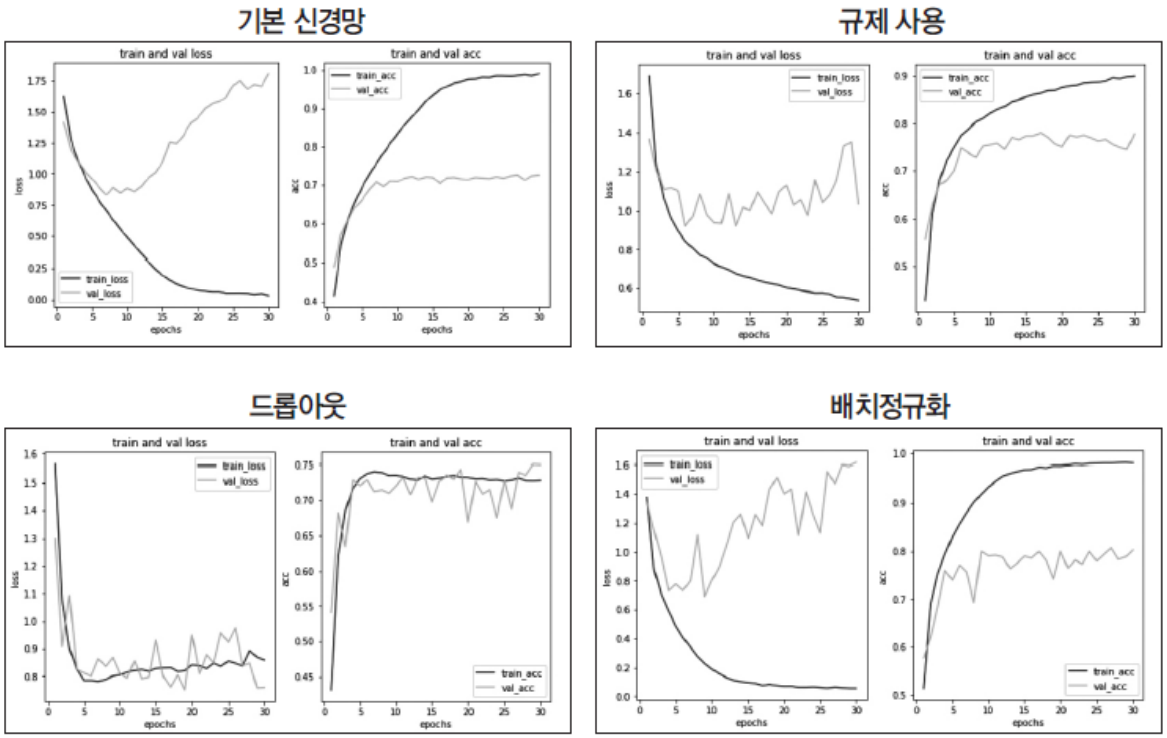
- 배치 정규화 (BatchNormalization)
  - 근본적으로 과대적합을 피하기 위한 방법으로 설명되지 않으나 드롭아웃과 비교되어 사용
  - **내부 공선성(Internal Covariance Shift)**를 해결하기 위해 고안된 방법
  - 신경망 층의 출력값은 다양한 입력 데이터에 따라 쉽게 변할 수 있는데, 매우 큰 범위의 출력값은 신경망을 불안정하게 할 수 있음. 따라서, 이 범위를 제한시켜 불확실성을 감소시키는 방법
  - 그래디언트 손실/폭발 없이 **높은 학습률을 사용할 수 있음**
  - **자체적인 규제화 효과가 포함**되어 있음. 보장하진 않으나 "이를 사용하면 별도의 규제화 함수나 드롭아웃을 사용하지 않아도 된다"라는 의견이 다수

- 배치 정규화 사용 순서
  - Dense층 또는 Conv2D층 → BatchNormalization() → Activation()
  - 최근 BatchNormalization() 함수를 층의 가장 앞에 사용되는 패턴도 사용되고 있음

```
09  model.add(Conv2D(filters = 32, kernel_size = 3, padding = 'same'))
10  model.add(BatchNormalization())
11  model.add(Activation('relu'))
```

과대적합 피하기

- 전체 결과
  - 그래프가 벌어지지 않는 것: 드롭아웃
  - 배치 정규화는 과대적합이 발생했지만, 가장 높은 성능을 보여줌
  - **하지만 "드롭아웃은 과대적합에 매~우 강력하다거나 배치 정규화를 사용하면 무조건 높은 성능을 얻을 것이다" 라는 편협적 시각은 절대 XXX!**



[그림 5-16] 예제 결과 모음

데이터 증식 사용하기

- 데이터 증식(Data Augmentation)을 사용한 성능 향상
  - 딥러닝의 고질적인 문제: 일반화(Generalization)의 해결책, But 근본적으로는 해결 불가

- 데이터 증식의 장점
  - 다양한 데이터를 입력시킴으로써 모델을 더욱 **견고하게** 만들어주기 때문에 테스트 시에 더 높은 성능 기대
  - 수집된 데이터가 적은 경우 강력한 힘 → **일반화**

- 데이터 증식은 모델 성능에 큰 영향을 끼치기 때문에 관련 연구가 활발하게 진행되고 있음
  - Augmentation
  - Auto Augmentation

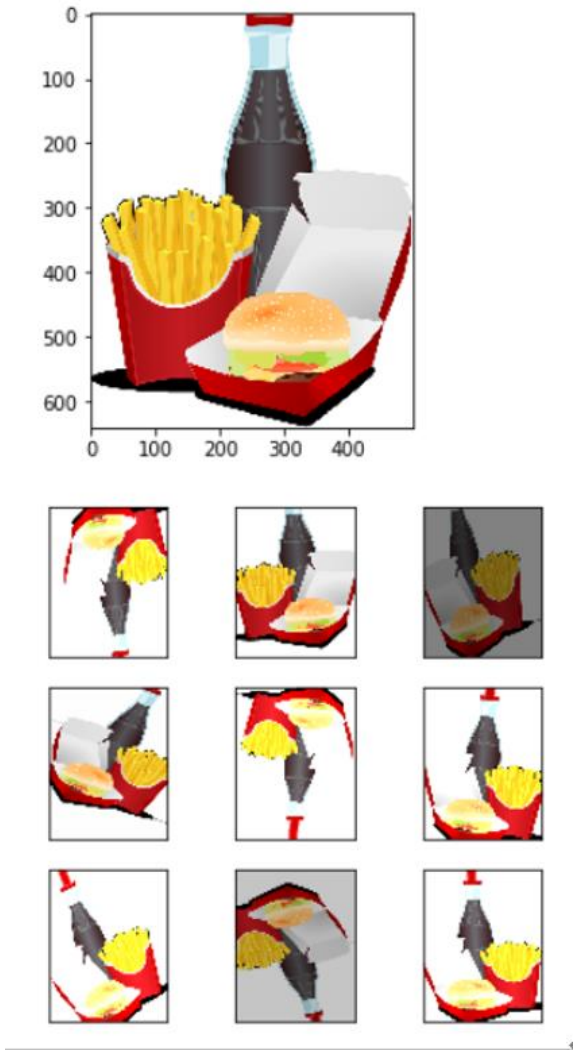
데이터 증식 사용하기

- 케라스는 이를 편리하게 사용할 수 있도록 **이미지 제네레이터(Image Generator)**를 제공하며, 변환 방식은 다음과 같음
  - width_shift_range: 임의의 크기만큼 너비 방향으로 이동시킵니다.
    - 0.2이고 이미지의 너비가 100이라면, -20~+20의 범위에서 너비 방향으로 이동시킵니다.
  - height_shift_range: 임의의 크기만큼 높이 방향으로 이동시킵니다.
    - 0.2이고 이미지의 높이가 100이라면, -20~+20의 범위에서 높이 방향으로 이동시킵니다.
  - brightness_range: 이미지의 밝기 정도를 조정합니다.
    - (0.5, 1.5)이면 원본 대비 최대 50%의 비율로 어둡거나 밝게 조절합니다.
  - shear_range: 시계 반대 방향으로 밀림 강도를 조절합니다.
    - 0.5이면, 최대 50%의 비율로 시계 반대 방향으로 기울어지게 됩니다.
  - zoom_range: 임의의 비율만큼 이미지를 확대/축소시킵니다.
    - 0.5이면, 0.5~1.5배의 범위에서 이미지의 크기를 조절합니다.
  - rotation_range: 이미지를 임의로 회전시킵니다.
    - 180이라면, 0~180의 범위에서 임의로 이미지를 회전시킵니다.
  - rescale: 이미지 픽셀값의 크기를 조절합니다.
    - 1/255이면, 각 픽셀값에 해당 값이 곱해집니다.
  - fill_mode: 이미지 변환 시에 새로 생기는 픽셀을 채울 방법을 결정합니다.
    - ["nearest", "constant", "reflect or wrap"]
  - horizontal_flip: True일 경우, 임의로 이미지를 수평 방향으로 뒤집습니다.
  - vertical_flip: True일 경우, 임의로 이미지를 수직 방향으로 뒤집습니다.
  - preprocessing_function: 사용자 정의 전처리 함수 또는 전처리 함수를 적용합니다.

데이터 증식 사용하기

- 이미지 제네레이터를 활용한 결과 확인

```
05  train_datagen = ImageDataGenerator(horizontal_flip = True,
06                                     vertical_flip = True,
07                                     shear_range = 0.5,
08                                     brightness_range = [0.5, 1.5],
09                                     zoom_range = 0.2,
10                                     width_shift_range = 0.1,
11                                     height_shift_range = 0.1,
12                                     rotation_range = 30,
13                                     fill_mode = 'nearest')
```



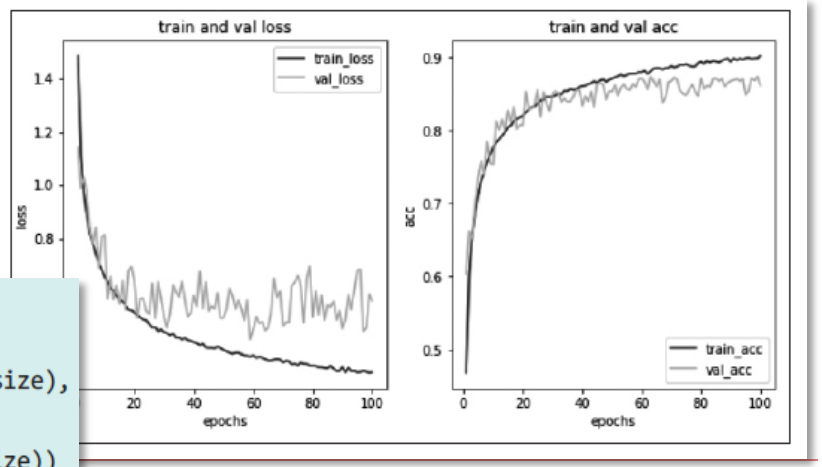
데이터 증식 사용하기

- 이미지 제네레이터 정의
  - 검증 데이터에 활용될 이미지 제네레이터는 증식 옵션을 사용하지 않음

```
03  train_datagen = ImageDataGenerator(horizontal_flip = True,
04                                     zoom_range = 0.2,
05                                     width_shift_range = 0.1,
06                                     height_shift_range = 0.1,
07                                     rotation_range = 30,
08                                     fill_mode = 'nearest')
09
10  val_datagen = ImageDataGenerator()
```

- 모델 학습
  - steps_per_epoch 인자
    - 1 에폭에 배치만큼의 크기를 몇번 전달할 것인지
    - '학습 데이터 개수/배치 크기 ' 를 전달
    - 제공되는 코드는 1094번
    - 해당 횟수보다 적은 값을 전달할 경우, 전체 데이터를 사용하지 않으므로 주의

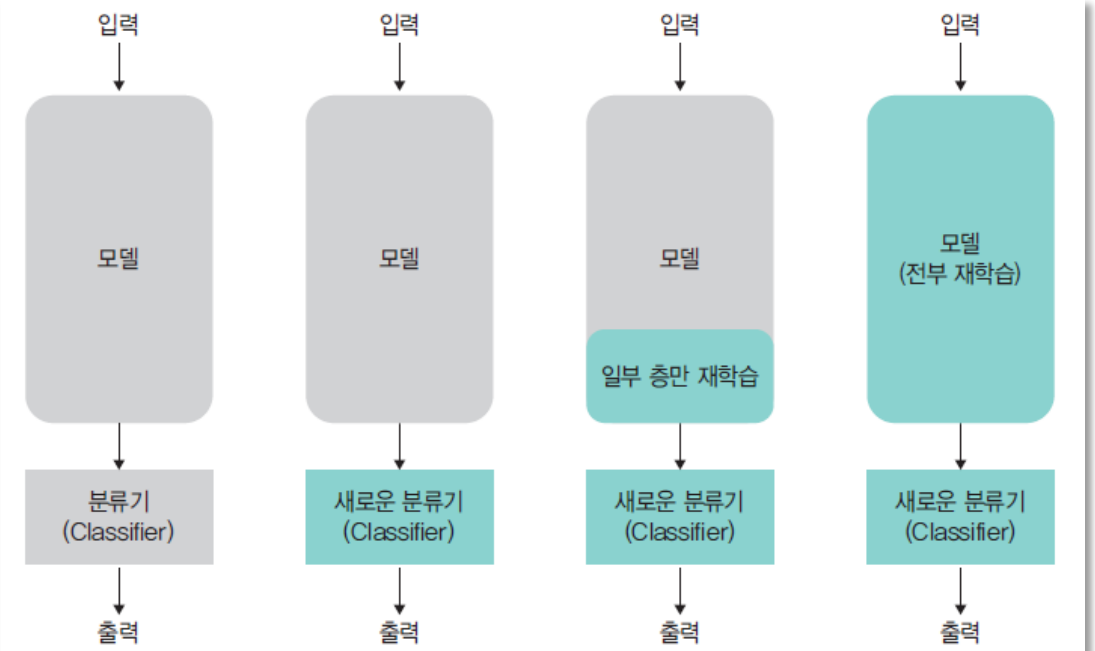
```
27  history = model.fit(train_generator,
28                      epochs = 100,
29                      steps_per_epoch = get_step(len(x_train), batch_size),
30                      validation_data = val_generator,
31                      validation_steps = get_step(len(x_val), batch_size))
```



전이 학습

- 해결하려는 문제에서 빠른 속도로 일정 수준의 베이스라인(baseline) 성능을 얻고 싶을 때, 가장 쉽고 빠른 방법 → **전이 학습(Transfer Learning)**
  - 사전 학습(Pre-trained)된 네트워크의 가중치를 사용하는 것

- 전이 학습의 대표적인 세 가지 방법
  1. 모델을 변형하지 않고 사용
  2. 모델 분류기를 재학습(가장 대표적)
  3. 모델 일부를 동결 해제하여 재학습



[그림 5-17] 전이 학습을 사용하는 경우

전이 학습

- 전이 학습은 주로 ImageNet 데이터를 학습시킨 가중치를 사용하거나 보유한 데이터셋을 활용하여 재학습을 진행하는데, 이를 **미세조정(Fine-Tuning)**이라고 함

- 케라스는 이를 위한 다양한 모델을 제공하고 있음
  - https://keras.io/api/applications/
  - 어떤 구조의 모델이 최고 성능을 보여줄지 모르기 때문에, 다양한 모델을 사용해보는 것도 중요

- 제공하고 있는 모델을 활용하여 기준을 마련하고, 결과 확인 후 사용할 모델 크기를 결정

Available models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

전이 학습

- ImageNet 데이터로 사전 학습된 VGG16 모델 사용해보기
  - VGG16 모델 정의

```
05  from tensorflow.keras.applications import VGG16
06
07  # ImageNet 데이터를 학습한 모델을 불러옵니다.
08  vgg16 = VGG16(weights = 'imagenet',
                  input_shape = (32, 32, 3), include_top = False)
```

  - 모델에 추가합니다.

```
10  model = Sequential()
11  model.add(vgg16)
```

  - 분류기는 직접 정의합니다.

```
12  # 분류기를 직접 정의합니다.
13  model.add(Flatten())
14  model.add(Dense(256))
15  model.add(BatchNormalization())
16  model.add(Activation('relu'))
17  model.add(Dense(10, activation = 'softmax'))
```

전이 학습

- VGG16 외에도 다양한 모델을 정의하여 사용할 수 있습니다.

예시: 케라스가 제공하는 다양한 모델

```
01 from tensorflow.keras.applications import *
02
03 mobilenet = MobileNet(weights = None, input_shape = None,
                        include_top = True)
04 resnet50 = ResNet50(weights = None, input_shape = None, include_top = True)
05 xception = Xception(weights = None, input_shape = None, include_top = True)
```

- 사전 학습 모델 사용 시, 주로 세 가지 인자를 사용
  - **weights:** ImageNet 데이터를 학습시킨 가중치의 사용 여부를 결정. 기본값은 None이며, 가중치를 사용하고 싶다면 'imagenet'을 전달
  - **input_shape:** 입력 데이터의 형태를 전달
  - **include_top:** 모델의 분류기층을 포함해서 구성할지를 결정. False를 전달할 경우, 데이터셋에 적합한 분류기를 직접 정의해야 함

전이 학습(실습)

- 사용하는 데이터셋 특성이 고유하다면, 모델의 일부분만 동결을 해제시키고 재학습시킬 필요가 있음

[함께 해봐요] **모델 동결 해제하기**

```
01  # 끝의 네 개 층만 동결을 해제합니다.
02  for layer in vgg16.layers[:-4]:
03      layer.trainable = False
```

  - 위 코드를 통해 모델 끝의 4개의 층은 학습을 통해 가중치가 재조정되며, 그 외의 층은 학습되지 않음

- 모델 끝에 존재하는 층을 동결 해제하는 이유?
  - **모델은 상위층(출력과 가까운 층)에서 데이터의 "구체적 특성"을 학습하고, 하위층(입력과 가까운 층)에서 "단순한(일반적) 특징"을 학습하기 때문**

    ex) 얼굴을 학습한다면, 모델 상위층은 입 모양, 눈 모양, 코 모양과 같이 구체적 특징을 학습하고, 모델 하위층은 얼굴 모양을 학습할 것

- 제공되는 코드를 통해 전이 학습을 실습해보세요.

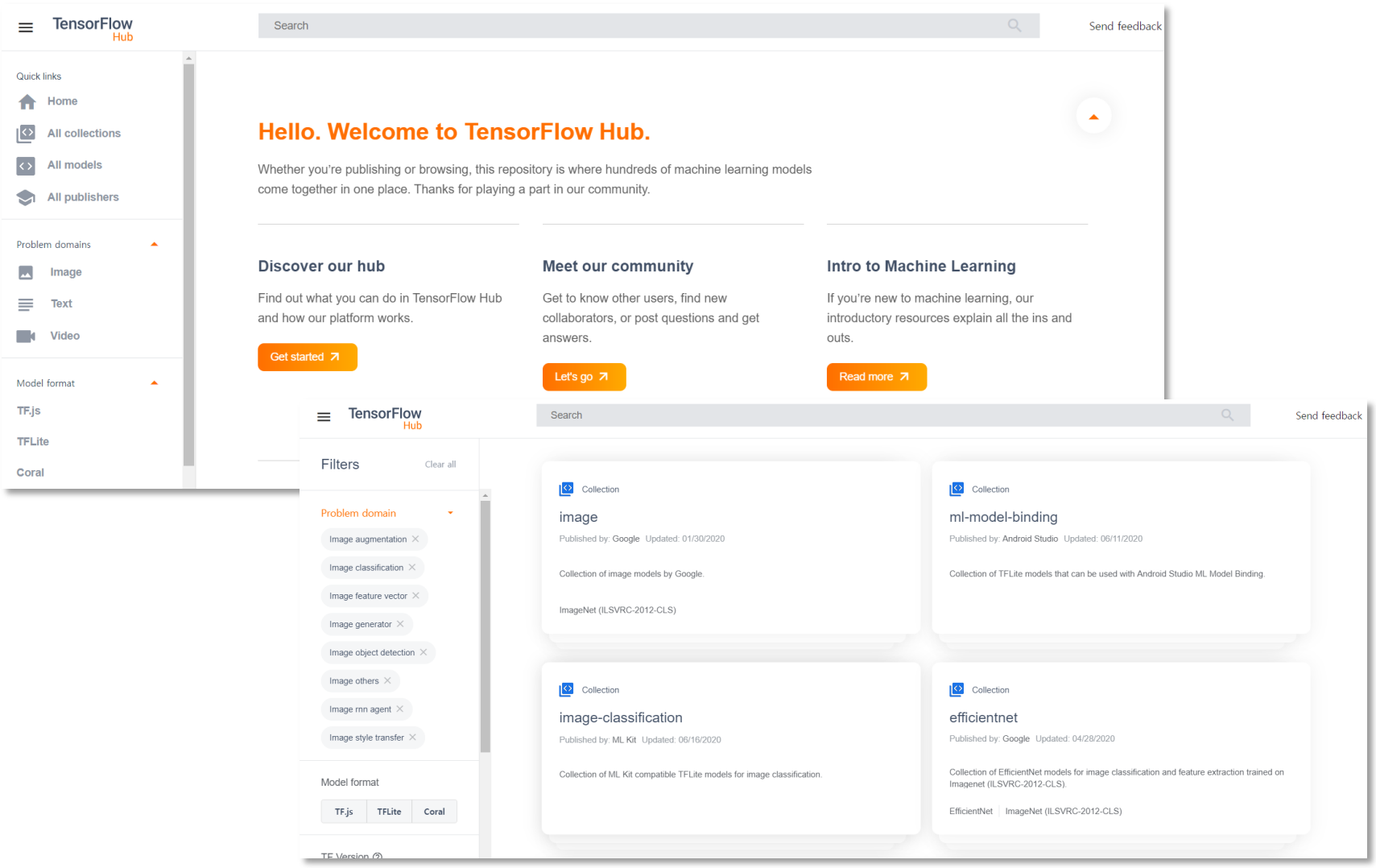
TensorFlow Hub

- 사전 학습된 가중치는 다음 조건을 만족해야 함
  - 우리가 해결하려는 문제와 특성이 동일해야 함
  - 양과 질이 보장된 데이터셋을 사용했어야 함
  - 사용된 데이터셋이 문제를 일반화할 수 있어야 함

- 앞서 사용한 VGG16은 ImageNet 데이터셋을 활용하여 만들어진 가중치를 사용하였음
  - 우리가 직접 만들 수 있을까? → 실험 환경, 시간적으로 매우 제한적임
  - 깃허브 저장소, 기타 장소에서 제공하는 가중치 → 적용이 어렵고, 안전하지 않으며, 적합하지 않을 수 있음

- 이 같은 문제를 겪고 있는 우리를 위해 구글에서 **TensorFlow Hub**를 제공
  - 목적: 우리가 원하는 사전 학습된 모델을 쉽게 찾을 수 있도록 함
  - 안전하고, 검증된 모델을 제공
  - 모바일 앱, TensorFlow.js 모델도 제공

TensorFlow Hub



요약 정리

1. 컨볼루션 신경망은 이미지 데이터에 특화되어 있지만, 음성 인식이나 비디오, 6장에서 다뤄볼 텍스트 데이터에도 사용됩니다.
2. **완전연결층은 공간 정보를 손실**하는 반면, **컨볼루션층은 공간정보를 유지**합니다.
3. 컨볼루션층은 컨볼루션 필터를 통해 이미지의 특징을 인식할 수 있게 됩니다. 또한, **컨볼루션 필터가 가지는 파라미터**는 이미지 필터와 다르게 직접 정의해주지 않고, **학습을 통해 조정**됩니다.
4. 컨볼루션층에서는 주요한 인자로서 **컨볼루션 필터 개수, 스트라이드 크기, 패딩 여부**를 사용하고, 풀링층에서는 주요한 인자로서 **커널 크기, 스트라이드 크기**를 사용합니다.
5. 컨볼루션층은 1x1 크기를 사용하여 최대한 공간정보를 손실하지 않도록 하며, 다운샘플링이 필요할 경우 최대 풀링층을 사용합니다.
6. model.summary(), plot_model() 함수는 모델 구조를 확인하기에 유용합니다.
7. **규제화 함수, 드롭아웃, 배치 정규화**는 과대적합을 방지할 뿐, 100% 해결해주지 않습니다.

요약 정리

8. **데이터 증식 방법**은 다양한 데이터를 모델에 입력해 **더욱 견고한 모델**을 얻을 수 있도록 도와줍니다. 케라스에는 이를 위한 이미지 제네레이터가 준비되어 있습니다.
9. **전이 학습**은 **사전 학습된 가중치를 사용**하여 더욱 빠르게 향상된 성능을 얻을 수 있도록 도와줍니다. 케라스는 수많은 이미지 데이터로 구성된 ImageNet 데이터를 학습한 다양한 모델을 제공하고 있습니다.
10. **모델 상위층은 데이터의 구체적 특징**을 학습하며, **모델 하위층은 단순한 특징을 학습**합니다.
11. **텐서플로우 허브**는 우리가 원하는 모델을 쉽게 찾을 수 있도록 도와줍니다.