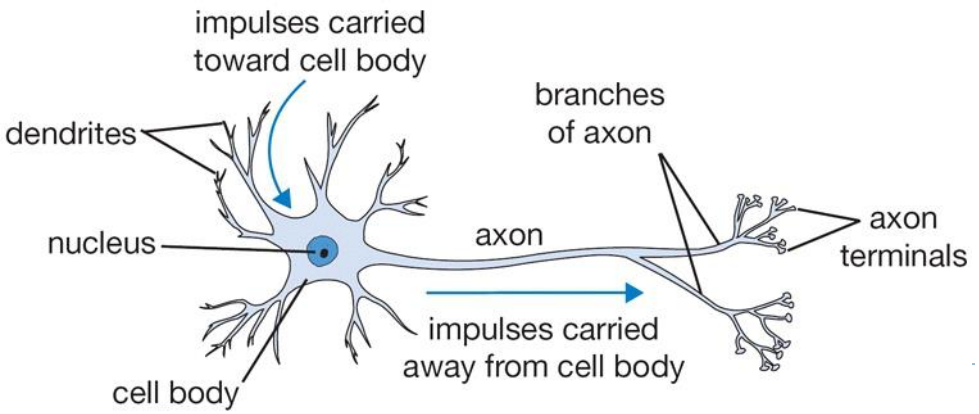
4. 레이어 개념잡기

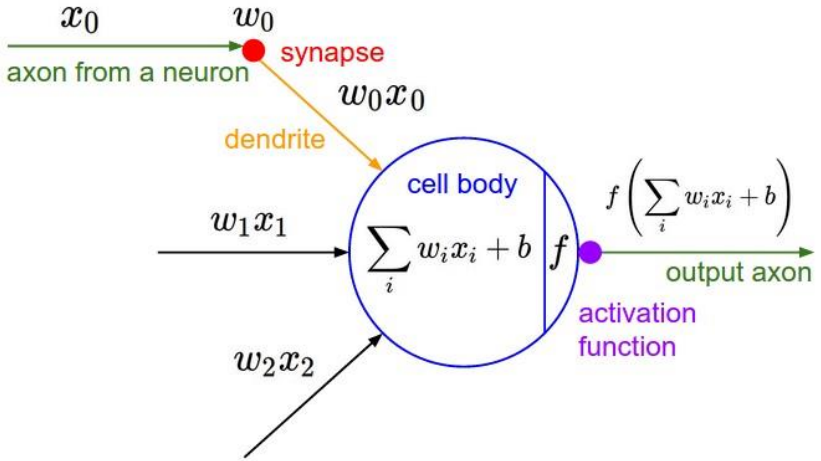
다층 퍼셉트론 레이어 이야기

- 이번에는 케라스에서 사용되는 레이어(layer, 층) 개념에 대해서 알아봅니다.
- 케라스의 핵심 데이터 구조는 모델이고, 이 모델을 구성하는 것이 레이어입니다.
- 간단히 뉴런에 대해서 알아본 다음, 주요 레이어에 대해 기본 개념, 역할 등에 대해서 살펴보고, 레이어를 어떻게 쌓아서 모델을 만들 수 있는 지 알아봅니다.
- 기본적인 레이어 개념을 익히면 레고 쌓는 것 처럼 쉽게 구성할 수 있는데, 실제 레고로도 쌓아보겠습니다.
- 본 강좌에서는 다층 퍼셉트론 모델에서 사용되는 Dense 레이어에 대해서만 알아보겠습니다.

다층 퍼셉트론 레이어 이야기

- 인간의 신경계를 흉내낸 뉴런 이야기
  - 신경망에서 사용되는 뉴런은 인간의 신경계를 흉내낸 것입니다.
  - 아래 왼쪽 그림이 인간의 뉴런이고, 오른쪽 그림이 이를 모델링 한 것입니다.
    - axon (축삭돌기) : 팔처럼 몸체에서 뻗어나와 다른 뉴런의 수상돌기와 연결됩니다.
    - dendrite (수상돌기) : 다른 뉴런의 축삭 돌기와 연결되며, 몸체에 나뭇가지 형태로 붙어 있습니다.
    - synapse (시냅스) : 축사돌기와 수상돌기가 연결된 지점입니다. 여기서 한 뉴런이 다른 뉴런으로 신호가 전달됩니다.



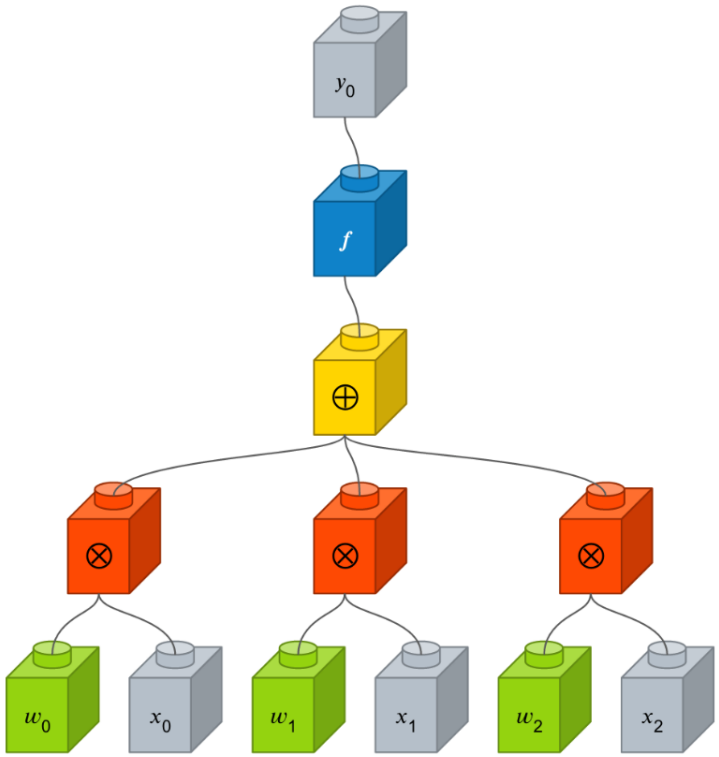


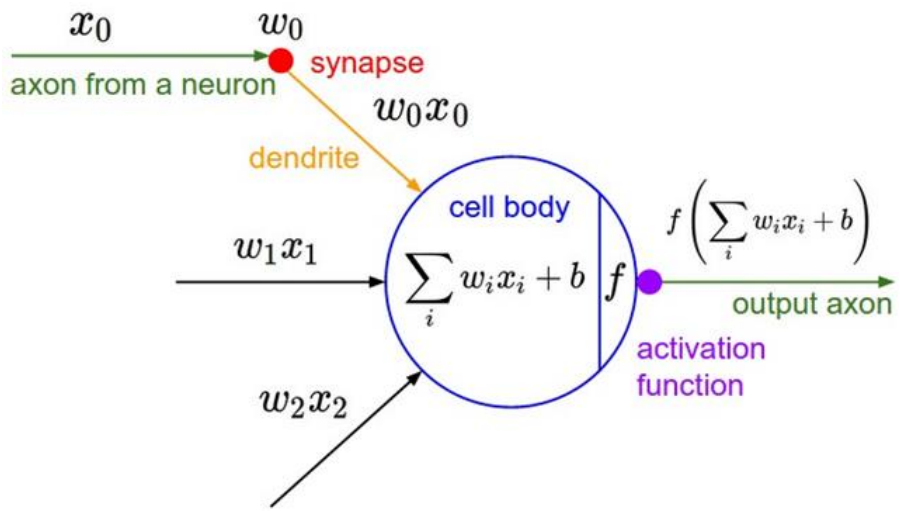
다층 퍼셉트론 레이어 이야기

- 인간의 신경계를 모사한 뉴런이야기
  - 하나의 뉴런은 여러 다른 뉴런의 축삭돌기와 연결되어 있는데, 연결된 시냅스의 강도가 연결된 뉴런들의 영향력이 결정됩니다.
  - 이러한 영향력의 합이 어떤 값을 초과하면 신호가 발생하여 축삭돌기를 통해서 다른 뉴런에게 신호가 전달되는 식입니다.
  - 오른쪽 그림의 모델링과는 다음과 같이 매칭됩니다.
    - x0, x1, x2 : 입력되는 뉴런의 축삭돌기로부터 전달되는 신호의 양
    - w0, w1, w2 : 시냅스의 강도, 즉 입력되는 뉴런의 영향력을 나타냅니다.
    - w0*x0 + w1*x1 + w2*x2 : 입력되는 신호의 양과 해당 신호의 시냅스 강도가 곱해진 값의 합계
    - f : 최종 합계가 다른 뉴런에게 전달되는 신호의 양을 결정짓는 규칙, 이를 활성화 함수라고 부릅니다

다층 퍼셉트론 레이어 이야기

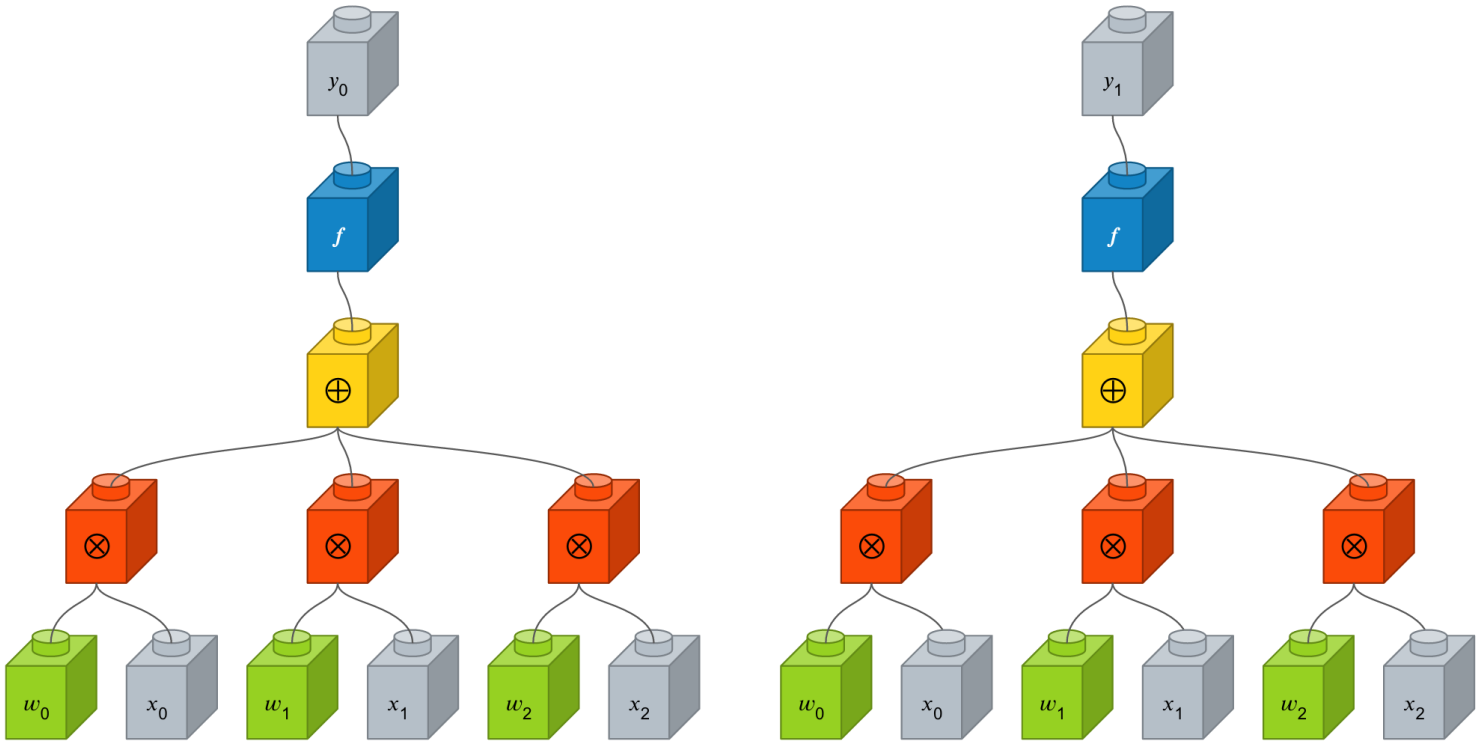
- 인간의 신경계를 모사한 뉴런이야기
  - 세 개의 신호를 받아 하나의 신호를 전달하는 뉴런을 레고로 표현하면 다음과 같다.
  - 녹색 블럭은 시냅스의 강도, 노란색과 빨간색 블럭은 연산자, 파란색 블럭은 활성화 함수를 나타냅니다.





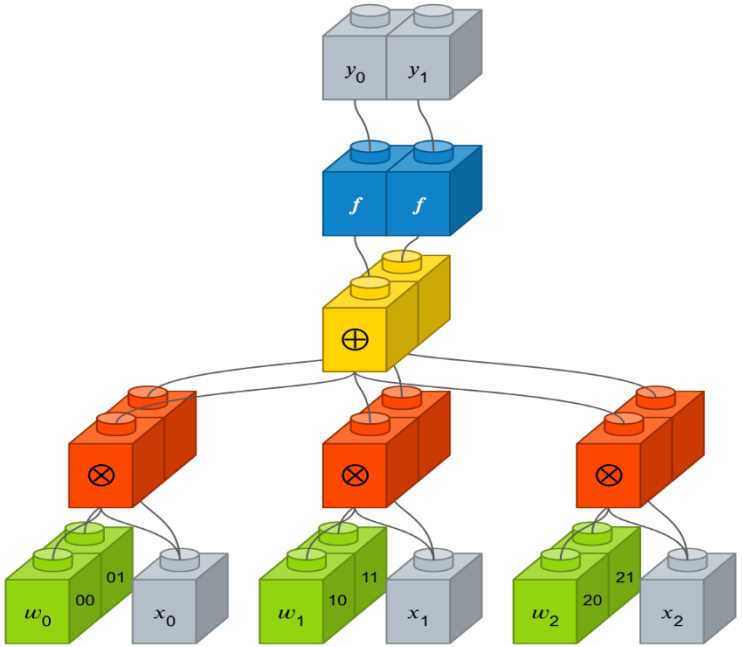
다층 퍼셉트론 레이어 이야기

- 인간의 신경계를 모사한 뉴런이야기
  - 만약 세 개의 신호가 서로 다른 뉴런 두 개에 전달된다고 한다면, 각 뉴런은 하나의 신호가 출력되므로, 총 두 개의 신호가 출력됩니다.
  - 이를 레고로 표현하면 다음과 같습니다.



다층 퍼셉트론 레이어 이야기

- 인간의 신경계를 모사한 뉴런 이야기
  - 위와 같은 표현이지만 이를 겹쳐 표현하면 아래와 같습니다.
  - 다시 말해 세 개의 신호를 받는 뉴런 두 개를 표현 한 것입니다.
  - 여기서 유심히 봐야할 점은 시냅스의 강도 즉 녹색 블럭의 개수입니다.
  - 세 개의 신호가 뉴런 두 개에 연결되므로 총 연결 경우의 수 (3*2=6)인 6개가 됩니다.



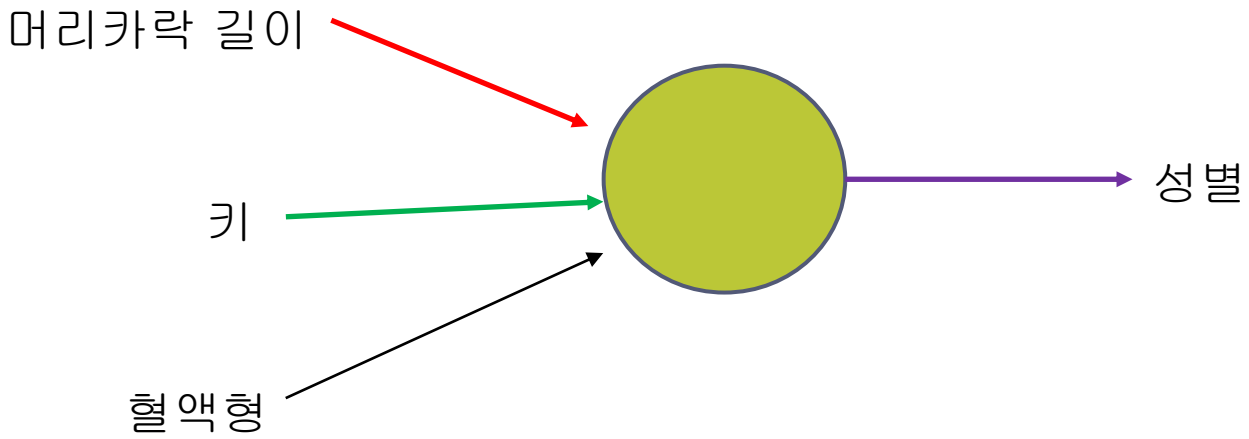
다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - Dense 레이어는 입력과 출력을 모두 연결해줍니다.
  - 예를 들어 입력 뉴런이 4개, 출력 뉴런이 8개 있다면 총 연결선은 32개(4*8=32) 입니다.
  - 각 연결선에는 가중치(weight)를 포함하고 있는데, 이 가중치가 나타내는 의미는 연결강도라고 보시면 됩니다.
  - 현재 연결선이 32개이므로 가중치도 32개입니다.

가중치가 높을수록 해당 입력 뉴런이 출력 뉴런에 미치는 영향이 크고, 낮을수록 미치는 영향이 적다.

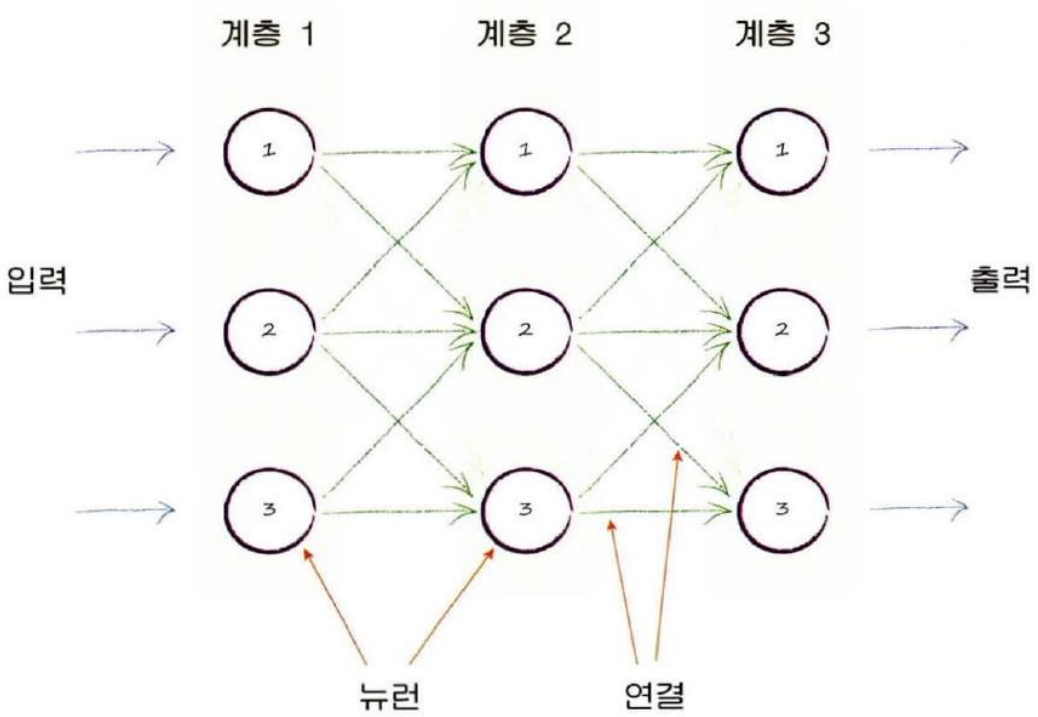
다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 예를 들어 성별을 판단하는 문제에 있어서, 출력 뉴런의 값이 성별을 의미하고, 입력 뉴런에 머리카락 길이, 키, 혈액형 등이 있다고 가정했을 때, 머리카락길이의 가중치가 가장 높고, 키의 가중치가 중간이고, 혈액형의 가중치가 가장 낮을 겁니다.



다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 딥러닝 학습과정에서 이러한 가중치들이 조정됩니다.
  - 이렇게 입력 뉴런과 출력 뉴런을 모두 연결한다고 해서 전결합층이라고 불리고, 케라스에서는 Dense라는 클래스로 구현이 되어 있습니다.



다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 아래는 Dense 클래스 사용 예제입니다.

```
Dense(8, input_dim=4, init='uniform', activation='relu'))
```

  - 주요 인자는 다음과 같습니다.
    - 첫번째 인자 : 출력 뉴런의 수를 설정합니다.
    - input_dim : 입력 뉴런의 수를 설정합니다.
    - init : 가중치 초기화 방법을 설정합니다.
      - ‘uniform’ : 균일 분포
      - 'normal’ : 가우시안 분포
    - activation : 활성화 함수를 설정합니다.
      - ‘linear’ : 디폴트 값, 입력뉴런과 가중치로 계산된 결과값이 그대로 출력으로 나옵니다.
      - ‘relu’ : rectifier 함수, 은닉층에 주로 쓰입니다.
      - ‘sigmoid’ : 시그모이드 함수, 이진 분류 문제에서 출력층에 주로 쓰입니다.
      - ‘softmax’ : 소프트맥스 함수, 다중 클래스 분류 문제에서 출력층에 주로 쓰입니다.

다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - Dense 레이어는 입력 뉴런 수에 상관없이 출력 뉴런 수를 자유롭게 설정할 수 있기 때문에 출력층으로 많이 사용됩니다.
  - 이진 분류문제에서는 0과 1을 나타내는 출력 뉴런이 하나만 있으면 되기 때문에 아래 코드처럼 출력 뉴런이 1개이고, 입력 뉴런과 가중치를 계산한 값을 0에서 1사이로 표현할 수 있는 활성화 함수인 sigmoid을 사용합니다.

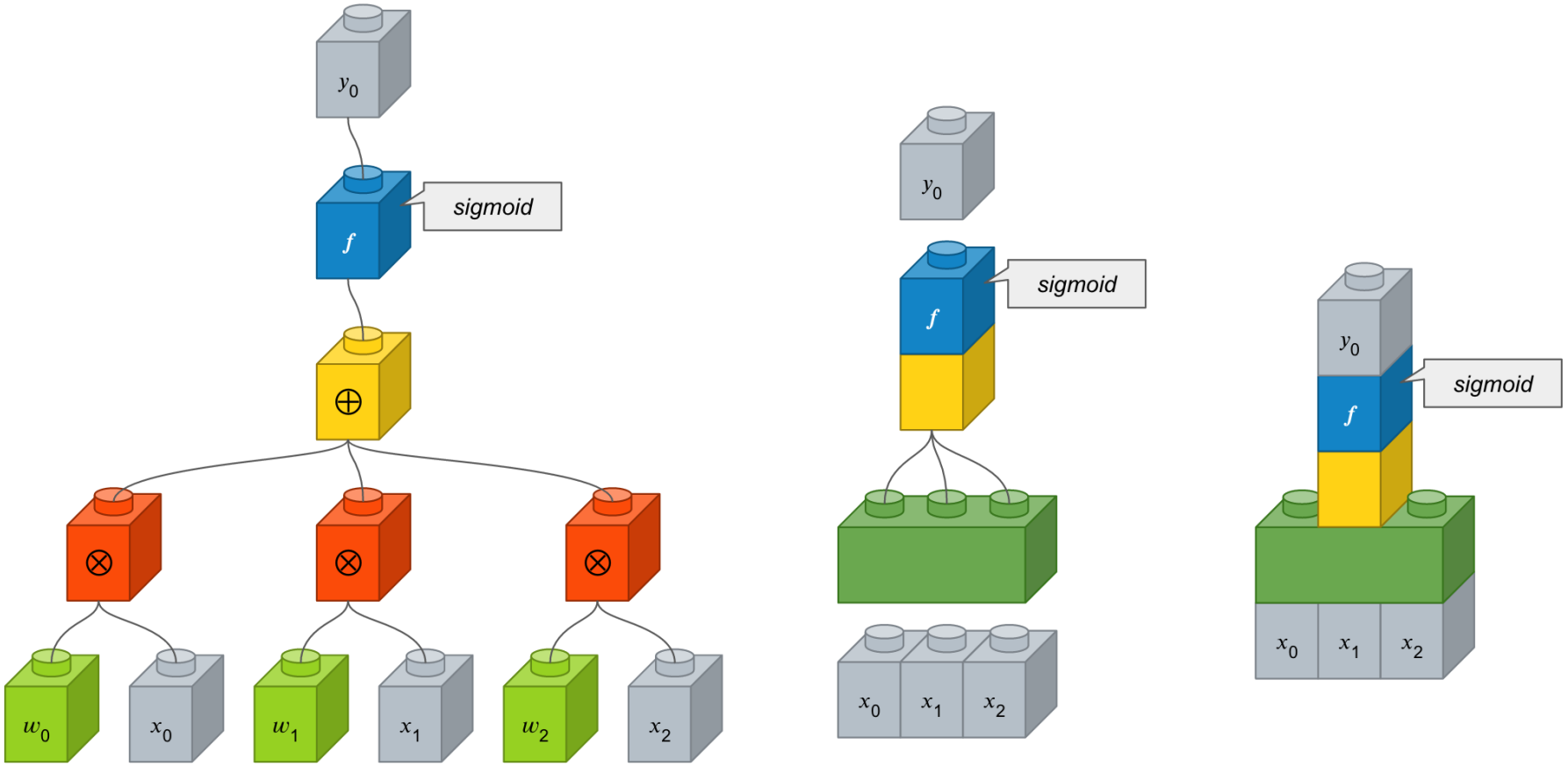
```
Dense(1, input_dim=3, activation='sigmoid'))
```

다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 이를 레고로 표시하면 다음과 같습니다.
  - 왼쪽 그림은 앞서 설명한 뉴런 상세 구조를 도식화 한 것이고, 오른쪽 그림은 이를 간단하게 도식화한 것입니다.
  - 왼쪽 그림에서 시냅스 강도가 녹색 블럭으로 표시되어 있다면, 중간 그림에서는 시냅스 강도가 연결선으로 표시되어 있고, 오른쪽 그림에서는 생략되어 있습니다.
  - 생략되어 있더라도 입력 신호의 수와 출력 신호의 수만 알면 곱셈으로 쉽게 유추할 수 있습니다.

다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 이를 레고로 표시하면 다음과 같습니다.



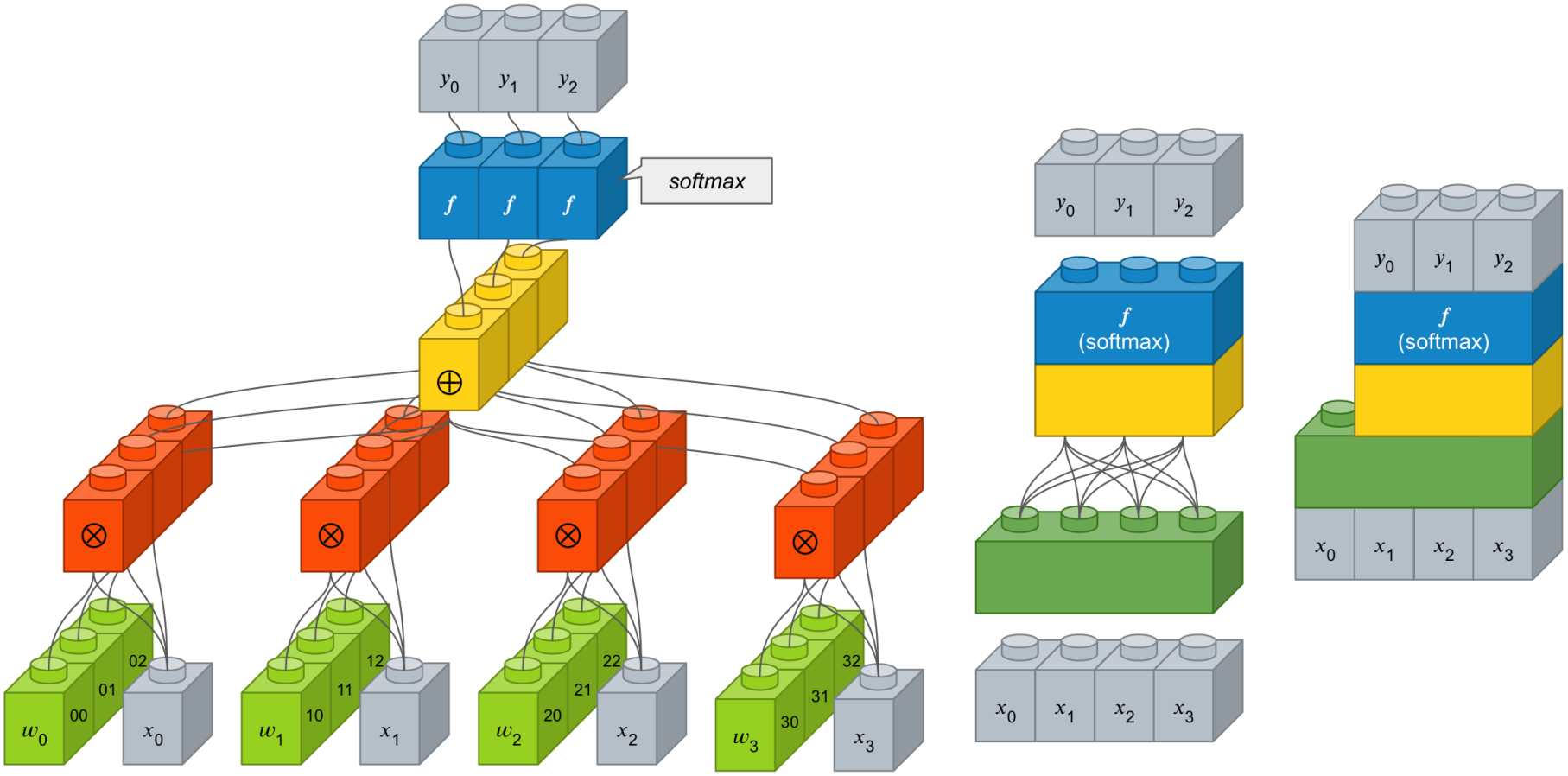
다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 다중클래스 분류문제에서는 클래스 수만큼 출력 뉴런이 필요합니다.
  - 만약 세가지 종류로 분류한다면, 아래 코드처럼 출력 뉴런이 3개이고, 입력 뉴런과 가중치를 계산한 값을 각 클래스의 확률 개념으로 표현할 수 있는 활성화 함수인 softmax를 사용합니다.

```
Dense(3, input_dim=4, activation='softmax'))
```

다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 이를 레고로 표시하면 다음과 같습니다. 입력 신호가 4개이고 출력 신호가 3개이므로 시냅스 강도의 개수는 12개입니다.



다층 퍼셉트론 레이어 이야기

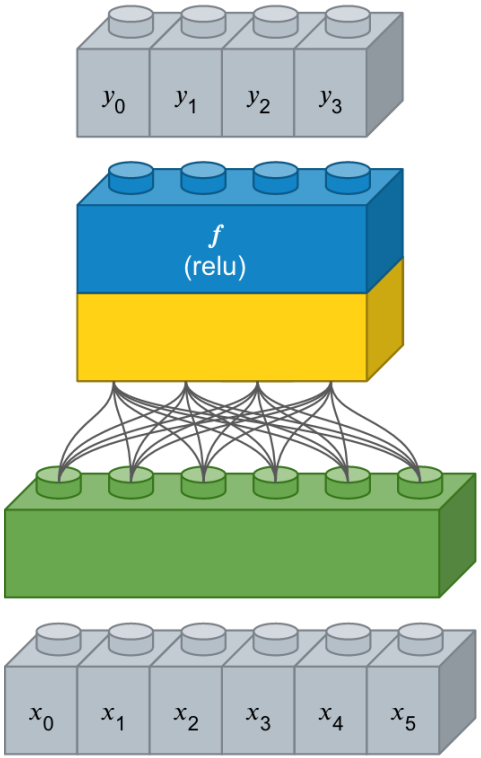
- 입출력을 모두 연결해주는 Dense 레이어
  - Dense 레이어는 보통 출력층 이전의 은닉층으로도 많이 쓰이고, 영상이 아닌 수치자료 입력 시에는 입력층으로도 많이 쓰입니다.
  - 이 때 활성화 함수로 'relu'가 주로 사용됩니다.
  - 'relu'는 학습과정에서 역전파 시에 좋은 성능이 나는 것으로 알려져 있습니다.

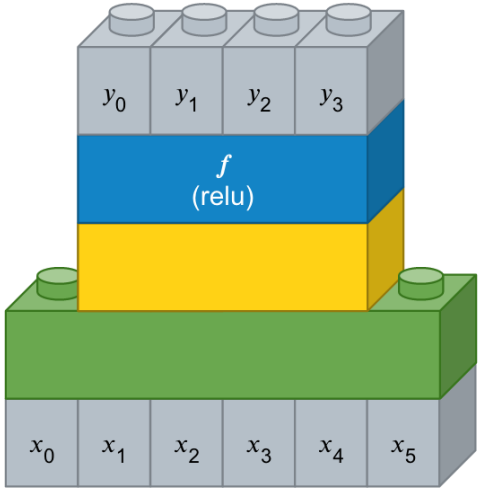
```
Dense(4, input_dim=6, activation='relu'))
```

다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 이를 레고로 표시하면 다음과 같습니다.

```
Dense(4, input_dim=6, activation='relu'))
```





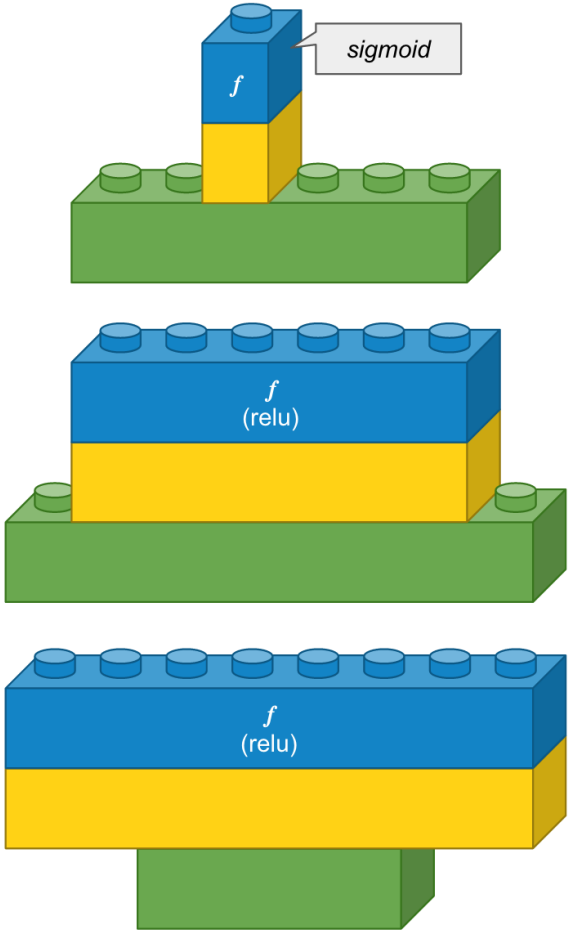
다층 퍼셉트론 레이어 이야기

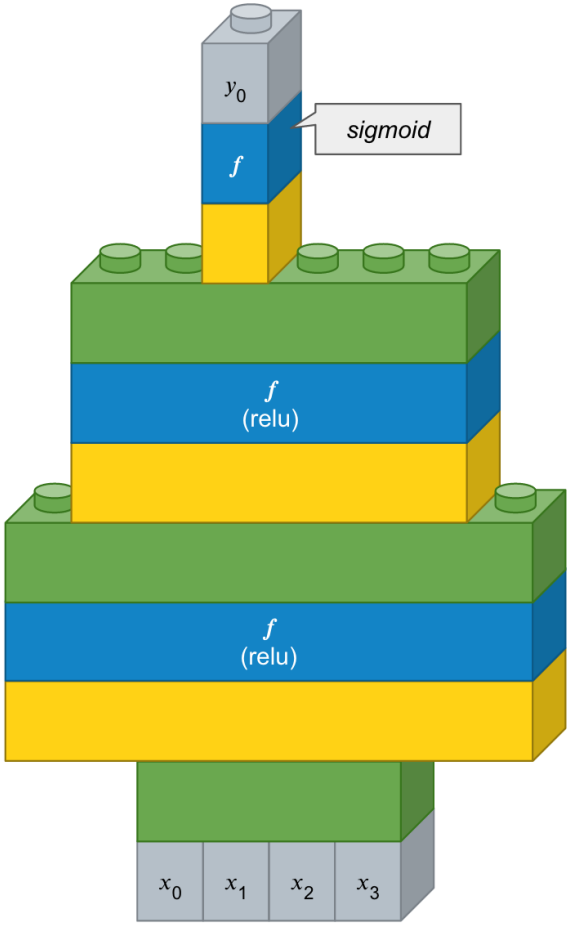
- 입출력을 모두 연결해주는 Dense 레이어
  - 또한 입력층이 아닐 때에는 이전층의 출력 뉴런 수를 알 수 있기 때문에 input_dim을 지정하지 않아도 됩니다.
  - 아래 코드를 보면, 입력층에만 input_dim을 정의하였고, 이후 층에서는 input_dim을 지정하지 않았습니다.

```
model.add(Dense(8, input_dim=4, init='uniform', activation='relu'))
model.add(Dense(6, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))
```

다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 이를 레고로 표시하면 다음과 같습니다.





다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 이를 레고로 표시하면 다음과 같습니다.
  - 왼쪽 그림은 Dense 레이어 세 개를 도식화 한 것이고, 오른쪽 그림은 입력과 출력의 수에 맞게 연결하여 입력 신호가 인가되었을 때, 출력 신호가 나오는 것 까지의 구성을 표시한 것입니다.
  - 이제 레고 블럭만 봐도 입력값이 4이고 출력값이 0에서 1까지 범위를 가지는 값이 나올 수 있도록 설계된 구조임을 알 수 있습니다.
  - 활성화 함수가 sigmoid이기 때문에 이진 분류에 적합합니다.

다층 퍼셉트론 레이어 이야기

- 입출력을 모두 연결해주는 Dense 레이어
  - 쌓았던 레고를 실제 케라스로 구현해봅니다.
  - 4개의 입력 값을 받아 이진 분류하는 문제를 풀 수 있는 모델입니다.

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

model.add(Dense(8, input_dim=4, init='uniform', activation='relu'))
model.add(Dense(6, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))
```

다층 퍼셉트론 레이어 이야기

- 결론
  - 본 강좌를 통해 신경망의 기본인 뉴런에 대해서 알아보고, 이를 다양한 방식으로 도식화 해봤습니다.
  - 그리고 다층 퍼셉트론 모델에서 가장 기본이 되는 전결합층인 Dense 레이어와 Dense 레이어를 쌓는 법에 대해서 알아봤습니다.
  - 다음 강좌에는 레이어를 조합하여 실제로 다층 퍼셉트론 모델을 만들어봅니다.

다층 퍼셉트론 모델 만들어보기

- 문제 정의하기
  - 다층 퍼셉트론 모델은 가장 기본적인 모델이라 대부분 문제에 적용할 수 있습니다.
  - 본 예제에서는 비교적 쉬운 이진 분류 문제를 적용해보고자 합니다.
  - 이진 분류 예제에 적합한 데이터셋은 8개 변수와 당뇨병 발병 유무가 기록된 '피마족 인디언 당뇨병 발병 데이터셋'이 있습니다.
  - 이 데이터셋을 이용하여 8개 변수를 독립변수로 보고 당뇨병 발병 유무를 예측하는 이진 분류 문제로 정의해보겠습니다.
  - 데이터셋은 아래 링크에서 다운로드 받으실 수 있습니다.
    - https://github.com/jjin300/deep_learning

다층 퍼셉트론 모델 만들어보기

- 문제 정의하기
  - '피마족 인디언 당뇨병 발병 데이터셋'을 선정한 이유는 다음과 같습니다.
    - 인스턴스 수와 속성 수가 예제로 사용하기에 적당합니다.
    - 모든 특징이 정수 혹은 실수로 되어 있어서 별도의 전처리 과정이 필요 없습니다.
  - 데이터셋을 준비하기에 앞서, 매번 실행 시마다 결과가 달라지지 않도록 랜덤 시드를 명시적으로 지정합니다.
  - 이것을 하지 않으면 매번 실행 시 마다 동일 모델인데도 불구하고 다른 결과가 나오기 때문에, 연구개발 단계에서 파라미터 조정이나 데이터셋에 따른 결과 차이를 보려면 랜덤 시드를 지정해주는 것이 좋습니다.

다층 퍼셉트론 모델 만들어보기

- 구글 드라이브와 연동하기

```
from google.colab import drive
drive.mount('/content/gdrive')
```

- 문제 정의하기

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense

# 랜덤시드 고정시키기
np.random.seed(5)
```

다층 퍼셉트론 모델 만들어보기

- 데이터 준비하기
  - 위 링크에서 'pima-indians-diabetes.names'을 열어보면 데이터셋에 대한 설명이 포함되어 있습니다.
  - 먼저 몇가지 주요 항목을 살펴보겠습니다.
    - 인스턴스 수 : 768개
    - 속성 수 : 8가지
    - 클래스 수 : 2가지

다층 퍼셉트론 모델 만들어보기

- 데이터 준비하기
  - 8가지 속성(1번~8번)과 결과(9번)의 상세 내용은 다음과 같습니다.
    - 1. 임신 횟수
    - 2. 경구 포도당 내성 검사에서 2시간 동안의 혈장 포도당 농도
    - 3. 이완기 혈압 (mm Hg)
    - 4. 삼두근 피부 두겹 두께 (mm)
    - 5. 2 시간 혈청 인슐린 (mu U/ml)
    - 6. 체질량 지수
    - 7. 당뇨 직계 가족력
    - 8. 나이 (세)
    - 9. 5년 이내 당뇨병이 발병 여부

다층 퍼셉트론 모델 만들어보기

- 데이터 준비하기
  - 좀 더 살펴보면, 양성인 경우가 268개(34.9%), 음성인 경우가 500개(65.1%)입니다.
  - 즉 모델이 모두 음성이라고 판별을 한다하더라도 65.1%의 기본 정확도(baseline accuracy)를 달성할 수 있습니다.
  - 즉 우리의 모델이 65.1%보다 낮으면 모두 음성이라고 판별하는 것보다 낮은 정확도를 가진다고 생각하시면 됩니다.
  - 지금까지 개발된 알고리즘의 최대 정확도는 10-fold 교차검증(cross validation) 했을 때 77.7%이라고 웹사이트에는 표기되어 있습니다.

```
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
```

다층 퍼셉트론 모델 만들어보기

- 데이터 준비하기
  - 속성별 간단한 통계 정보는 다음과 같습니다.

No.	속성	평균	표준편차
1	임신횟수	3.8	3.4
2	포도당 내성	120.9	32.0
3	이완기 혈압	69.1	19.3
4	심두근 피부 두겹 두께	20.5	16.0
5	혈청 인슐린	79.8	115.2
6	체질량 지수	32.0	7.9
7	당뇨 직계 가족력	0.5	0.3
8	나이	33.2	11.8

  - numpy 패키지에서 제공하는 loadtxt() 함수를 통해 데이터를 불러옵니다.

```
dataset = np.loadtxt("/content/gdrive/My Drive/pima-indians-diabetes.csv", delimiter=",")
```

다층 퍼셉트론 모델 만들어보기

- 데이터셋 생성하기
  - csv 형식의 파일은 numpy 패키지에서 제공하는 loadtxt() 함수로 직접 불러올 수 있습니다.
  - 데이터셋에는 속성값과 판정결과가 모두 포함되어 있기 때문에 입력(속성값 8개)와 출력(판정결과 1개) 변수로 분리합니다.

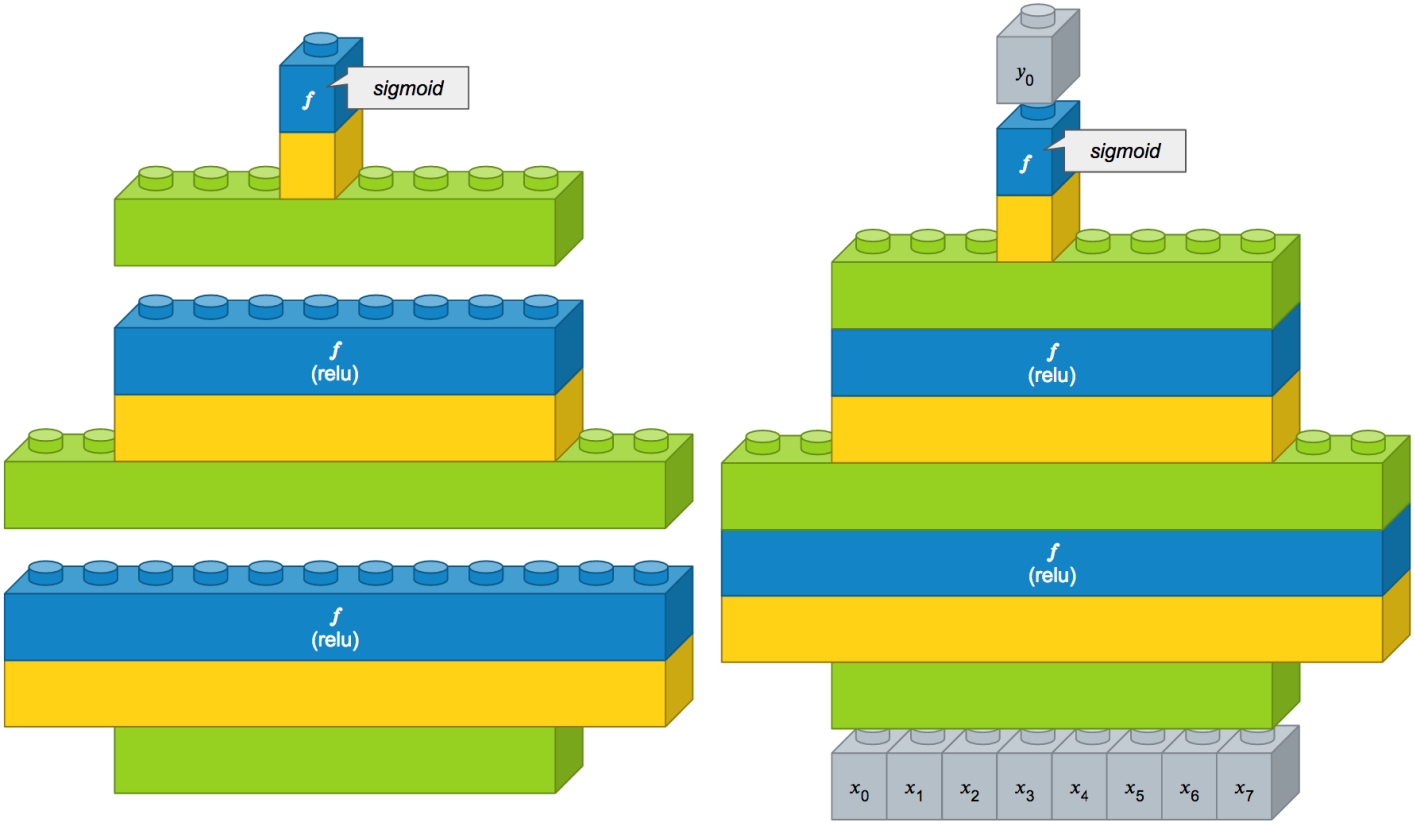
```
x_train = dataset[:700,0:8]
y_train = dataset[:700,8]
x_test = dataset[700:,0:8]
y_test = dataset[700:,8]
```

다층 퍼셉트론 모델 만들어보기

- 모델 구성하기
  - 앞 강좌에서 배운 Dense 레이어만을 사용하여 다층 퍼셉트론 모델을 구성할 수 있습니다.
  - 속성이 8개이기 때문에 입력 뉴런을 8개이고, 이진 분류이기 때문에 0~1사이의 값을 나타내는 출력 뉴런이 1개입니다.
    - 첫번째 Dense 레이어는 은닉층(hidden layer)으로 8개 뉴런을 입력받아 12개 뉴런을 출력합니다.
    - 두번째 Dense 레이어는 은닉층으로 12개 뉴런을 입력받아 8개 뉴런을 출력합니다.
    - 마지막 Dense 레이어는 출력 레이어로 8개 뉴런을 입력받아 1개 뉴런을 출력합니다.

다층 퍼셉트론 모델 만들어보기

- 모델 구성하기
  - 이 구성을 블록으로 표시해봤습니다.
  - 총 세 개의 Dense 레이어 블록으로 모델을 구성한 다음, 8개의 속성 값을 입력하면 1개의 출력값을 얻을 수 있는 구성입니다.



다층 퍼셉트론 모델 만들어보기

- 모델 구성하기
  - 이 블록을 코드로 만들면 다음과 같습니다.
  - 총 세 개의 Dense 레이어 블록으로 모델을 구성한 다음, 8개의 속성 값을 입력하면 1개의 출력값을 얻을 수 있는 구성입니다.

```
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

다층 퍼셉트론 모델 만들어보기

- 모델 구성하기
  - 은닉 레이어의 활성화 함수는 모두 'relu'를 사용하였고, 출력 레이어만 0과 1사이로 값이 출력될 수 있도록 활성화 함수를 'sigmoid'로 사용하였습니다.
  - 0과 1사이의 실수값이 나오기 때문에 양성 클래스의 확률로 쉽게 매칭할 수 있습니다.

다층 퍼셉트론 모델 만들어보기

- 모델 학습과정 설정하기
  - 모델을 정의했다면 모델을 손실함수와 최적화 알고리즘으로 엮어봅니다.
    - loss : 현재 가중치 세트를 평가하는 데 사용한 손실 함수 입니다. 이진 클래스 문제이므로 'binary_crossentropy'으로 지정합니다.
    - optimizer : 최적의 가중치를 검색하는 데 사용되는 최적화 알고리즘으로 효율적인 경사 하강법 알고리즘 중 하나인 'adam'을 사용합니다.
    - metrics : 평가 척도를 나타내며 분류 문제에서는 일반적으로 'accuracy'으로 지정합니다.

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

다층 퍼셉트론 모델 만들어보기

- 모델 학습시키기
  - 모델을 학습시키기 위해서 fit() 함수를 사용합니다.
    - 첫번째 인자 : 입력 변수입니다. 8개의 속성 값을 담고 있는 X를 입력합니다.
    - 두번째 인자 : 출력 변수 즉 라벨값입니다. 결과 값을 담고 있는 Y를 입력합니다.
    - epochs : 전체 훈련 데이터셋에 대해 학습 반복 횟수를 지정합니다. 1500번을 반복적으로 학습시켜 보겠습니다.
    - batch_size : 가중치를 업데이트할 배치 크기를 의미하며, 64개로 지정했습니다.

```
model.fit(x_train, y_train, epochs=1500, batch_size=64)
```

다층 퍼셉트론 모델 만들어보기

- 모델 평가하기
  - 시험셋으로 학습한 모델을 평가해봅니다.

```
scores = model.evaluate(x_test, y_test)
print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))
```

  - 77.94% 이라는 결과가 나왔습니다.
  - 평가 방법이 조금 다르기는 하지만 77.7%이라고 웹사이트 표기된 것에 비교하면 만족할 만한 수준입니다.

다층 퍼셉트론 모델 만들어보기

- 전체 소스

```
# 0. 사용할 패키지 불러오기
import numpy as np
from keras.models import Sequential
from keras.layers import Dense

# 랜덤시드 고정시키기
np.random.seed(5)

# 1. 데이터 준비하기
dataset = np.loadtxt("/content/gdrive/My Drive/pima-indians-diabetes.csv",
delimiter=",")

# 2. 데이터셋 생성하기
x_train = dataset[:700,0:8]
y_train = dataset[:700,8]
x_test = dataset[700:,0:8]
y_test = dataset[700:,8]
```

다층 퍼셉트론 모델 만들어보기

- 전체 소스

```
# 3. 모델 구성하기
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# 4. 모델 학습과정 설정하기
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 5. 모델 학습시키기
model.fit(x_train, y_train, epochs=1500, batch_size=64)

# 6. 모델 평가하기
scores = model.evaluate(x_test, y_test)
print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))
```

다층 퍼셉트론 모델 만들어보기

- 요약
  - 다층 퍼셉트론 모델을 만들어보고 실제 데이터셋을 사용하여 학습시켜봤습니다.
  - 수치로 된 데이터를 불러오는 법과 모델에 학습시키기 위해서 간단히 가공을 해봤습니다.
  - 또한 이진 분류 문제를 적용하기 위해서 입력 레이어와 출력 레이어를 어떻게 구성해야 하는 지도 알아봤습니다.

컴퓨터 비전과 사물 인식

- 컴퓨터 비전은 이미지에서 의미를 추출하는 프로그램을 연구하는 공학분야입니다.
- 오래된 '도시 전설' 에 따르면, 1960년대 MIT의 마빈 민스키 교수가 학부생들에게 낸 여름 방학 과제가 컴퓨터 비전의 시초라고 합니다.
- 당시 학생들은 컴퓨터에 카메라를 달고 이미지에 있는 모든 것을 설명하는 프로그램을 여름 내에 만들어야 했으며 당연히 아무도 끝내지 못했습니다.
- 컴퓨터 비전은 심지어 오늘날에도 과학자들이 연구를 지속하는 매우 복잡한 분야입니다.

컴퓨터 비전과 사물 인식

- 컴퓨터 비전의 초기 연구는 큰 성과를 내지 못했습니다.
- 1960년대 연구자들은 사진에 있는 형태나 선, 테두리 등을 감지하는 알고리즘을 만들었습니다.
- 그 후 수십 년간 컴퓨터 비전은 신호처리, 이미지 처리, 컴퓨터 측광(photometry), 사물 인식(object recognition) 등 여러 하위 분야로 나뉘어 발전했습니다.
- 그 중 사물 인식은 가장 보편적이며 오랜 기간 연구된 응용 분입니다.
- 초기에는 사물의 여러 다양한 외관을 인식하는 연구가 주를 이뤘습니다.
- 초기 연구자들은 템플릿을 사용한 사물 인식 기법을 집중 연구했지만 각도와 조명, 색상의 변화로 인해 종종 어려움을 겪었습니다.

컴퓨터 비전과 사물 인식

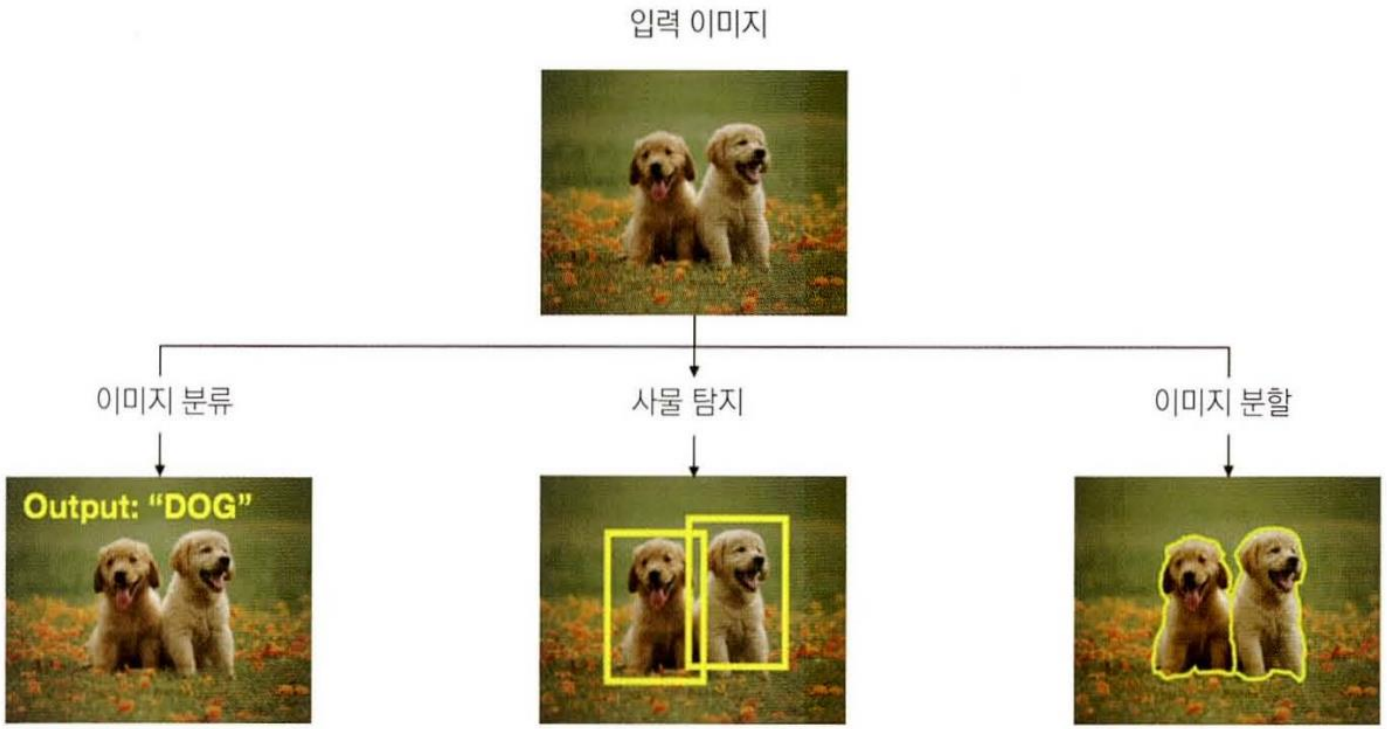
- 그러나 최근 신경망과 딥러닝의 진보에 힘입어 사물 인식 분야도 급격하게 발전했습니다.
- 2012년 알렉스 크리체프스키외 연구진은 사물 인식에 CNN을 사용해 ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 대회에서 다른 경쟁자를 큰 폭으로 제치고 우승했습니다(이 아키텍처를 AlexNet이라고 부릅니다).
- AlexNet은 사물 인식 연구에 획기적인 진보를 가져왔습니다.
- 이후 신경망은 사물 인식과 컴퓨터 비전 분야의 주류 기술로 떠올랐습니다.

컴퓨터 비전과 사물 인식

- 사물 인식 기술의 발전은 오늘날 인공 지능의 부흥으로 이어졌습니다.
- 페이스북은 안면 인식 기술을 사용해 사진 속의 사용자와 친구들을 자동으로 분류하고 태깅합니다.
- 안면 인식 기술은 보안 시스템에도 적용되어 사용자를 인증하거나 침입자를 탐지합니다.
- 또한, 자율 주행 자동자는 사물 인식 기술을 활용해 보행자, 신호등과 같은 도로 위의 다양한 사물을 감지합니다.
- 사실 근원이 매우 다른 사물 인식, 컴퓨터 비전, 인공 지능 기술은 이제 우리 삶 곳곳에 녹아들어 일반 대중은 기술 간 차이를 구분하지 못한 정도가 됐습니다.

사물 인식 기술 유형

- 사물 인식 기술의 유형에 따라 신경망 아키텍처도 크게 달라질 수 있습니다.
- 사물인식 기술은 크게 다음 세 가지 유형으로 나눌 수 있습니다.
  - 이미지 분류
  - 사물 탐지
  - 이미지 분할



사물 인식 기술 유형

- 이미지 분류
  - 이미지 분류 기술은 이미지를 입력받아 이미지가 속한 클래스를 예측합니다.
  - 앞 장에서 당뇨 발병 위험을 예측한 분류 모델과 유사합니다.
  - 앞 장의 분류 모델과 이미지 분류 문제의 다른 점은 판다스 DataFrame 형태의 테이블 데이터가 아닌 화소(pixel) (정확하게는 각 화소의 강도(intensity)를 입력 데이터로 사용한다는 점입니다.

사물 인식 기술 유형

- 사물 탐지
  - 사물 탐지는 이미지를 입력받아 사물을 탐지하고 주변에 경계선을 그리는 기술입니다.
  - 이미지 분류 기술보다 한 단계 더 진보했다고 볼 수 있습니다.
  - 이미지에 클래스가 하나만 있다고 단정할 수 없고 여러 클래스가 섞여 있다고 가정하기 때문입니다.
  - 사물 탐지 알고리즘은 이미지의 각 클래스를 인식하고 각각에 경계선을 그려야 합니다.
  - 신경망이 등장하기 전까지 사물 탐지는 굉장히 어려운 문제였지만, 오늘날 신경망은 사물 탐지 또한 효율적으로 수행할 수 있습니다.
  - AlexNet이 등장하고 나서 2년이 지난 후인 2014년에 길쉬크(Girshick)와 연구진은 이미지 분류 결과를 사물 탐지로 일반화하는 방법을 고안했습니다.
  - 먼저 사물이 있을 만한 위치에 경계선을 여러 개 만들고 나서 CNN을 사용해 각 경계선 안에 있는 사물의 클래스를 예측하는 방법으로, R-CNN(Region with CNN)이라고 부릅니다.

사물 인식 기술 유형

- 이미지 분할
  - 이미지 분할 기술은 이미지를 입력받아 각 클래스에 해당하는 화소들을 묶는 기술입니다.
  - 사물 탐지를 더욱 정교하게 다듬은 기술이며, 오늘날 다양한 응용에 활용됩니다.
  - 스마트폰 카메라에 대부분 탑재된 인물 사진 기능은 이미지 분할 기술을 사용해 배경과 전면 사물을 분리하거나 멋진 아웃 포커스 효과를 만들어 줍니다.
  - 이미지 분할 기술은 자율 주행 자동차에도 중요한데, 자동차 주변에 있는 각 사물의 위치를 매우 정밀하게 인식하는데 활용합니다.
  - 2017년 제안된 Mask R-CNN은 이미지 분할에 상당히 효과적인 기술입니다.

컨볼루션 신경망 레이어 이야기

- 이번 강좌에서는 컨볼루션 신경망 모델에서 주로 사용되는 컨볼루션(Convolution) 레이어, 맥스풀링(Max Pooling) 레이어, 플래튼(Flatten) 레이어에 대해서 알아보겠습니다.

- 각 레이어별로 레이어 구성 및 역할에 대해서 알아보겠습니다.

컨볼루션 신경망 레이어 이야기

- 필터로 특징을 뽑아주는 컨볼루션(Convolution) 레이어
  - 케라스에서 제공되는 컨볼루션 레이어 종류에도 여러가지가 있으나 영상 처리에 주로 사용되는 Conv2D 레이어를 살펴보겠습니다.
  - 레이어는 영상 인식에 주로 사용되며, 필터가 탑재되어 있습니다.
  - 아래는 Conv2D 클래스 사용 예제입니다.

```
Conv2D(32, (5, 5), padding='valid', input_shape=(28, 28, 1), activation='relu')
```

컨볼루션 신경망 레이어 이야기

- 필터로 특징을 뽑아주는 컨볼루션(Convolution) 레이어
  - 주요 인자는 다음과 같습니다.
    - 첫번째 인자 : 컨볼루션 필터의 수 입니다.
    - 두번째 인자 : 컨볼루션 커널의 (행, 열) 입니다.
    - padding : 경계 처리 방법을 정의합니다.
      - ‘valid’ : 유효한 영역만 출력이 됩니다. 따라서 출력 이미지 사이즈는 입력 사이즈보다 작습니다.
      - ‘same’ : 출력 이미지 사이즈가 입력 이미지 사이즈와 동일합니다.
    - input_shape : 샘플 수를 제외한 입력 형태를 정의 합니다. 모델에서 첫 레이어일 때만 정의하면 됩니다.
      - (행, 열, 채널 수)로 정의합니다. 흑백영상인 경우에는 채널이 1이고, 컬러(RGB) 영상인 경우에는 채널을 3으로 설정합니다.

컨볼루션 신경망 레이어 이야기

- 필터로 특징을 뽑아주는 컨볼루션(Convolution) 레이어
  - 주요 인자는 다음과 같습니다.
    - activation : 활성화 함수를 설정합니다.
      - ‘linear’ : 디폴트 값, 입력뉴런과 가중치로 계산된 결과값이 그대로 출력으로 나옵니다.
      - ‘relu’ : rectifier 함수, 은닉층에 주로 쓰입니다.
      - ‘sigmoid’ : 시그모이드 함수, 이진 분류 문제에서 출력층에 주로 쓰입니다.
      - ‘softmax’ : 소프트맥스 함수, 다중 클래스 분류 문제에서 출력층에 주로 쓰입니다.
  - 입력 형태는 다음과 같습니다.
    - image_data_format이 'channels_first'인 경우 (샘플 수, 채널 수, 행, 열)로 이루어진 4D 텐서입니다.
    - image_data_format이 'channels_last'인 경우 (샘플 수, 행, 열, 채널 수)로 이루어진 4D 텐서입니다.
    - image_data_format 옵션은 "keras.json" 파일 안에 있는 설정입니다. 콘솔에서 "vi ~/.keras/keras.json"으로 keras.json 파일 내용을 변경할 수 있습니다.

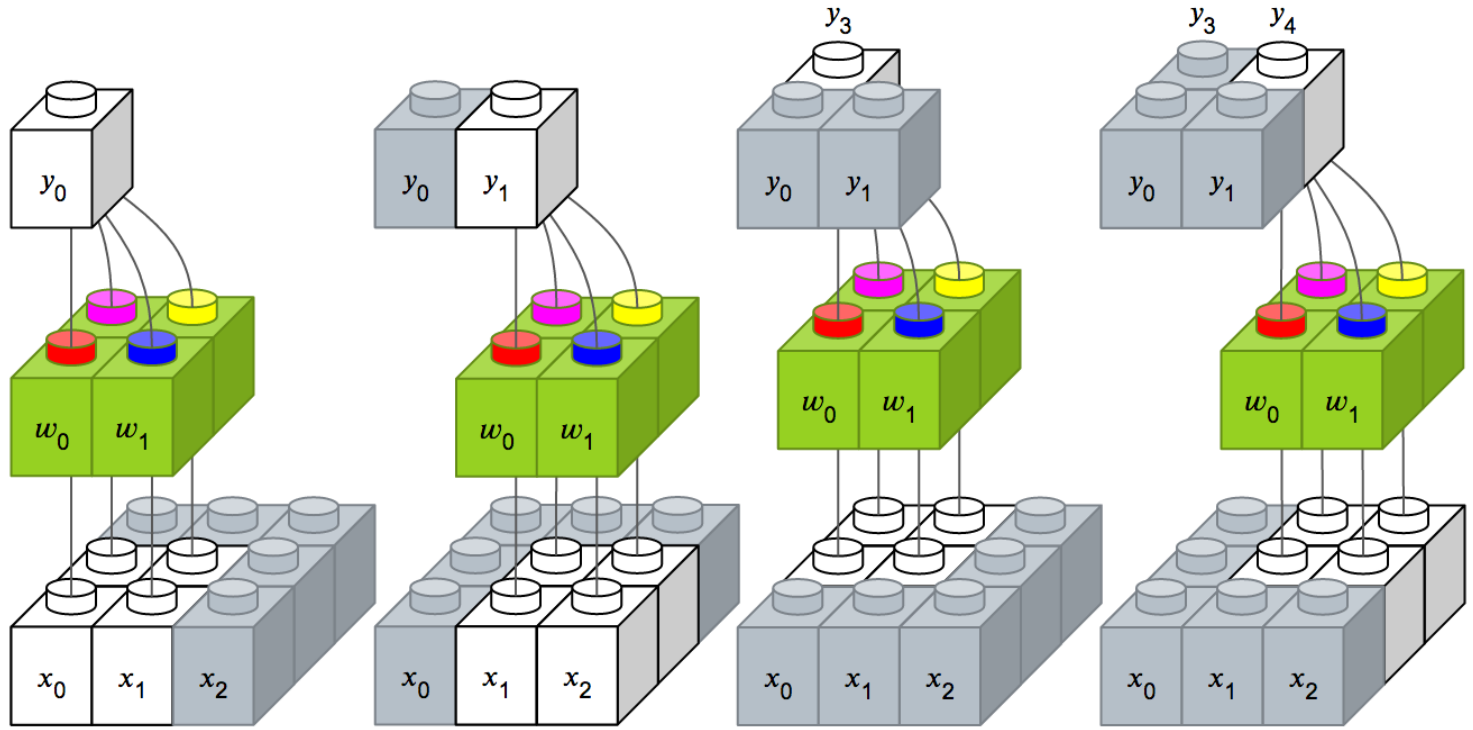
컨볼루션 신경망 레이어 이야기

- 필터로 특징을 뽑아주는 컨볼루션(Convolution) 레이어
  - 출력 형태는 다음과 같습니다.
    - image_data_format이 'channels_first'인 경우 (샘플 수, 필터 수, 행, 열)로 이루어진 4D 텐서입니다.
    - image_data_format이 'channels_last'인 경우 (샘플 수, 행, 열, 필터 수)로 이루어진 4D 텐서입니다.
    - 행과 열의 크기는 padding가 'same'인 경우에는 입력 형태의 행과 열의 크기가 동일합니다.
  - 간단한 예제로 컨볼루션 레이어와 필터에 대해서 알아보겠습니다.
  - 입력 이미지는 채널 수가 1, 너비가 3 픽셀, 높이가 3 픽셀이고, 크기가 2 x 2인 필터가 하나인 경우를 레이어로 표시하면 다음과 같습니다.
  - 단 image_data_format이 'channels_last'인 경우 입니다.

```
Conv2D(1, (2, 2), padding='valid', input_shape=(3, 3, 1))
```

컨볼루션 신경망 레이어 이야기

- 필터로 특징을 뽑아주는 컨볼루션(Convolution) 레이어
  - 이를 도식화하면 다음과 같습니다.



컨볼루션 신경망 레이어 이야기

- 필터로 특징을 뽑아주는 컨볼루션(Convolution) 레이어
  - 필터는 가중치를 의미합니다.
  - 하나의 필터가 입력 이미지를 순회하면서 적용된 결과값을 모으면 출력 이미지가 생성됩니다.
  - 여기에는 두 가지 특성이 있습니다.
  - 하나의 필터로 입력 이미지를 순회하기 때문에 순회할 때 적용되는 가중치는 모두 동일합니다.
  - 이를 파라미터 공유라고 부릅니다.
  - 이는 학습해야 할 가중치 수를 현저하게 줄여줍니다.
  - 출력에 영향을 미치는 영역이 지역적으로 제한되어 있습니다.
  - 즉 그림에서 y0에 영향을 미치는 입력은 x0, x1, x3, x4으로 한정되어 있습니다.
  - 이는 지역적인 특징을 잘 뽑아내게 되어 영상 인식에 적합합니다.
  - 예를 들어 코를 볼 때는 코 주변만 보고, 눈을 볼 때는 눈 주변만 보면서 학습 및 인식하는 것입니다.

컨볼루션 신경망 레이어 이야기

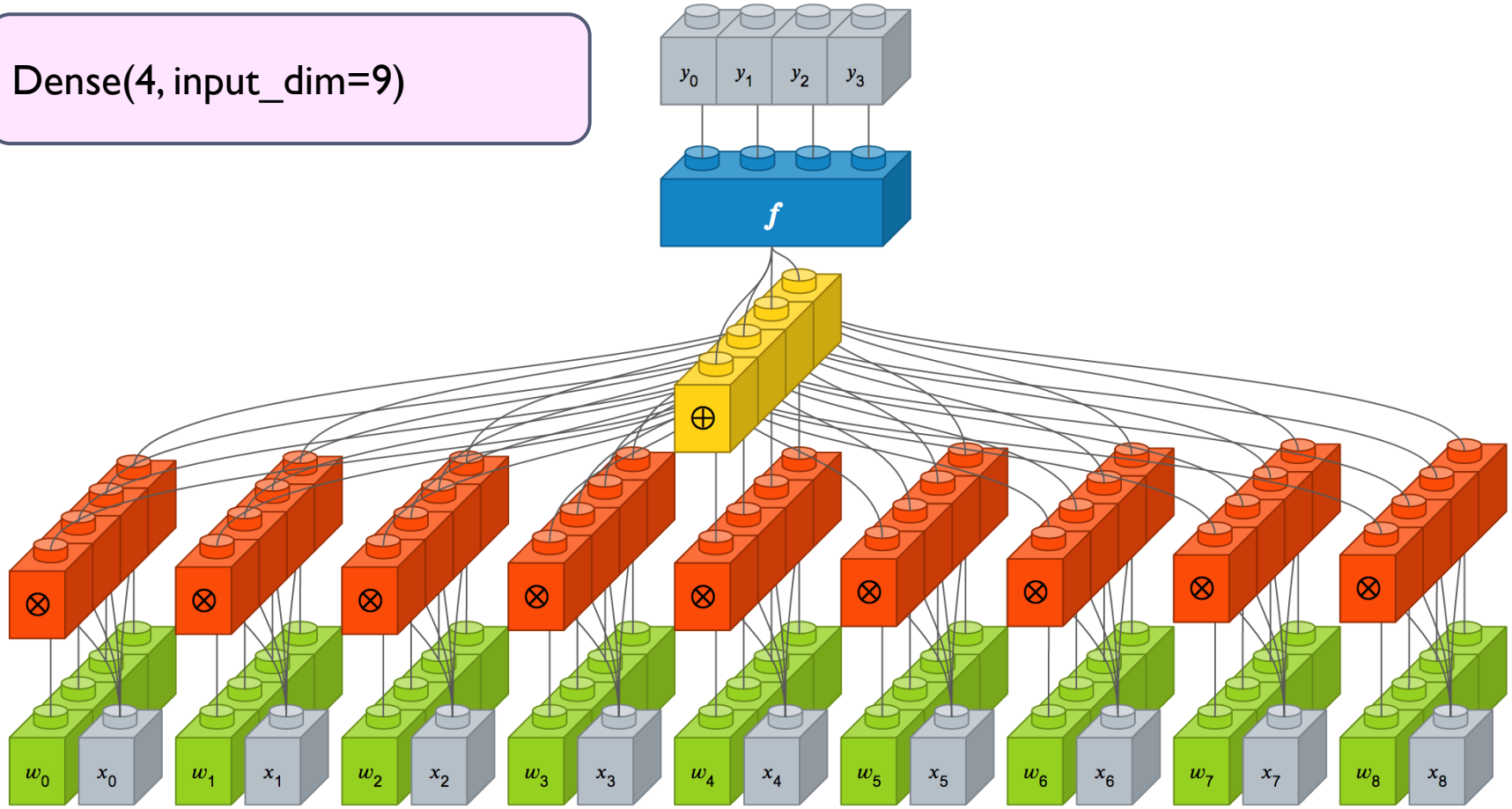
- 가중치의 수
  - 이를 Dense 레이어와 컨볼루션 레이어와 비교를 해보면서 차이점을 알아보겠습니다.
  - 영상도 결국에는 픽셀의 집합이므로 입력 뉴런이 9개 (3 x 3)이고, 출력 뉴런이 4개 (2 x 2)인 Dense 레이어로 표현할 수 있습니다.

```
Dense(4, input_dim=9)
```

컨볼루션 신경망 레이어 이야기

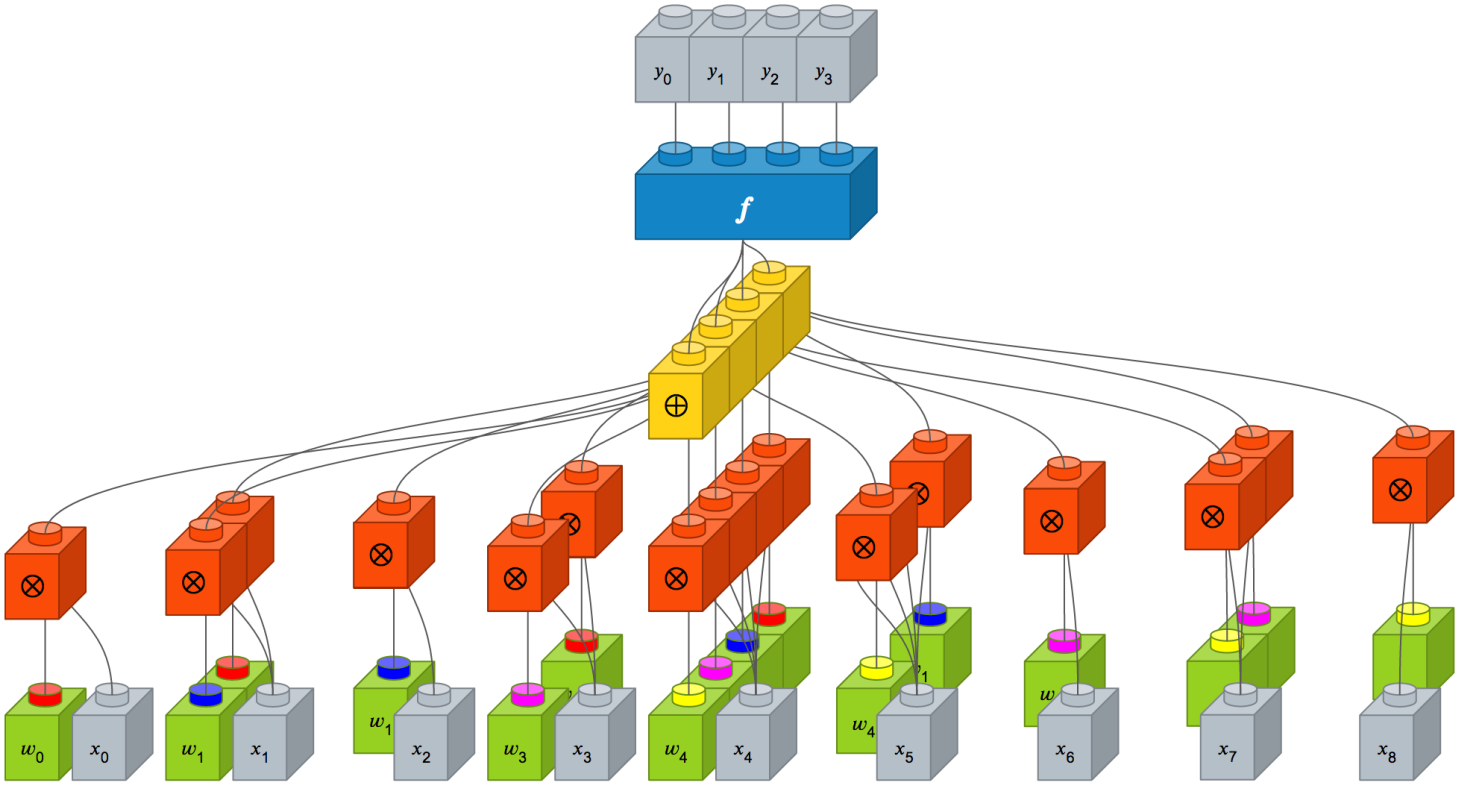
- 가중치의 수
  - 이를 도식화하면 다음과 같습니다.

Dense(4, input_dim=9)



컨볼루션 신경망 레이어 이야기

- 가중치의 수
  - 가중치 즉 시냅스 강도는 녹색 블럭으로 표시되어 있습니다.
  - 컨볼루션 레이어에서는 가중치 4개로 구성된 크기가 2 x 2인 필터를 적용하였을 때의 뉴런 상세 구조는 다음과 같습니다.

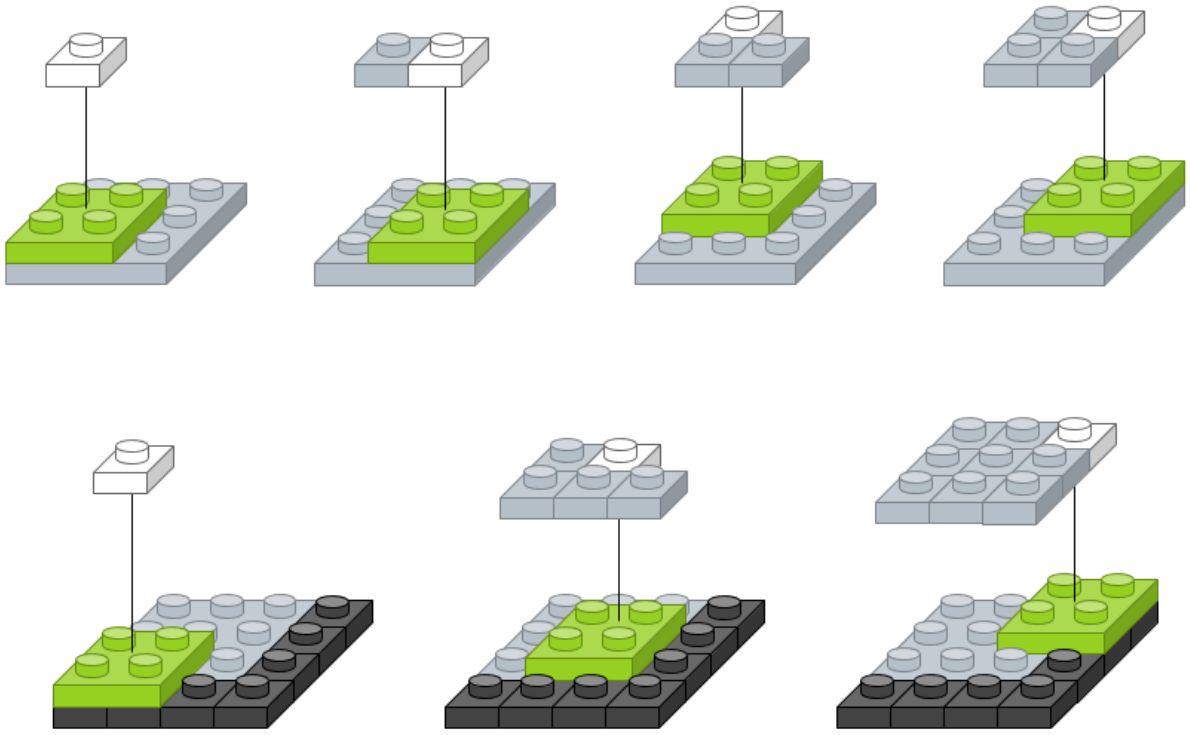


컨볼루션 신경망 레이어 이야기

- 가중치의 수
  - 필터가 지역적으로만 적용되어 출력 뉴런에 영향을 미치는 입력 뉴런이 제한적이므로 Dense 레이어와 비교했을 때, 가중치가 많이 줄어든 것을 보실 수 있습니다.
  - 게다가 녹색 블럭 상단에 표시된 빨간색, 파란색, 분홍색, 노란색끼리는 모두 동일한 가중치(파라미터 공유)이므로 결국 사용되는 가중치는 4개입니다.
  - 즉 Dense 레이어에서는 36개의 가중치가 사용되었지만, 컨볼루션 레이어에서는 필터의 크기인 4개의 가중치만을 사용합니다.

컨볼루션 신경망 레이어 이야기

- 경계 처리 방법
  - 이번에는 경계 처리 방법에 대해서 알아봅시다.
  - 컨볼루션 레이어 설정 옵션에는 border_mode가 있는데, ‘valid’와 ‘same’으로 설정할 수 있습니다.
  - 이 둘의 차이는 아래 그림에서 확인할 수 있습니다.



컨볼루션 신경망 레이어 이야기

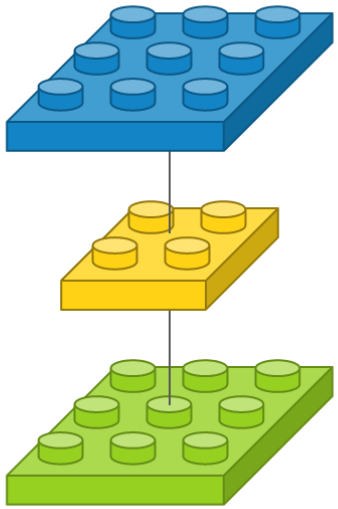
- 경계 처리 방법
  - ‘valid’인 경우에는 입력 이미지 영역에 맞게 필터를 적용하기 때문에 출력 이미지 크기가 입력 이미지 크기보다 작아집니다.
  - 반면에 ‘same’은 출력 이미지와 입력 이미지 사이즈가 동일하도록 입력 이미지 경계에 빈 영역을 추가하여 필터를 적용합니다.
  - ‘same’으로 설정 시, 입력 이미지에 경계를 학습시키는 효과가 있습니다.

컨볼루션 신경망 레이어 이야기

- 필터 수
  - 다음은 필터의 개수에 대해서 알아봅니다.
  - 입력 이미지가 단채널의 3 x 3이고, 2 x 2인 필터가 하나 있다면 다음과 같이 컨볼루션 레이어를 정의할 수 있습니다.

```
Conv2D(1, (2, 2), padding='same', input_shape=(3, 3, 1))
```

  - 이를 도식화하면 다음과 같습니다.

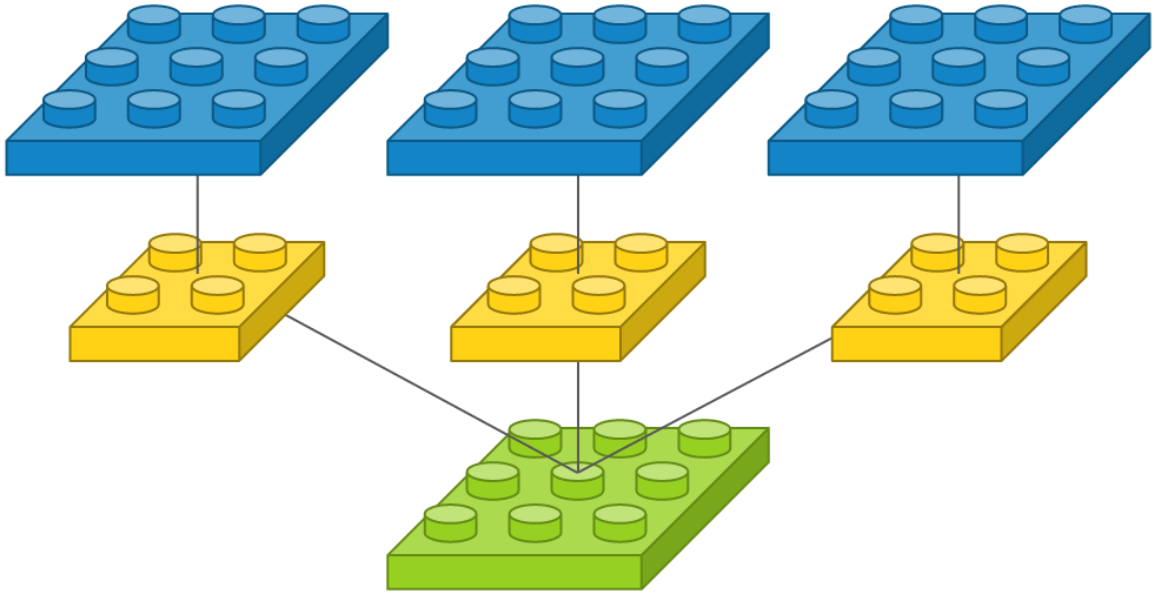


컨볼루션 신경망 레이어 이야기

- 필터 수
  - 만약 여기서 사이즈가 2 x 2 필터를 3개 사용한다면 다음과 같이 정의할 수 있습니다.

```
Conv2D(3, (2, 2), padding='same', input_shape=(3, 3, 1))
```

  - 이를 도식화하면 다음과 같습니다.

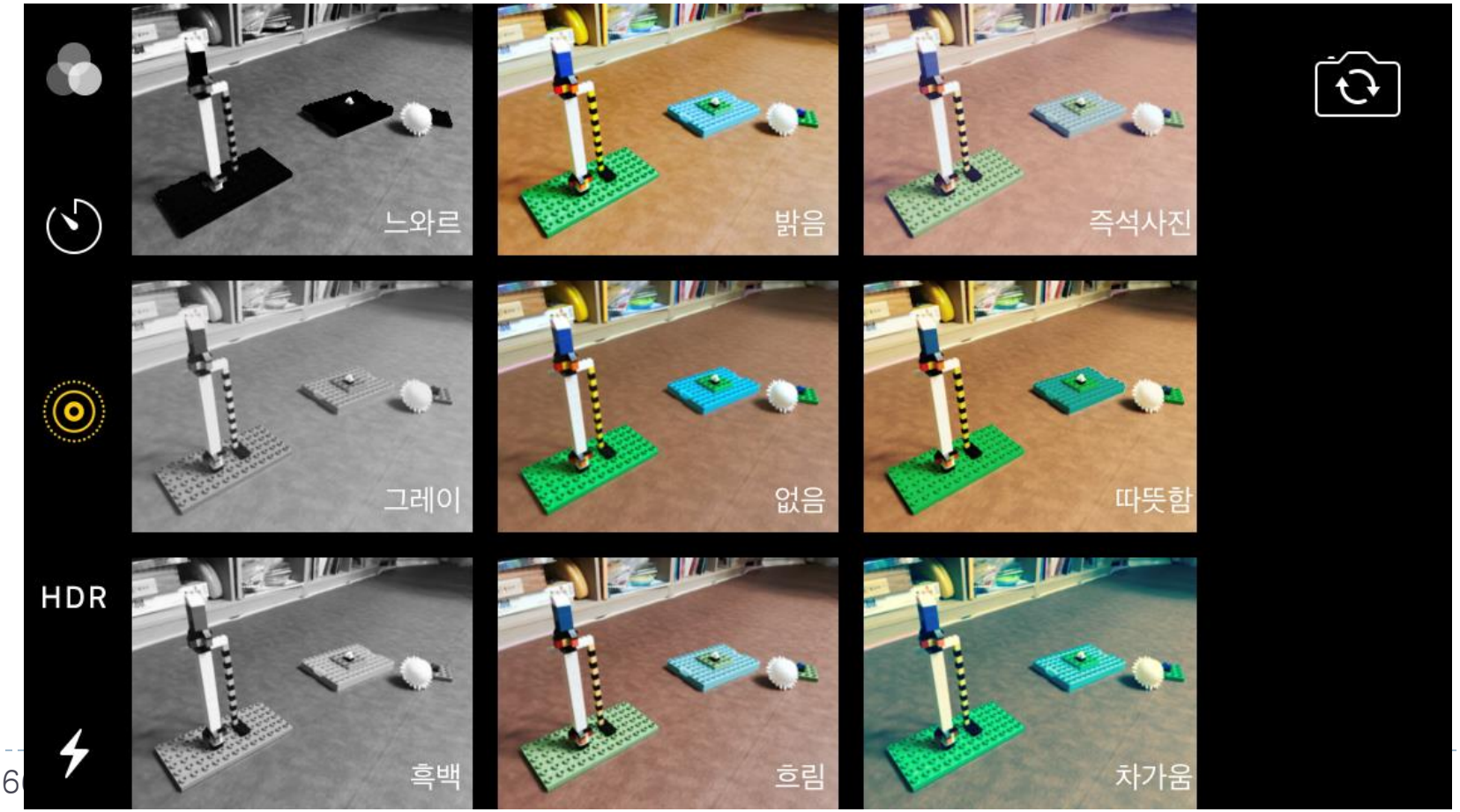


컨볼루션 신경망 레이어 이야기

- 필터 수
  - 여기서 살펴봐야할 것은 필터가 3개라서 출력 이미지도 필터 수에 따라 3개로 늘어났습니다.
  - 총 가중치의 수는 3 x 2 x 2으로 12개입니다.
  - 필터마다 고유한 특징을 뽑아 고유한 출력 이미지로 만들기 때문에 필터의 출력값을 더해서 하나의 이미지로 만들거나 그렇게 하지 않습니다.
  - 필터에 대해 생소하신 분은 카메라 필터라고 생각하시면 됩니다.
  - 스마트폰 카메라로 사진을 찍을 때 필터를 적용해볼 수 있는 데, 적용되는 필터 수에 따라 다른 사진이 나옴을 알 수 있습니다.

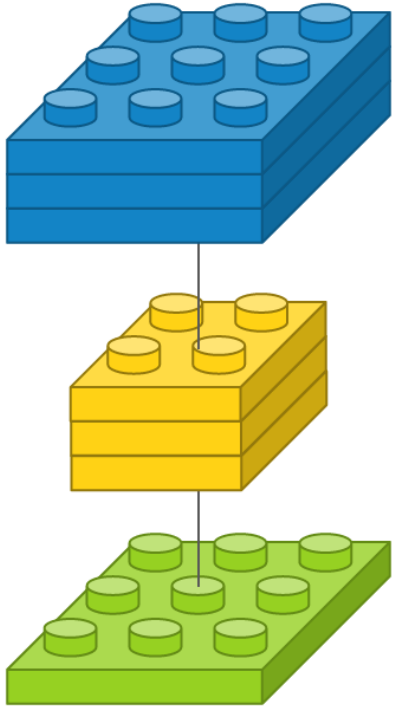
컨볼루션 신경망 레이어 이야기

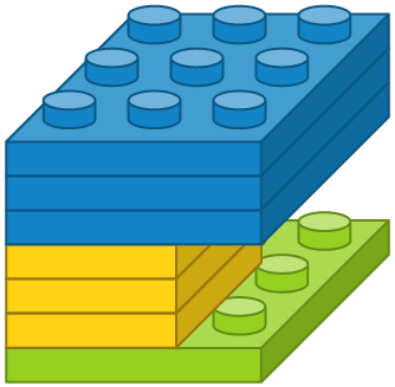
- 필터 수
  - 스마트폰 카메라로 사진을 찍을 때 필터를 적용해볼 수 있는 데, 적용되는 필터 수에 따라 다른 사진이 나옴을 알 수 있습니다.



컨볼루션 신경망 레이어 이야기

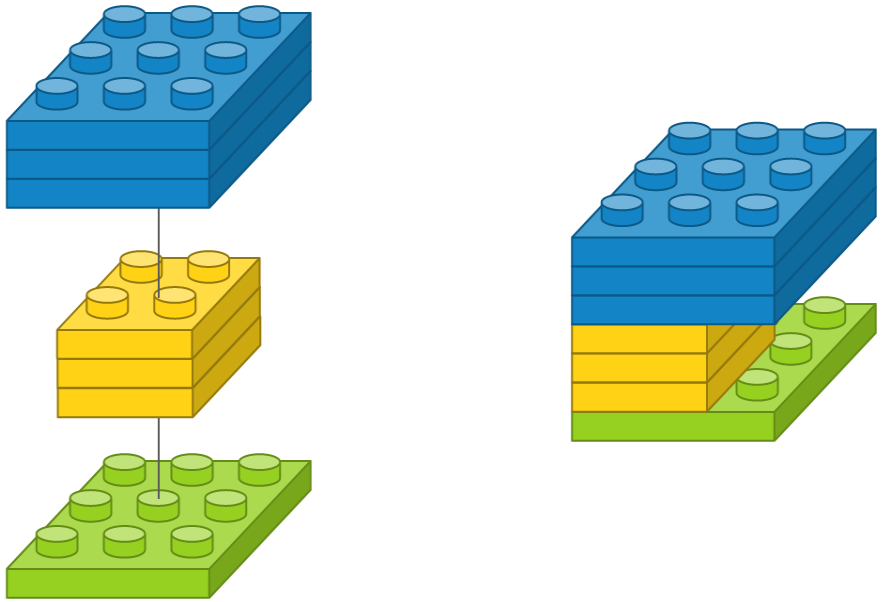
- 필터 수
  - 뒤에서 각 레이어를 레고처럼 쌓아올리기 위해서 약식으로 표현하면 다음과 같습니다.





컨볼루션 신경망 레이어 이야기

- 필터 수
  - 이 표현은 다음을 의미합니다.
  - 입력 이미지 사이즈가 3 x 3 입니다.
  - 2 x 2 커널을 가진 필터가 3개입니다. 가중치는 총 12개 입니다.
  - 출력 이미지 사이즈가 3 x 3이고 총 3개입니다. 이는 채널이 3개다라고도 표현합니다.

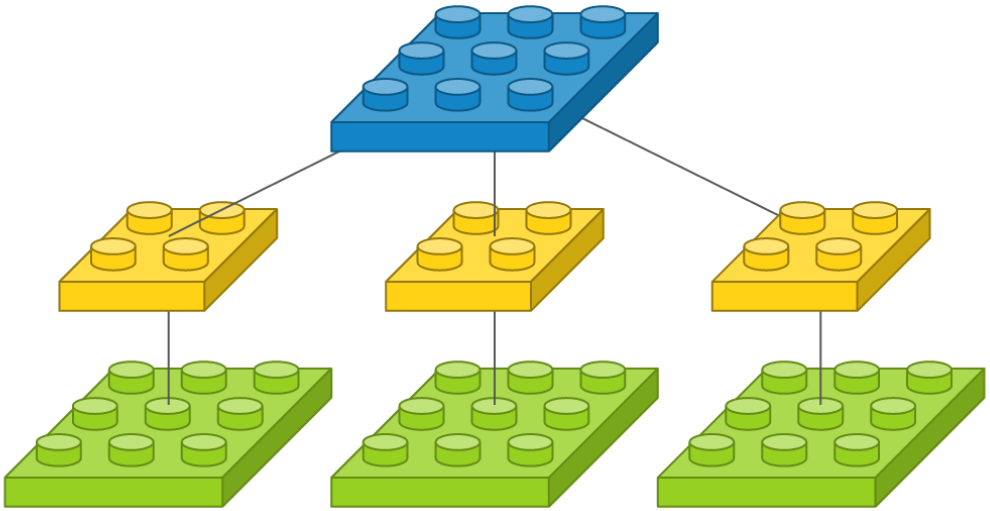


컨볼루션 신경망 레이어 이야기

- 필터 수
  - 다음은 입력 이미지의 채널이 여러 개인 경우를 살펴보겠습니다.
  - 만약 입력 이미지의 채널이 3개이고 사이즈가 3 x 3이고, 사이즈가 2 x 2 필터를 1개 사용한다면 다음과 같이 컨볼루션 레이어를 정의할 수 있습니다.

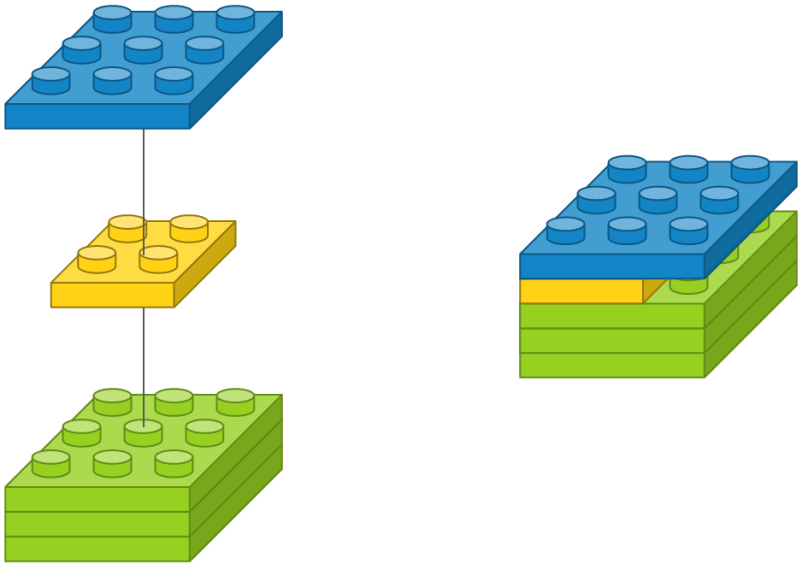
```
Conv2D(1, (2, 2), padding='same', input_shape=(3, 3, 3))
```

  - 이를 도식화하면 다음과 같습니다.



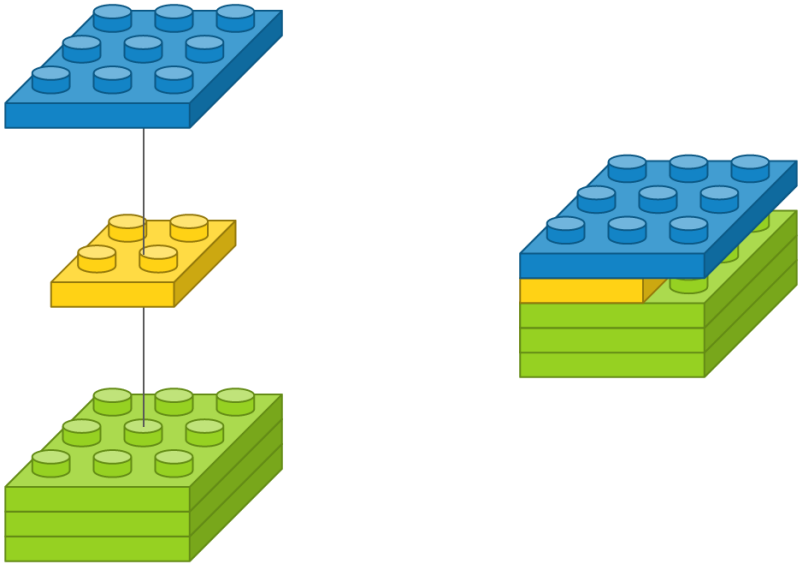
컨볼루션 신경망 레이어 이야기

- 필터 수
  - 필터 개수가 3개인 것처럼 보이지만 이는 입력 이미지에 따라 할당되는 커널이고, 각 커널의 계산 값이 결국 더해져서 출력 이미지 한 장을 만들어내므로 필터 개수는 1개입니다.
  - 이는 Dense 레이어에서 입력 뉴런이 늘어나면 거기에 상응하는 시냅스에 늘어나서 가중치의 수가 늘어나는 것과 같은 원리입니다.
  - 가중치는 2 x 2 x 3으로 총 12개 이지만 필터 수는 1개입니다. 이를 약식으로 표현하면 다음과 같습니다.



컨볼루션 신경망 레이어 이야기

- 필터 수
  - 이 표현은 다음을 의미합니다.
    - 입력 이미지 사이즈가 3 x 3 이고 채널이 3개입니다.
    - 2 x 2 커널을 가진 필터가 1개입니다.
    - 채널마다 커널이 할당되어 총 가중치는 12개 입니다.
    - 출력 이미지는 사이즈가 3 x 3 이고 채널이 1개입니다.

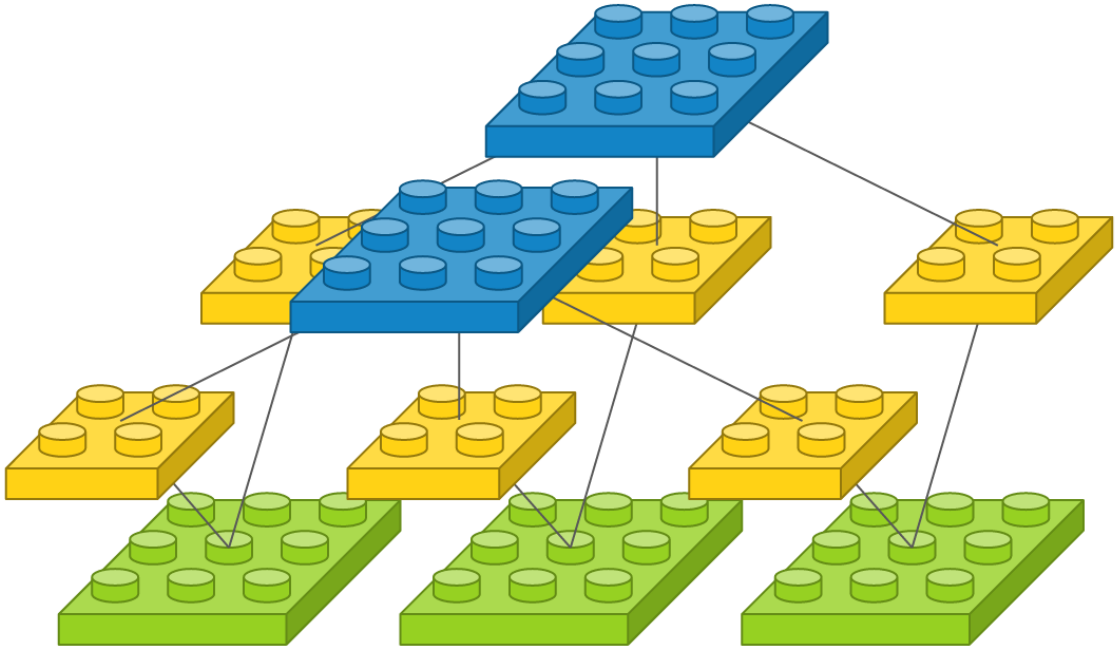


컨볼루션 신경망 레이어 이야기

- 필터 수
  - 마지막으로 입력 이미지의 사이즈가 3 x 3이고 채널이 3개이고, 사이즈가 2 x 2인 필터가 2개인 경우를 살펴보겠습니다.

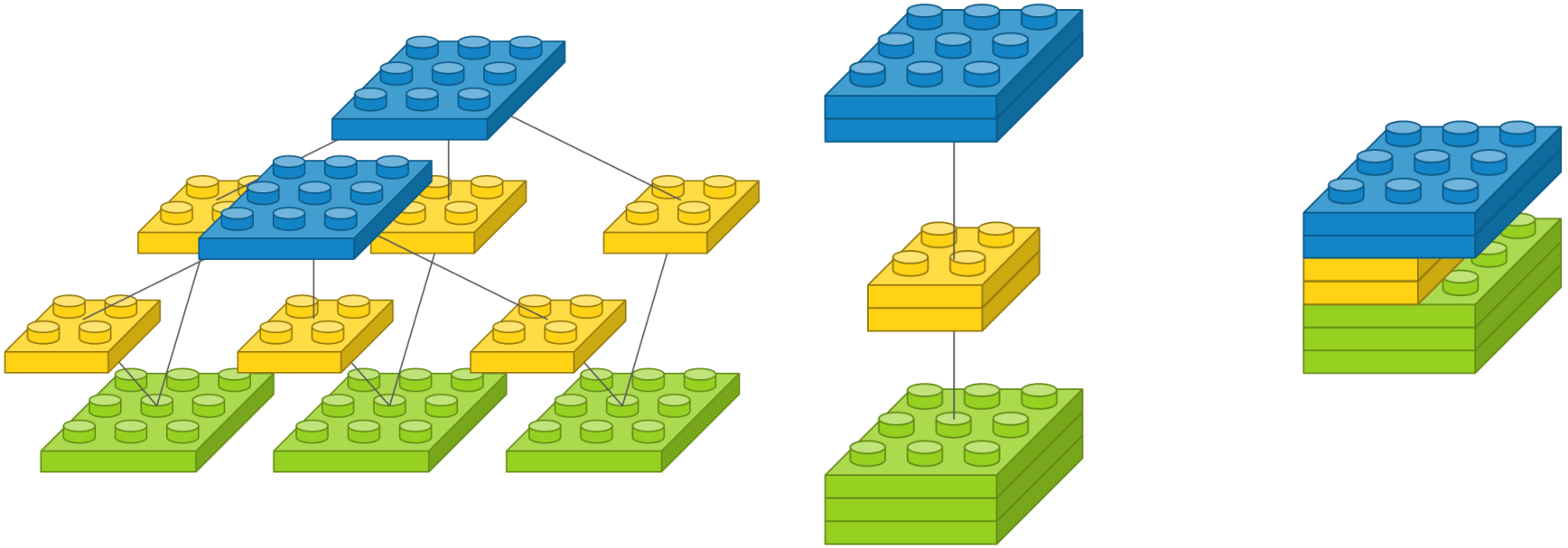
```
Conv2D(2, (2, 2), padding='same', input_shape=(3, 3, 3))
```

  - 이를 도식화하면 다음과 같습니다.



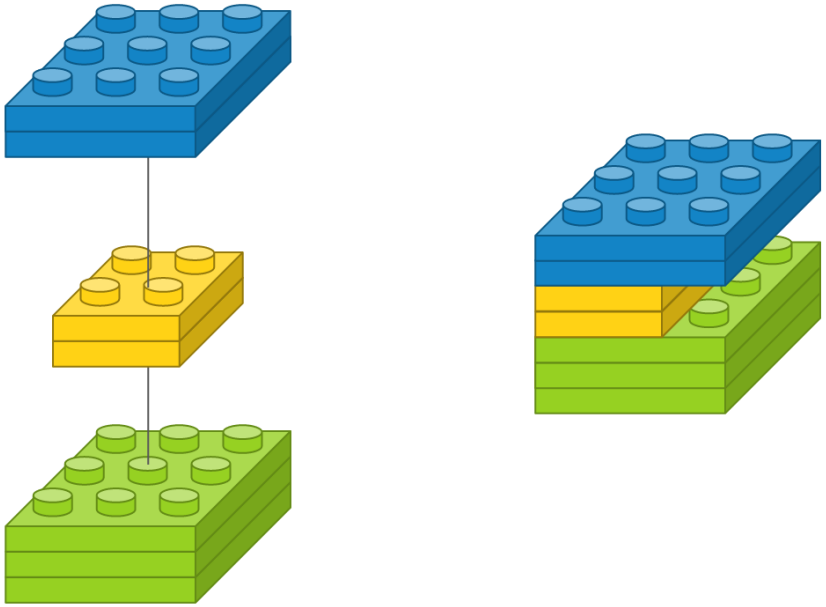
컨볼루션 신경망 레이어 이야기

- 필터 수
  - 필터가 2개이므로 출력 이미지도 2개입니다.
  - 약식 표현은 다음과 같습니다.



컨볼루션 신경망 레이어 이야기

- 필터 수
  - 이 표현은 다음을 의미합니다.
    - 입력 이미지 사이즈가 3 x 3 이고 채널이 3개입니다.
    - 2 x 2 커널을 가진 필터가 2개입니다.
    - 채널마다 커널이 할당되어 총 가중치는 3 x 2 x 2 x 2로 24개 입니다.
    - 출력 이미지는 사이즈가 3 x 3 이고 채널이 2개입니다.



컨볼루션 신경망 레이어 이야기

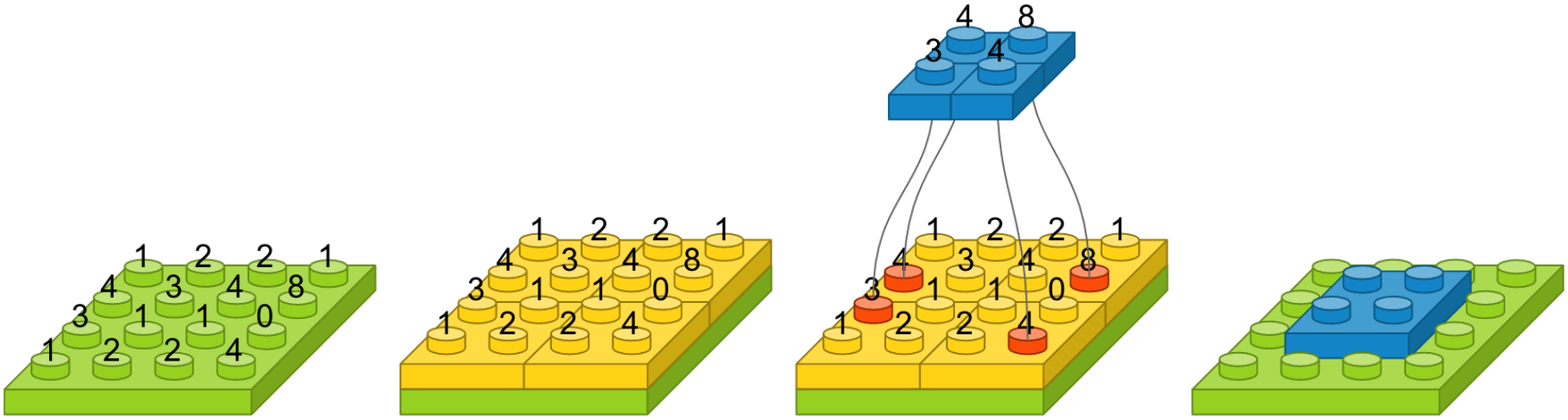
- 사소한 변화를 무시해주는 맥스풀링(Max Pooling) 레이어
  - 컨볼루션 레이어의 출력 이미지에서 주요값만 뽑아 크기가 작은 출력 영상을 만듭니다.
  - 이것은 지역적인 사소한 변화가 영향을 미치지 않도록 합니다.

```
MaxPooling2D(pool_size=(2, 2))
```

  - 주요 인자는 다음과 같습니다.
    - pool_size : 수직, 수평 축소 비율을 지정합니다.
    - (2, 2)이면 출력 영상 크기는 입력 영상 크기의 반으로 줄어듭니다.

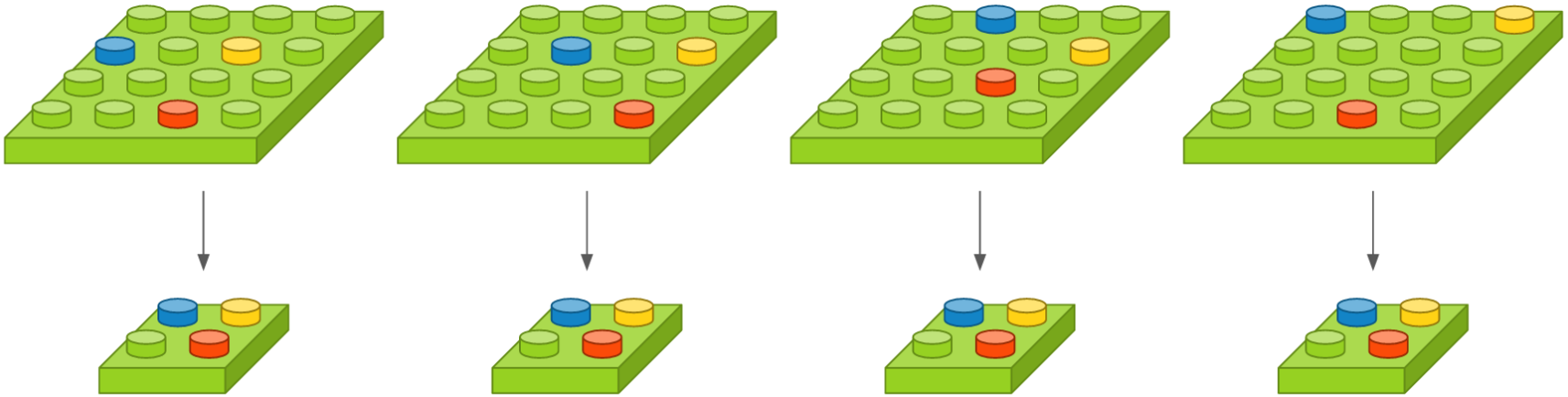
컨볼루션 신경망 레이어 이야기

- 사소한 변화를 무시해주는 맥스풀링(Max Pooling) 레이어
  - 예를 들어, 입력 영상 크기가 4 x 4이고, 풀 크기를 (2, 2)로 했을 때를 도식화하면 다음과 같습니다.
  - 녹색 블럭은 입력 영상을 나타내고, 노란색 블럭은 풀 크기에 따라 나눈 경계를 표시합니다.
  - 해당 풀에서 가장 큰 값을 선택하여 파란 블럭으로 만들면, 그것이 출력 영상이 됩니다.
  - 가장 오른쪽은 맥스풀링 레이어를 약식으로 표시한 것입니다.



컨볼루션 신경망 레이어 이야기

- 사소한 변화를 무시해주는 맥스풀링(Max Pooling) 레이어
  - 이 레이어는 영상의 작은 변화라던지 사소한 움직임이 특징을 추출할 때 크게 영향을 미치지 않도록 합니다.
  - 영상 내에 특징이 세 개가 있다고 가정했을 때, 아래 그림에서 첫 번째 영상을 기준으로 두 번째 영상은 오른쪽으로 이동하였고, 세 번째 영상은 약간 비틀어 졌고, 네 번째 영상은 조금 확대되었지만, 맥스풀링한 결과는 모두 동일합니다.
  - 얼굴 인식 문제를 예를 들면, 맥스풀링의 역할은 사람마다 눈, 코, 입 위치가 조금씩 다른데 이러한 차이가 사람이라고 인식하는 데 있어서는 큰 영향을 미치지 않게 합니다.



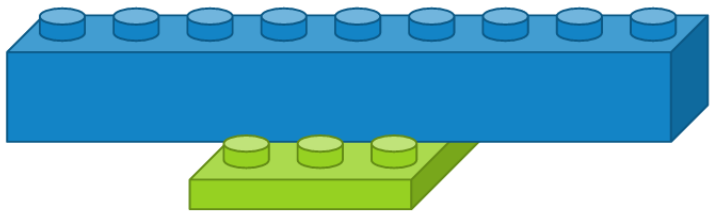
컨볼루션 신경망 레이어 이야기

- 영상을 일차원으로 바꿔주는 플래튼(Flatten) 레이어
  - CNN에서 컨볼루션 레이어나 맥스풀링 레이어를 반복적으로 거치면 주요 특징만 추출되고, 추출된 주요 특징은 전결합층에 전달되어 학습됩니다.
  - 컨볼루션 레이어나 맥스풀링 레이어는 주로 2차원 자료를 다루지만 전결합층에 전달하기 위해선 1차원 자료로 바꿔줘야 합니다.
  - 이 때 사용되는 것이 플래튼 레이어입니다.
  - 사용 예시는 다음과 같습니다.

```
Flatten()
```

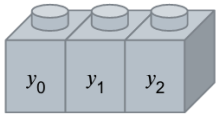
컨볼루션 신경망 레이어 이야기

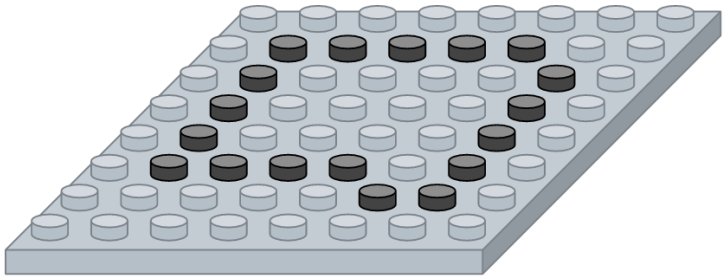
- 영상을 일차원으로 바꿔주는 플래튼(Flatten) 레이어
  - 이전 레이어의 출력 정보를 이용하여 입력 정보를 자동으로 설정되며, 출력 형태는 입력 형태에 따라 자동으로 계산되기 때문에 별도로 사용자가 파라미터를 지정해주지 않아도 됩니다.
  - 크기가 3 x 3인 영상을 1차원으로 변경했을 때는 도식화하면 다음과 같습니다.



컨볼루션 신경망 레이어 이야기

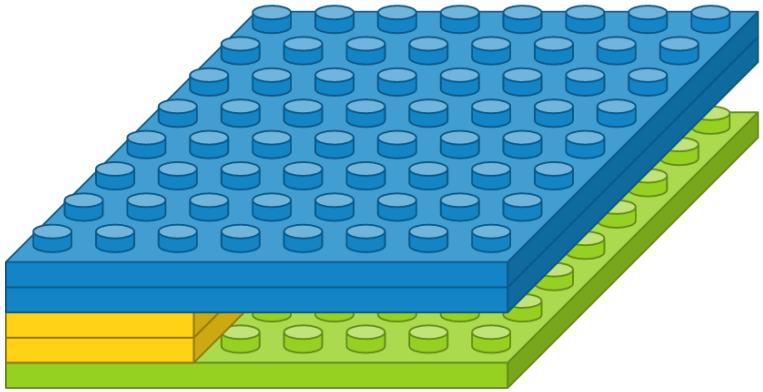
- 한 번 쌓아보기
  - 지금까지 알아본 레이어를 이용해서 간단한 컨볼루션 신경망 모델을 만들어보겠습니다.
  - 먼저 간단한 문제를 정의해봅시다. 손으로 삼각형, 사각형, 원을 손으로 그린 이미지가 있고 이미지 크기가 8×8이라고 가정해봅니다.
  - 삼각형, 사각형, 원을 구분하는 3개의 클래스를 분류하는 문제이기 때문에 출력 벡터는 3개여야 합니다.
  - 필요하다고 생각하는 레이어를 구성해봤습니다.



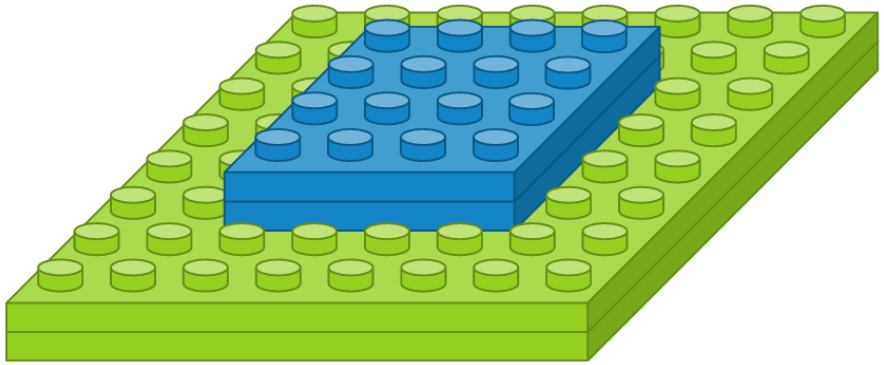


컨볼루션 신경망 레이어 이야기

- 한 번 쌓아보기
  - 컨볼루션 레이어 : 입력 이미지 크기 8 x 8, 입력 이미지 채널 1개, 필터 크기 3 x 3, 필터 수 2개, 경계 타입 'same', 활성화 함수 'relu'

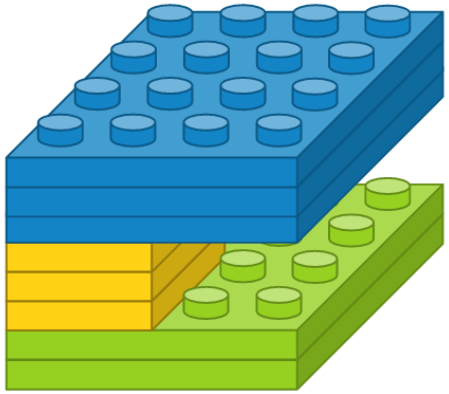


  - 맥스풀링 레이어 : 풀 크기 2 x 2

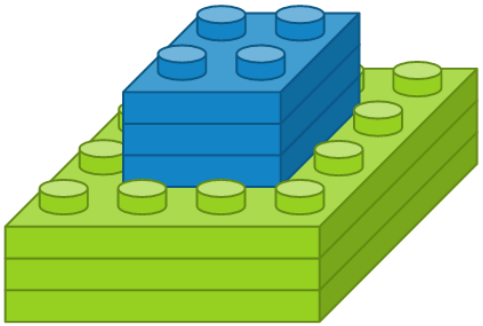


컨볼루션 신경망 레이어 이야기

- 한 번 쌓아보기
  - 컨볼루션 레이어 : 입력 이미지 크기 4 x 4, 입력 이미지 채널 2개, 필터 크기 2 x 2, 필터 수 3개, 경계 타입 'same', 활성화 함수 'relu'

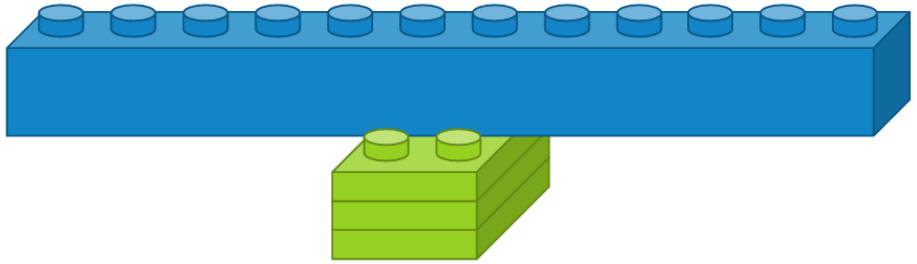


  - 맥스풀링 레이어 : 풀 크기 2 x 2

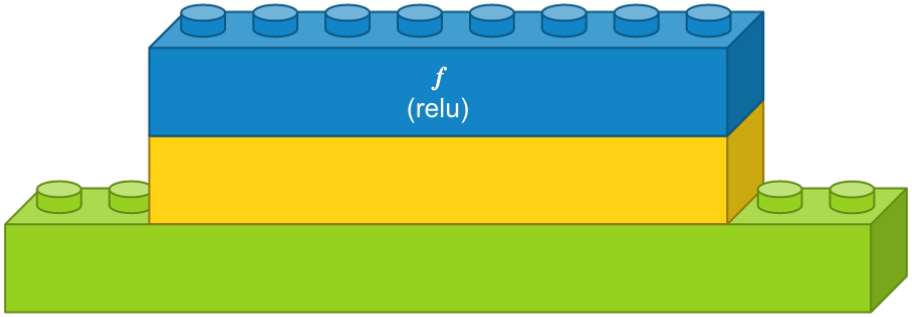


컨볼루션 신경망 레이어 이야기

- 한 번 쌓아보기
  - 플래튼 레이어

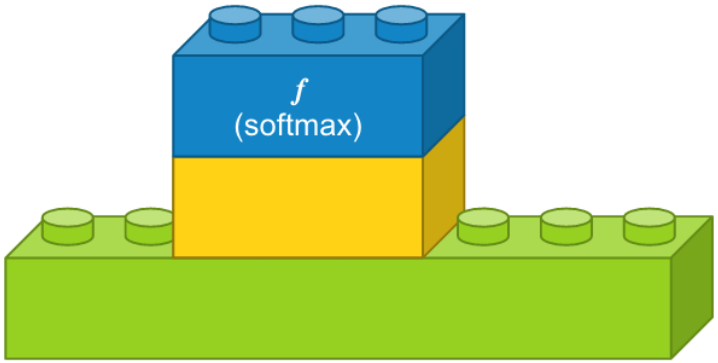


  - 댄스 레이어 : 입력 뉴런 수 12개, 출력 뉴런 수 8개, 활성화 함수 'relu'



컨볼루션 신경망 레이어 이야기

- 한 번 쌓아보기
  - 댄스 레이어 : 입력 뉴런 수 8개, 출력 뉴런 수 3개, 활성화 함수 'softmax'

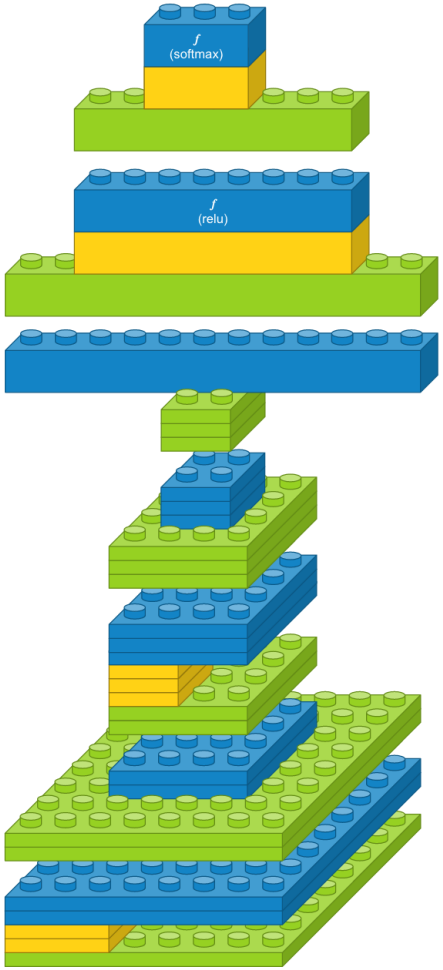


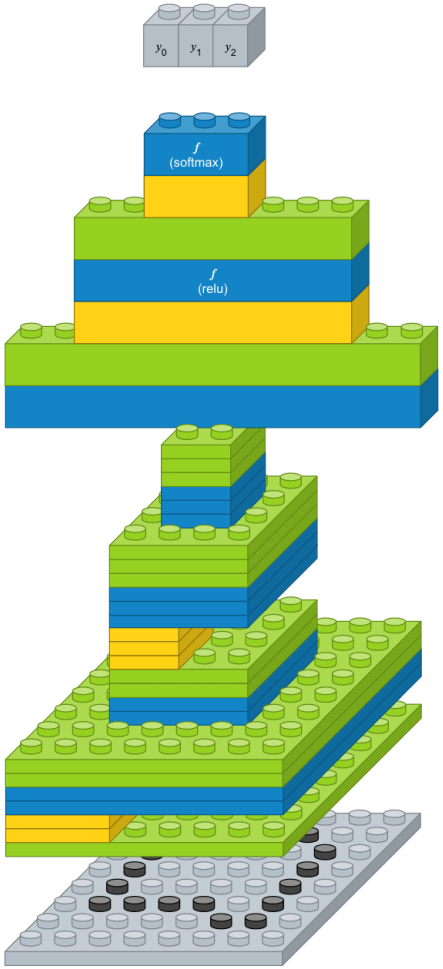
컨볼루션 신경망 레이어 이야기

- 한 번 쌓아보기
  - 모든 레이어 블럭이 준비되었으니 이를 조합해 봅니다.
  - 입출력 크기만 맞으면 레고 끼우듯이 합치면 됩니다.
  - 참고로 케라스 코드에서는 가장 첫번째 레이어를 제외하고는 입력 형태를 자동으로 계산하므로 이 부분은 신경쓰지 않아도 됩니다.
  - 레이어를 조립하니 간단한 컨볼루션 모델이 생성되었습니다.
  - 이 모델에 이미지를 입력하면, 삼각형, 사각형, 원을 나타내는 벡터가 출력됩니다.

컨볼루션 신경망 레이어 이야기

- 한 번 쌓아보기
  - 모든 레이어 블럭이 준비되었으니 이를 조합해 봅니다.





컨볼루션 신경망 레이어 이야기

- 한 번 쌓아보기
  - 그럼 케라스 코드로 어떻게 구현하는 지 알아봅니다.
  - 먼저 필요한 패키지를 추가하는 과정입니다.
  - 케라스의 레이어는 'keras.layers'에 정의되어 있으며, 여기서 필요한 레이어를 추가합니다.

```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
```

컨볼루션 신경망 레이어 이야기

- 한 번 쌓아보기
  - Sequential 모델을 하나 생성한 뒤 위에서 정의한 레이어를 차례차례 추가하면 컨볼루션 모델이 생성됩니다.

```
model = Sequential()

model.add(Conv2D(2, (3, 3), padding='same', activation='relu', input_shape=(8, 8, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(3, (2, 2), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

컨볼루션 신경망 레이어 이야기

- 결론
  - 본 강좌를 통해 컨볼루션 모델에서 사용되는 주요 레이어에 대해서 알아보았고, 레이어를 조합하여 컨볼루션 모델을 만들어봤습니다.
  - 다음 강좌에서는 이 모델을 이용하여 실제로 데이터셋을 학습시킨 후 분류가 제대로 되는 지 알아보겠습니다.

컨볼루션 신경망 모델 만들어보기

- 문제 정의하기
  - 좋은 예제와 그와 관련된 데이터셋도 공개된 것이 많이 있지만, 직접 문제를 정의하고 데이터를 만들어보는 것도 처럼 딥러닝을 접하시는 분들에게는 크게 도움이 될 것 같습니다.
  - 컨볼루션 신경망 모델에 적합한 문제는 이미지 기반의 분류입니다.
  - 따라서 우리는 직접 손으로 삼각형, 사각형, 원을 그려 이미지로 저장한 다음 이를 분류해보는 모델을 만들어보겠습니다.
  - 문제 형태와 입출력을 다음과 같이 정의해봅니다.
    - 문제 형태 : 다중 클래스 분류
    - 입력 : 손으로 그린 삼각형, 사각형, 원 이미지
    - 출력 : 삼각형, 사각형, 원일 확률을 나타내는 벡터

컨볼루션 신경망 모델 만들어보기

- 문제 정의하기
  - 가장 처음 필요한 패키지를 불러오고, 매번 실행 시마다 결과가 달라지지 않도록 랜덤 시드를 명시적으로 지정합니다.

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator

# 랜덤시드 고정시키기
np.random.seed(3)
```

컨볼루션 신경망 모델 만들어보기

- 데이터 준비하기
  - 손으로 그린 삼각형, 사각형, 원 이미지를 만들기 위해서는 여러 가지 방법이 있을 수 있겠네요.
  - 태블릿을 이용할 수도 있고, 종이에 그려서 사진으로 찍을 수도 있습니다. 저는 그림 그리는 툴을 이용해서 만들어봤습니다.
  - 이미지 사이즈는 24 x 24 정도로 해봤습니다.
  - 모양별로 20개 정도를 만들어서 15개를 훈련에 사용하고, 5개를 테스트에 사용해보겠습니다.
  - 이미지는 png나 jpg로 저장합니다.
  - 실제로 데이터셋이 어떻게 구성되어 있는 지 모른 체 튜토리얼을 따라하거나 예제 코드를 실행시키다보면 결과는 잘 나오지만 막상 실제 문제에 적용할 때 막막해질 때가 있습니다.
  - 간단한 예제로 직접 데이터셋을 만들어봄으로써 실제 문제에 접근할 때 시행착오를 줄이는 것이 중요합니다.

컨볼루션 신경망 모델 만들어보기

- 데이터 준비하기
  - 데이터 폴더는 다음과 같이 구성했습니다.
    - train
      - circle
      - rectangle
      - triangle
    - test
      - circle
      - rectangle
      - Triangle
  - 직접 그려보는 것을 권장하지만 아래 링크에서 다운로드를 받으실 수 있습니다.
    - https://github.com/jjin300/deep_learning/handwriting_shape.zip

컨볼루션 신경망 모델 만들어보기

- 데이터셋 생성하기
  - 케라스에서는 이미지 파일을 쉽게 학습시킬 수 있도록 ImageDataGenerator 클래스를 제공합니다.
  - ImageDataGenerator 클래스는 데이터 증강 (data augmentation)을 이용해 이미지 분류 성능을 향상 시키는 기법 중에 하나입니다.
  - 데이터 증강 (data augmentation)은 CNN등에 학습시키는데 데이터가 부족하니 기존의 이미지를 회전이동, 평행이동, 밝기조절, 확대/축소 등을 통해 재가공해서 새로운 이미지를 얻어내고 그 데이터를 train set에 추가하여 같이 학습을 시킵니다.

컨볼루션 신경망 모델 만들어보기

- 데이터셋 생성하기
  - 이것이 왜 유용할까요?
    - 첫 번째는 서로 다른 위치, 크기 등을 가진 이미지라도 잘 예측하게 해줍니다.
    - 예를 들어 제가 가진 말 이미지는 머리가 모두 사진의 왼쪽에 있다고 가정해 봅시다.
    - 이런 경우, 오른쪽에 머리가 있는 말 사진은 예측이 잘 안 될 수도 있습니다.
    - 그런데 Data augmentation을 통해 서로 다른 상황까지 고려해 train set에 넣어서 정확도를 더 높이는 것이죠.

    - 두번째는 기존 데이터의 라벨링을 따라가고 형상이 유지되기 때문에 생성된 이미지를 추가해서 쓴다고 하더라도 성능 저하가 없습니다.

컨볼루션 신경망 모델 만들어보기

- 데이터셋 생성하기
  - 먼저 ImageDataGenerator 클래스를 이용하여 객체를 생성한 뒤 flow_from_directory() 함수를 호출하여 제네레이터(generator)를 생성합니다.
  - flow_from_directory() 함수의 주요인자는 다음과 같습니다.
    - 첫번재 인자 : 이미지 경로를 지정합니다.
    - target_size : 패치 이미지 크기를 지정합니다. 폴더에 있는 원본 이미지 크기가 다르더라도 target_size에 지정된 크기로 자동 조절됩니다.
    - batch_size : 배치 크기를 지정합니다.
    - class_mode : 분류 방식에 대해서 지정합니다.
      - categorical : 2D one-hot 부호화된 라벨이 반환됩니다.
      - binary : 1D 이진 라벨이 반환됩니다.
      - sparse : 1D 정수 라벨이 반환됩니다.
      - None : 라벨이 반환되지 않습니다.

컨볼루션 신경망 모델 만들어보기

- 데이터셋 생성하기
  - 본 예제에서는 이미지 크기를 24 x 24로 하였으니 target_size도 (24, 24)로 셋팅하였습니다.
  - 훈련 데이터 수가 클래스당 15개이니 배치 크기를 3으로 지정하여 총 5번 배치를 수행하면 하나의 epoch가 수행될 수 있도록 하였습니다.
  - 다중 클래스 문제이므로 class_mode는 'categorical'로 지정하였습니다.
  - 그리고 제네레이터는 훈련용과 검증용으로 두 개를 만들었습니다.

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

train_generator = train_datagen.flow_from_directory(
        'warehouse/handwriting_shape/train',
        target_size=(24, 24),
        batch_size=3,
        class_mode='categorical')
```

컨볼루션 신경망 모델 만들어보기

- 데이터셋 생성하기
  - 본 예제에서는 이미지 크기를 24 x 24로 하였으니 target_size도 (24, 24)로 셋팅하였습니다.

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

train_generator = train_datagen.flow_from_directory(
        ' /content/gdrive/My Drive/data/handwriting_shape/train',
        target_size=(24, 24),
        batch_size=3,
        class_mode='categorical')

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = train_datagen.flow_from_directory(
        ' /content/gdrive/My Drive/data/handwriting_shape/test',
        target_size=(24, 24),
        batch_size=3,
        class_mode='categorical')
```

컨볼루션 신경망 모델 만들어보기

- 모델 구성하기
  - 영상 분류에 높은 성능을 보이고 있는 컨볼루션 신경망 모델을 구성해보겠습니다.
  - 각 레이어들은 이전 강좌에서 살펴보았으므로 크게 어려움없이 구성할 수 있습니다.
    - 컨볼루션 레이어 : 입력 이미지 크기 24 x 24, 입력 이미지 채널 3개, 필터 크기 3 x 3, 필터 수 32개, 활성화 함수 'relu'
    - 컨볼루션 레이어 : 필터 크기 3 x 3, 필터 수 64개, 활성화 함수 'relu'
    - 맥스풀링 레이어 : 풀 크기 2 x 2
    - 플래튼 레이어
    - 댄스 레이어 : 출력 뉴런 수 128개, 활성화 함수 'relu'
    - 댄스 레이어 : 출력 뉴런 수 3개, 활성화 함수 'softmax'

컨볼루션 신경망 모델 만들어보기

- 모델 구성하기
  - 영상 분류에 높은 성능을 보이고 있는 컨볼루션 신경망 모델을 구성해보겠습니다.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',  input_shape=(24,24,3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

컨볼루션 신경망 모델 만들어보기

- 모델 학습과정 설정하기
  - 모델을 정의했다면 모델을 손실함수와 최적화 알고리즘으로 엮어봅니다.
    - loss : 현재 가중치 세트를 평가하는 데 사용한 손실 함수 입니다. 다중 클래스 문제이므로 'categorical_crossentropy'으로 지정합니다.
    - optimizer : 최적의 가중치를 검색하는 데 사용되는 최적화 알고리즘으로 효율적인 경사 하강법 알고리즘 중 하나인 'adam'을 사용합니다.
    - metrics : 평가 척도를 나타내며 분류 문제에서는 일반적으로 'accuracy'으로 지정합니다.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

컨볼루션 신경망 모델 만들어보기

- 모델 학습시키기
  - 케라스에서는 모델을 학습시킬 때 주로 fit() 함수를 사용하지만 제네레이터로 생성된 배치로 학습시킬 경우에는 fit_generator() 함수를 사용합니다.
  - 본 예제에서는 ImageDataGenerator라는 제네레이터로 이미지를 담고 있는 배치로 학습시키기 때문에 fit_generator() 함수를 사용하겠습니다.
    - 첫번째 인자 : 훈련데이터셋을 제공할 제네레이터를 지정합니다. 본 예제에서는 앞서 생성한 train_generator으로 지정합니다.
    - steps_per_epoch : 한 epoch에 사용한 스텝 수를 지정합니다. 총 45개의 훈련 샘플이 있고 배치사이즈가 3이므로 15 스텝으로 지정합니다.
    - epochs : 전체 훈련 데이터셋에 대해 학습 반복 횟수를 지정합니다. 100번을 반복적으로 학습시켜 보겠습니다.
    - validation_data : 검증데이터셋을 제공할 제네레이터를 지정합니다. 본 예제에서는 앞서 생성한 test_generator으로 지정합니다.
    - validation_steps : 한 epoch 종료 시 마다 검증할 때 사용되는 검증 스텝 수를 지정합니다. 총 15개의 검증 샘플이 있고 배치사이즈가 3이므로 5 스텝으로 지정합니다.

컨볼루션 신경망 모델 만들어보기

- 모델 학습시키기
  - 케라스에서는 모델을 학습시킬 때 주로 fit() 함수를 사용하지만 제네레이터로 생성된 배치로 학습시킬 경우에는 fit_generator() 함수를 사용합니다.

```
model.fit_generator(
        train_generator,
        steps_per_epoch=15,
        epochs=50,
        validation_data=test_generator,
        validation_steps=5)
```

컨볼루션 신경망 모델 만들어보기

- 모델 평가하기
  - 학습한 모델을 평가해봅니다.
  - 제네레이터에서 제공되는 샘플로 평가할 때는 evaluate_generator 함수를 사용합니다.

```
print("-- Evaluate --")
scores = model.evaluate_generator(test_generator, steps=5)
print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))
```

컨볼루션 신경망 모델 만들어보기

- 모델 사용하기
  - 모델 사용 시에 제네레이터에서 제공되는 샘플을 입력할 때는 predict_generator 함수를 사용합니다.
  - 예측 결과는 클래스별 확률 벡터로 출력되며, 클래스에 해당하는 열을 알기 위해서는 제네레이터의 'class_indices'를 출력하면 해당 열의 클래스명을 알려줍니다.

```
print("-- Predict --")
output = model.predict_generator(test_generator, steps=5)
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})
print(test_generator.class_indices)
print(output)
```

컨볼루션 신경망 모델 만들어보기

- 전체소스

```
# 0. 사용할 패키지 불러오기
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator

# 랜덤시드 고정시키기
np.random.seed(3)
```

컨볼루션 신경망 모델 만들어보기

- 전체소스

```
# 1. 데이터 생성하기
train_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        ‘ ./datasets/handwriting_shape/train',
        target_size=(24, 24),
        batch_size=3,
        class_mode='categorical')

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        ‘ ./datasets/handwriting_shape/test',
        target_size=(24, 24),
        batch_size=3,
        class_mode='categorical')
```

컨볼루션 신경망 모델 만들어보기

- 전체소스

```
# 2. 모델 구성하기
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(24,24,3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(3, activation='softmax'))

# 3. 모델 학습과정 설정하기
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

컨볼루션 신경망 모델 만들어보기

- 전체소스

```
# 4. 모델 학습시키기
model.fit_generator(
        train_generator,
        steps_per_epoch=15,
        epochs=50,
        validation_data=test_generator,
        validation_steps=5)

# 5. 모델 평가하기
print("-- Evaluate --")
scores = model.evaluate_generator(test_generator, steps=5)
print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))

# 6. 모델 사용하기
print("-- Predict --")
output = model.predict_generator(test_generator, steps=5)
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})
print(test_generator.class_indices)
print(output)
```

컨볼루션 신경망 모델 만들어보기

- 요약
  - 이미지 분류 문제에 높은 성능을 보이고 있는 컨볼루션 신경망 모델을 이용하여 직접 만든 데이터셋으로 학습 및 평가를 해보았습니다.
  - 학습 결과는 좋게 나왔지만 이 모델은 한 사람이 그린 것에 대해서만 학습이 되어 있어 다른 사람에 그린 모양은 분류를 잘 하지 못합니다.
  - 이를 해결하기 위한 방안으로 '데이터 부풀리기' 기법이 있습니다.
  - 참고로 실제 문제에 적용하기 전에 데이터셋을 직접 만들어보거나 좀 더 쉬운 문제로 추상화해서 프로토타이핑 하는 것을 권장합니다.

Fashion MNIST 데이터세트에 적용하기

- tf.keras 에서 Fashion_MNIST 데이터세트를 불러오고 정규화하는 부분입니다.

```
import tensorflow as tf
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_X, train_Y), (test_X, test_Y) = fashion_mnist.load_data()
train_X = train_X / 255.0
test_X = test_X / 255.0
```

Fashion MNIST 데이터세트에 적용하기

- 이것을 Conv2D 레이어로 컨볼루션 연산을 해야 합니다.
- 이미지는 보통 채널을 가지고 있고(컬러 이미지는 RGB의 3 채널, 흑백 이미지는 1 채널), Conv2D 레이어는 채널을 가진 형태의 데이터를 받도록 기본적으로 설정돼 있기 때문에 다음 코드에서는 채널을 갖도록 데이터의 Shape을 바꿉니다.

```
import tensorflow as tf
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_X, train_Y), (test_X, test_Y) = fashion_mnist.load_data()
train_X = train_X / 255.0
test_X = test_X / 255.0
```

- Fashion_MNIST 데이터를 구성하는 흑백 이미지는 1개의 채널을 갖기 때문에 reshape() 함수를 사용해 데이터의 가장 뒤쪽에 채널 차원을 추가합니다.
- 이 작업으로 데이터 수는 달라지지 않습니다-
- 60000 x 28 x 28과 60000 x 28 x 28 x 1 이 같은 값인 것처럼 말입니다.

Fashion MNIST 데이터세트에 적용하기

- 데이터를 확인하기 위해 matplotlib.pyplot을 사용해 그래프를 그려 볼 수 있습니다.

```
import matplotlib .pyplot as plt
# 전체 그래프의 크기를 width=10, height=10으로 지정합니다
plt.figure(figsize=(10, 10))
for c in range(16) :
    # 4행 4열로 지정한 그리드에서 (+1 번째의 칸에 그래프를 그립니다 1-16 번째 칸을 채우게 됩니다
    plt.subplot (4, 4, c+1)
    plt.imshow(train_X[c].reshape(28, 28), cmap='gray')

plt.show()

# 훈련 데이터의 첫 번째 - 16 번째 까지의 라벨을 프린트합니다 .
print(train_Y[:16])
```

Fashion MNIST 데이터세트에 적용하기

- 그래프를 그리기 위한 데이터는 2 차원이어야 하기 때문에 plt.imshow(train_X[c].reshape(28 ,28), cmap='gray') 에서 각 데이터를 대상으로 reshape() 함수를 취해 3차원 데이터를 다시 2차원 데이터로 변환합니다.
- 각 이미지는 plt.subplot() 함수에서 지정되는 그리드(grid)의 각 칸에 위에서 아래, 왼쪽에서 오른쪽 순서대로 그려집니다.
- 데이터의 범주 중 9 번은 신발, 0번은 티셔츠/상의, 3번은 드레스입 니다.
- 출력 이미지 첫 번째 행의 4개 이미지가 9, 0, 0, 3의 라벨로 잘 분류돼 있는 것을 확인할 수 있습니다.

라벨	범주
0	티셔츠/상의
1	바지
2	스웨터
3	드레스
4	코트
5	샌들
6	셔츠
7	운동화
8	가방
9	부츠

Fashion MNIST 데이터세트에 적용하기

- 이제 모델을 생성할 차례입니다.
- 먼저 비교를 위해 풀링 레이어 없이 컨볼루션 레이어만 사용한 모델을 정의해 보겠습니다.

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28, 28, 1), kernel_size=(3, 3), filters=16),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=32 ),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=64 ),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation= 'relu'),
    tf.keras.layers.Dense(units=10 , activation= 'softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
    loss= 'sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.summary()
```

Fashion MNIST 데이터세트에 적용하기

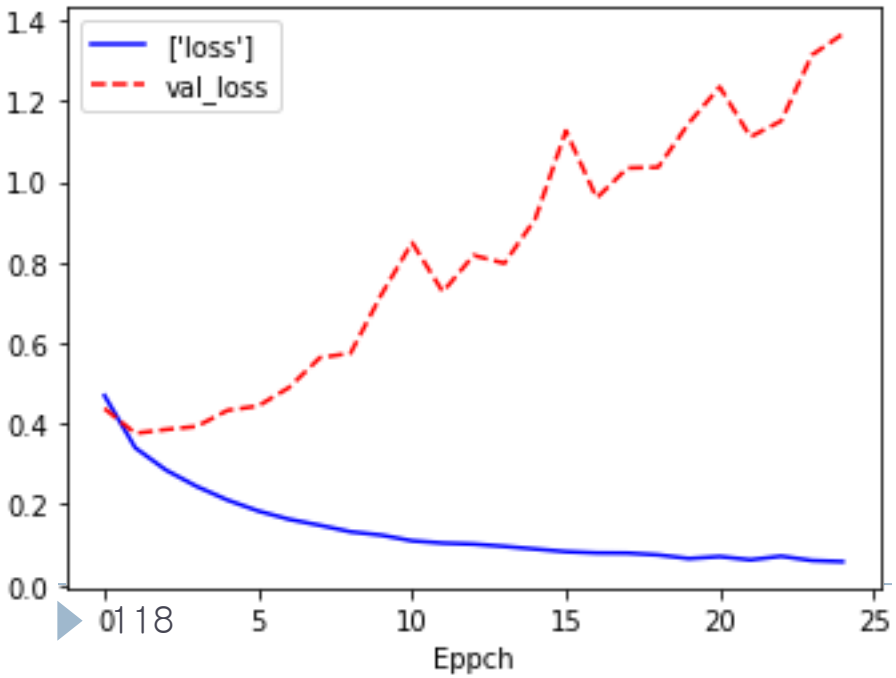
- 모델에는 총 3개의 Conv2D 레이어를 사용했고 그 중 첫 레이어의 input_shape 은 (28, 28, 1) 로 입력 이미지의 높이, 너비, 채널 수를 정의하고 있습니다.
- 필터의 수는 16, 32, 64로 뒤로 갈수록 2 배씩 늘렸습니다.
- Flatten 레이어로 다차원 데이터를 1 차원으로 정렬한 다음에 2 개의 Dense 레이어를 사용해 분류기를 만들었습니다.
- 그럼 이 모델의 퍼포먼스를 확인해보겠습니다.

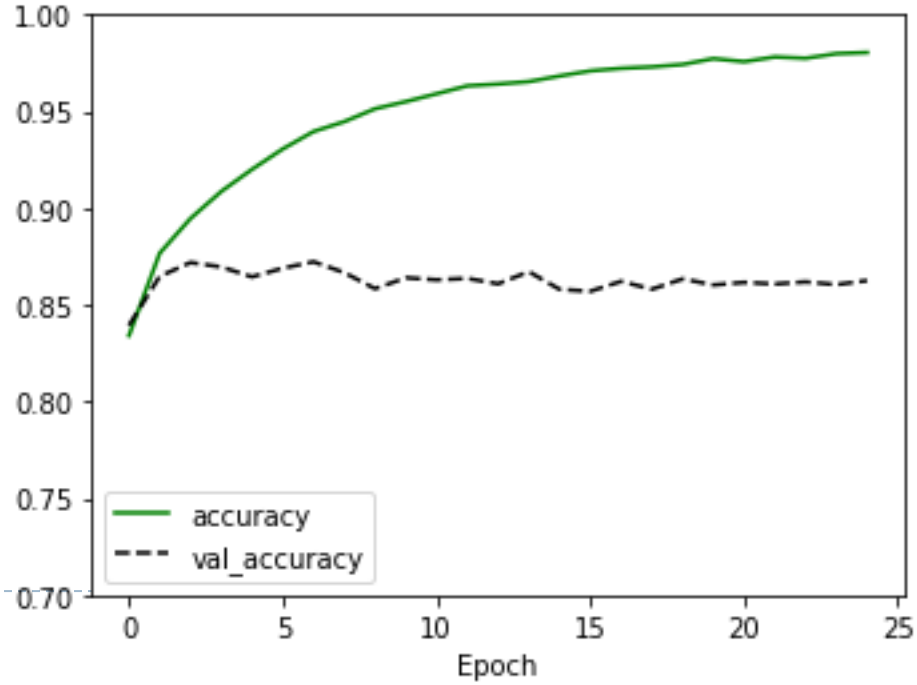
Fashion MNIST 데이터세트에 적용하기

```
history = model.fit(train_X , train_Y, epochs=25, validation_split=0.25)

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('Eppch')
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(0.7, 1)
plt.legend()

plt.show()

model.evaluate(test_X, test_Y, verbose=0)
```

Fashion MNIST 데이터세트에 적용하기

- 왼쪽 그래프를 확인해보면 loss는 감소하고 val_loss는 증가하는 전형적인 과적합의 형태를 나타냅니다.
- 오른쪽 그래프에서는 훈련 데이터에 대한 모델의 정확도인 accuracy가 빠르게 증가하는 데에 비해서 검증 데이터에 대한 정확도인 val_accuracy는 학습이 진행될수록 오히려 감소합니다.
- 마지막에 model.evaluate() 함수로 계산되는 결과 중 첫 번째가 테스트 데이터의 loss이고 두 번째가 테스트 데이터의 accuracy 입니다.





Fashion MNIST 데이터세트에 적용하기

- 이를 어떻게 개선할 수 있을까요?
- 일단 풀링 레이어와 드롭아웃 레이어를 모두 사용해보겠습니다.

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3 ,3), filters=32),
    tf.keras.layers.MaxPool2D(strides=(2, 2)),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=64),
    tf.keras.layers.MaxPool2D(strides=(2, 2)),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=128),
    tf.keras.layers.Flatten( ),
    tf.keras.layers.Dense(units=128, activation= 'relu'),
    tf.keras.layers.Dropout(rate=0.3 ),
    tf.keras.layers.Dense(units=18, activation= 'softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.summary()
```

Fashion MNIST 데이터세트에 적용하기

- summary() 함수로 레이어 구조를 확인해보면 총 파라미터 개수가 앞의 코드의 400만에 가까운 숫자에 비해 24만 정도로. 약 6%로 줄어들었습니다.
- 이는 풀링 레이어가 이미지의 크기를 줄여주고 있기 때문에 Flatten 레이어에 들어온 파라미터 수가 1,152로 이전 코드의 30,976 에 비해 감소했기 때문입니다.
- 두 모델에서 가장 많은 파라미터가 있는 레이어는 Flatten 레이어 다음의 첫 번째 Dense 레이어이기 때문에 이 레이어에 넘어오는 파라미터 수가 적을수록 전체 파라미터 수도 적어지는 것입니다.
- Dense 레이어 사이에는 드롭아웃 레이어도 사용됐습니다.

Fashion MNIST 데이터세트에 적용하기

- 풀링 레이어와 드롭아웃 레이어는 모두 과적합을 줄이는 데 기여하게 됩니다.
- 실제로 그렇게 되는지 네트워크 학습을 통해 확인해보겠습니다.

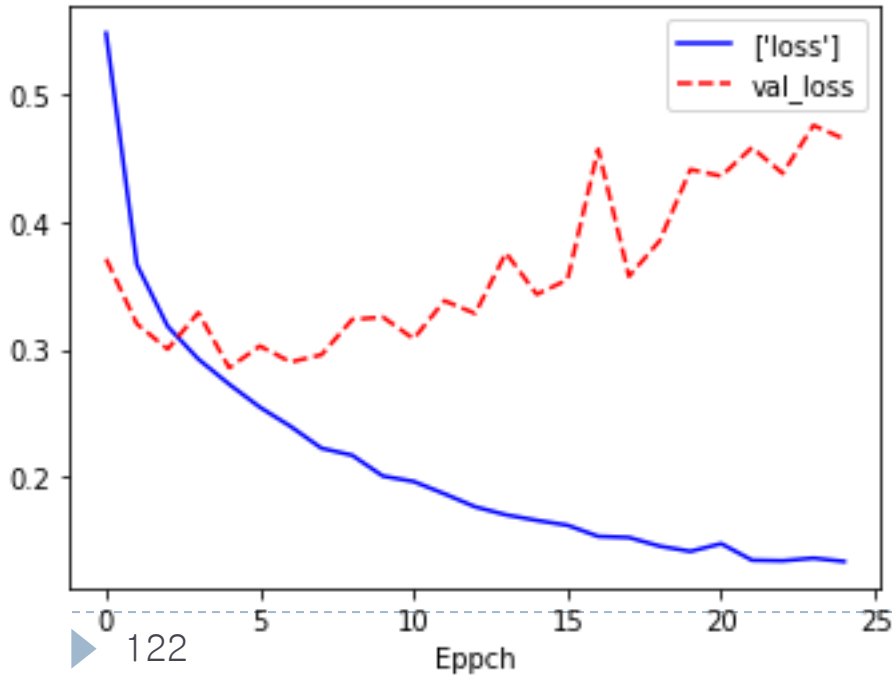
```
history = model.fit(train_X, train_Y , epochs=25 , validation_split =0.25 )
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

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('Eppch')
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(0.7, 1)
plt.legend()

plt.show()
```

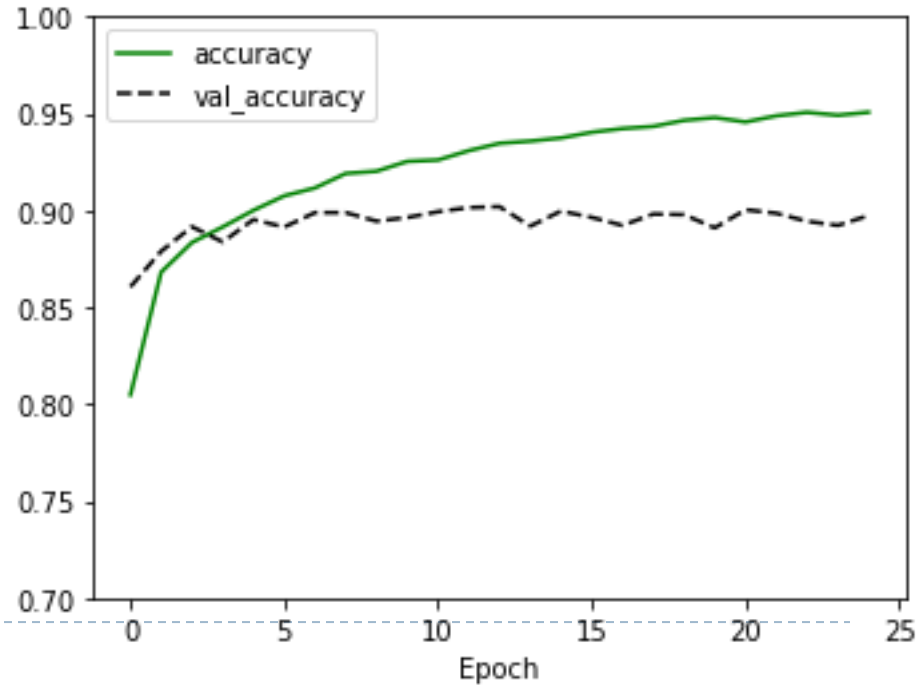


```
model.evaluate(test_X, test_Y, verbose=0)
```

Fashion MNIST 데이터세트에 적용하기

- val_ loss 가 여전히 증가하고 있지만 val_accuracy는 일정한 수준에 머무르고 있습니다.
- 테스트 데이터에 대한 분류 성적은 89.09%가 나옵니다.
- 풀링 레이어와 드롭아웃 레이어를 쓰지 않았을 때보다 개선된 수치입니다.
- 하지만 이보다 더 좋은 성과를 낼 수 있을 것 같습니다.
- Fashion_MNIST의 공식 깃허브 저장소에는 테스트 데이터 분류 성적에서 95% 이상을 달성한 몇몇 방법들이 기록돼 있습니다.
- 이 방법들을 참고해서 현재의 분류 성적을 90% 이상으로 끌어올려보겠습니다.





퍼포먼스 높이기

- 컨볼루션 신경망에서 퍼포먼스를 높이는 데는 여러 가지 방법이 있지만 그중 대표적이면서 쉬운 두 가지 방법은'더 많은 레이어 쌓기'와 '이미지 보강(Image Augmemation)' 기법입니다.
- 더 많은 레이어 쌓기
  - 컨볼루션 신경망의 역사, 더 나아가 딥러닝의 역사는 더 깊은 신경망을 쌓기 위한 노력이라고 해도 과언이 아닙니다.
  - 딥러닝에서 네트워크 구조를 깊게 쌓는 것이 가능해진 후 딥러닝의 발전을 이끈 컨볼루션 신경망에서는 컨볼루션 레이어가 중첩된 더 깊은 구조가 계속해서 나타났고, 그럴 때마다 이전 구조의 퍼포먼스를 크게 개선했습니다.

퍼포먼스 높이기

- 더 많은 레이어 쌓기
  - 앞 절에서 VGGNet 의 스타일로 구성한 컨볼루션 신경망을 사용해 Fashion_MNIST 데이터를 분류하는 모델을 정의했습니다.
  - VGG는 단순한 구조이면서도 성능이 괜찮기 때문에 지금도 이미지의 특징 추출을 위한 네트워크에서 많이 쓰이고 있습니다.
  - 유명한 Style Transfer 논문에서도 VGGNet(VGG-19)을 사용했습니다.

퍼포먼스 높이기

- 더 많은 레이어 쌓기
  - VGGNet 스타일의 Fashion_MNIST 분류를 위한 컨볼루션 신경망 모델 정의

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28, 28, 1), kernel_size=(3, 3), filters=32, padding= 'same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=64, padding= 'same', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=128, padding= 'same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=256, padding='valid', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size= (2, 2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=512, activation= 'relu' ),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=256, activation= 'relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=10 , activation='softmax' )
])
```

퍼포먼스 높이기

- 더 많은 레이어 쌓기
  - VGGNet 스타일의 Fashion_MNIST 분류를 위한 컨볼루션 신경망 모델 정의

```
model.compile(optimizer=tf.keras.optimizers.Adam(),
    loss='sparse_categorical_crossentropy',
    metrics=[ 'accuracy' ])

model.summary()
```

  - VGGNet은 여러 개의 구조로 실험했는데 그중 19개 의 레이어가 겹쳐진 VGG-19가 제일 깊은 구조입니다.
  - VGG-19는 특정 추출기의 초반에 컨볼루션 레이어를 2개 겹친 뒤 풀링 레이어 1개를 사용하는 패턴을 2 차례, 그 후 컨볼루션 레이어를 4개 겹친 뒤 풀링 레이어 1개를 사용하는 패턴을 3차례 반복합니다.

퍼포먼스 높이기

- 더 많은 레이어 쌓기
  - 여기서는 대상 이미지가 작기도 하고 연산 능력의 한계도 있어서 컨볼루션 레이어를 2 개 겹치고 풀링 레이어를 1 개 사용하는 패턴을 2 차례 반복했습니다.
  - 그리고 풀링 레이어의 다음에 드롭아웃 레이어를 위치시켜서 과적합을 방지했고, Flatten 레이어 다음에 이어지는 3개의 Dense 레이어 사이에도 드롭아웃 레이어를 배치했습니다.
  - VGGNet처럼 컨볼루션 레이어와 Dense 레이어의 개수만 세면 VGG-7 정도가 되겠습니다.
  - 오리지널 VGG-19보다는 깊이가 얕지만 총 파라미터 개수는 520만 개로 적지 않습니다.
  - 앞 절의 24 만 개보다는 약 20 배 이상 증가한 숫자입니다.

퍼포먼스 높이기

- 더 많은 레이어 쌓기
  - 그럼 이 모델의 성능은 어떤지 학습으로 알아보겠습니다.

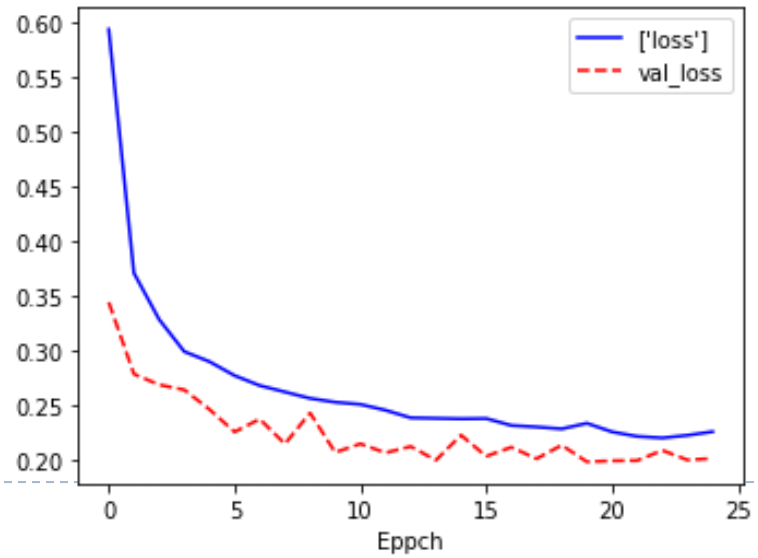
```
history = model.fit(train_X, train_Y , epochs=25 , validation_split =0.25 )
plt.figure(figsize=(12, 4))

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('Eppch')
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(0.7, 1)
plt.legend()

plt.show()
```

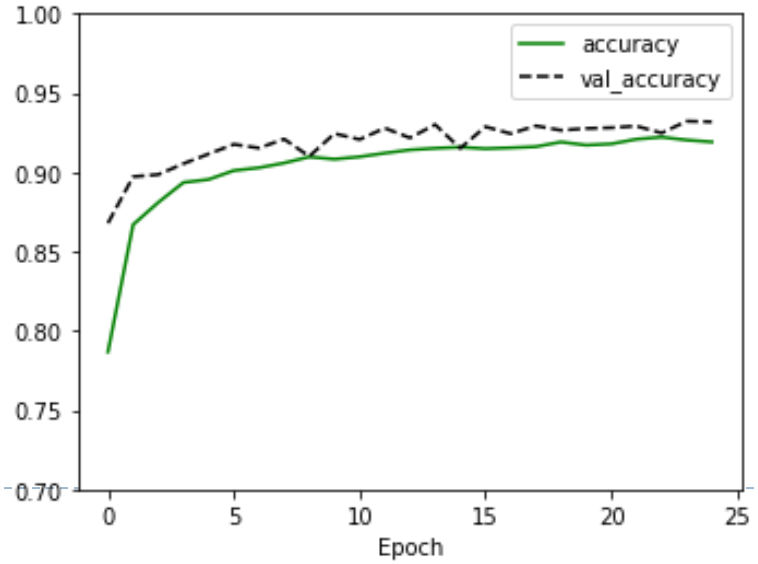


```
model.evaluate(test_X, test_Y, verbose=0)
```

퍼포먼스 높이기

- 더 많은 레이어 쌓기
  - 드디어 va1_loss가 잘 증가하지 않는 훌륭한 그래프를 얻었습니다.
  - 테스트 데이터에 대한 분류 성적도 92.52 %로 지금까지 거둔 성적 가운데 제일 우수합니다.
  - 모델은 아직 과적합되지 않았기 때문에 에포크 수를 늘려서 좀더 돌려볼 수도 있을 것 같습니다.
  - 이로써 간단한 네트워크 구조 변경만으로도 Fashion_MNTST 데이터의 분류 성능을 어느 정도 올릴 수 있다는 점을 확인했습니다 .





퍼포먼스 높이기

- 이미지 보강
  - 이미지 보강(Image Augmentation)은 훈련 데이터에 없는 이미지를 새롭게 만들어내서 훈련 데이터를 보강하는 것입니다.
  - 이때 새로운 이미지는 훈련 데이터의 이미지를 원본으로 삼고 일정한 변형을 가해서 만들어집니다.
  - 예를 들어, 다음과 같은 신발 이미지가 훈련 데이터에 있을 때 신발코가 왼쪽을 향하는 이미지만 훈련 데이터에 있고 테스트 데이터에는 신발코가 오른쪽을 향하는 이미지가 있을 경우, 컨볼루션 신경망은 테스트 데이터에서 새롭게 나오는 이미지에 대해 좋은 퍼포먼스를 내지 못합니다.
  - 이때 이미지를 가로로 뒤집어서(horiιontalf1ip) 신발코가 오른쪽을 향하는 이미지도 만들고, 약간 회전시키거나(rotate), 기울이거나(shear), 일부 확대하거나(zoom), 평행 이동시켜서(shift) 다양한 이미지를 만들어 내서 훈련 데이터의 표현력을 더 좋게 만드는 것입니다.

퍼포먼스 높이기

- 이미지 보강
  - tf.keras 에는 이러한 이미지 보강 작업을 쉽게 해주는 ImageDataGenerator가 있습니다.
  - 다음 코드는 이를 활용해 훈련 데이터의 첫 번째 이미지를 변형시킵니다.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

image_generator = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.10,
    shear_range=0.5,
    width_shift_range=0.10,
    height_shift_range=0.10,
    horizontal_flip=True,
    vertical_flip=False)

augment_size = 100
x_augmented = image_generator.flow(np.tile(train_X[0].reshape(28*28), 100).reshape(-1, 28, 28, 1), np.zeros(augment_size), batch_size=augment_size, shuffle=False).next()[0]
```

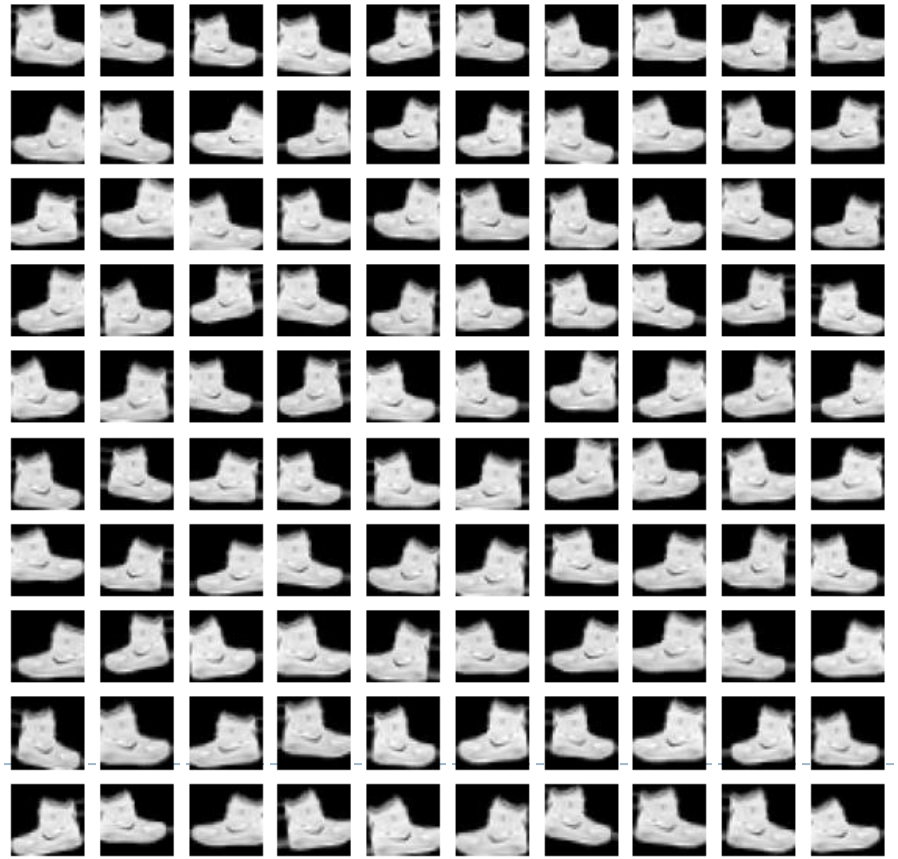
퍼포먼스 높이기

- 이미지 보강
  - tf.keras 에는 이러한 이미지 보강 작업을 쉽게 해주는 ImageDataGenerator가 있습니다.
  - 다음 코드는 이를 활용해 훈련 데이터의 첫 번째 이미지를 변형시킵니다.

```
# 새롭게 생성된 이미지 표시
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10 ))
for c in range (100) :
    plt.subplot(10, 10, c+1 )
    plt.axis( 'off')
    plt.imshow(x_augmented[c].reshape(28, 28), cmap= 'gray')

plt.show()
```

퍼포먼스 높이기

- 이미지 보강
  - 그림의 신발 이미지가 여러 가지 형태로 조금씩 바뀐 것을 확인할 수 있습니다.
  - 다양한 변형이 가해지기 때문에 각 이미지는 비슷하면서도 서로 조금씩 다릅니다.



퍼포먼스 높이기

- 이미지 보강
  - ImageDataGenerator의 주요 인수들은 rotation_range, zoom_range, shear_range 등 입니다.
  - 가로축으로 이미지를 뒤집는 horizontal_flip은 사용하지만 세로축으로 뒤집는 vertical_flip은 사용하지 않습니다.
  - Fashion_MNIST에는 보통 이미지가 위아래로 반듯하게 놓여 있기 때문에 vertical_flip 옵션을 True로 설정하면 대비하지 않아도 될 경우(이미지가 뒤집혀 있는 경우) 에 대해서도 대비하게 되어 퍼포먼스가 떨어집니다.

퍼포먼스 높이기

- 이미지 보강
  - flow ( ) 함수는 실제로 보강된 이미지를 생성합니다.
  - 이 함수는 Iterator라는 객체를 만드는데, 이 객체에서는 값을 순차적으로 꺼낼 수 있습니다.
  - 값을 꺼내는 방법은 next( ) 함수를 사용하는 것 입니다.
  - 한번에 생성할 이미지의 양인 batch_size를 위에서 설정한 augment_size와 같은 100으로 설정했기 때문에 next( ) 함수로 꺼내는 이미지는 100장이 됩니다.
  - 나머지 부분은 matplotlib.pyplot으로 생성된 보강 이미지를 그래프로 그려주는 부분입니다.

퍼포먼스 높이기

- 이미지 보강
  - 그럼 실제로 훈련 데이터 이미지를 보강하기 위해 다량의 이미지를 생성하고 학습을 위해 훈련 데이터에 추가해보겠습니다.

```
image_generator = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.10,
    shear_range=0.5,
    width_shift_range=0.10,
    height_shift_range=0.10,
    horizontal_flip=True,
    vertical_flip=False)

augment_size = 30000
```

퍼포먼스 높이기

- 이미지 보강
  - 그럼 실제로 훈련 데이터 이미지를 보강하기 위해 다량의 이미지를 생성하고 학습을 위해 훈련 데이터에 추가해보겠습니다.

```
randidx = np.random.randint(train_X.shape[0], size=augment_size)
x_augmented = train_X[randidx].copy()
y_augmented = train_Y[randidx].copy ()
x_augmented = image_generator.flow (x_augmented, np.zeros(augment_size),
batch_size=augment_size, shuffle=False ).next()[0]

# 원래 데이터인 x _train 에 이미지 보강된 x_augmented 를 추가합니다.
train_X = np.concatenate((train_X, x_augmented))
train_Y = np.concatenate((train_Y, y_augmented))
print (train_X.shape)
```

퍼포먼스 높이기

- 이미지 보강
  - 훈련 데이터의 50%인 30,000개의 이미지를 추가하기 위해 augment_size = 30000으로 설정하고, 이미지를 변형할 원본 이미지를 찾기 위해 np.random.randint () 함수를 써서 0 ~ 59,999 범위의 정수 중에서 30,000 개의 정수를 뽑았습니다.
  - 이때 뽑히는 정수는 중복될 수 있습니다.
  - 중복되지 않는 것을 원한다면 np.random.randint() 대신 np.random.choice() 함수를 사용하고 replace 인수를 False로 설정하면 됩니다.
  - randidx는 [2, 25432 , 425, ...]와 같은 정수로 구성된 넘파이 array 이고, 이 array를 이용해 train_X에서 각 array의 원소가 가리키는 이미지를 train_X [randidx] 로 한번에 선택할 수 있습니다.
  - 이렇게 선택한 데이터는 원본 데이터를 참조하는 형태이기 때문에 원본 데이터에 영향을 주지 않기 위해 copy ( ) 함수로 완전하게 복사본을 만들어줍니다.
  - 그 다음에는 ImageDataGenerator의 flow( ) 함수로 30,000 개의 새로운 이미지를 생성합니다.

퍼포먼스 높이기

- 이미지 보강
  - 마지막으로 np.concatenate() 함수로 훈련 데이터에 보강 이미지를 추가합니다.
  - 최종 출력에서 train_X.shape의 첫 번째 차원 수는 90,000 이 되어 정상적으로 이미지가 추가한 것을 확인할 수 있습니다.
  - 그럼 이제 VGGNet 스타일의 네트워크에 ImageDataGenerator로 보강된 훈련 데이터를 학습시켜 보겠습니다.

퍼포먼스 높이기

- 이미지 보강
  - VGGNet style 네트워크 + 이미지 보강학습

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28, 28, 1), kernel_size=(3, 3), filters=32, padding=
'same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=64, padding= 'same', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=128, padding= 'same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=256, padding='valid', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size= (2, 2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=512, activation= 'relu' ),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=256, activation= 'relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=10 , activation='softmax' )
])
```

퍼포먼스 높이기

- 이미지 보강
  - VGGNet style 네트워크 + 이미지 보강학습

```
model.compile(optimizer=tf.keras.optimizers.Adam(),
    loss='sparse_categorical_crossentropy',
    metrics=[ 'accuracy' ])

history = model.fit(train_X, train_Y , epochs=25 , validation_split =0.25 )
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('Eppch')
plt.legend()
```

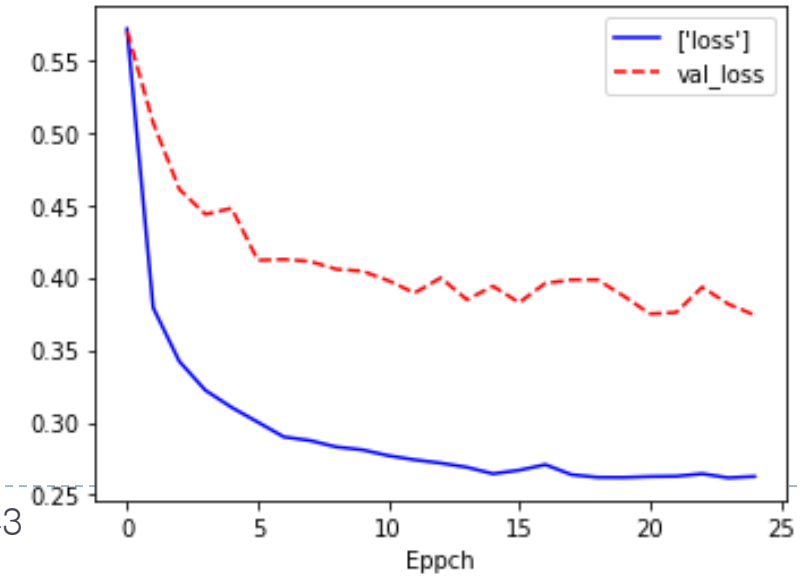
퍼포먼스 높이기

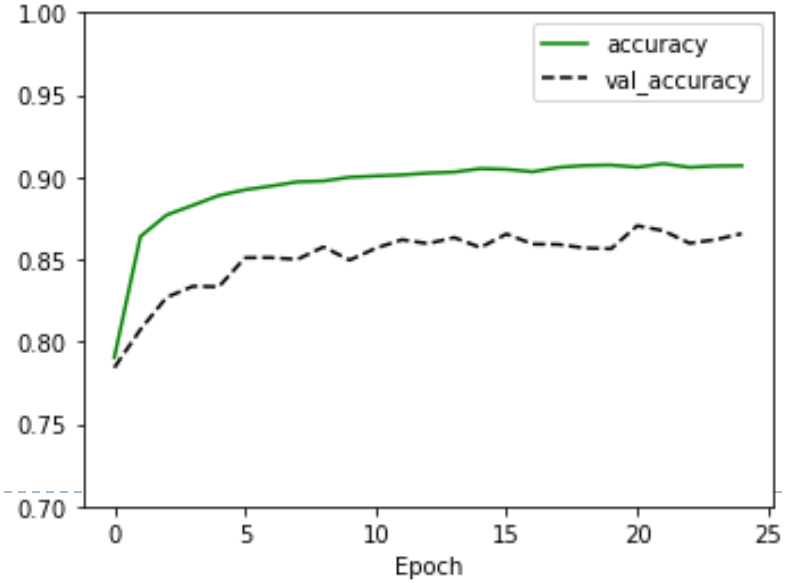
- 이미지 보강
  - VGGNet style 네트워크 + 이미지 보강학습

```
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(0.7, 1)
plt.legend()

plt.show()

model.evaluate(test_X, test_Y, verbose=0)
```

퍼포먼스 높이기

- 이미지 보강
  - 테스트 데이터에 대한 분류 성적은 92.88%로 92 .52 %보다 소폭 증가했습니다.
  - val_accuracy도 증가하는 추세를 보이고 있지 않아서 모델이 아직 과적합되지 않은 것으로 판단되며, 조금 더 학습시키면 성적이 더욱 잘 나올 것으로 기대됩니다.
  - 이렇게 해서 더 많은 레이어를 쌓는 것과 이미지 보강 기법이 컨볼루션 신경망의 분류 성적을 개선할 수 있다는 점을 배웠습니다





최신 CNN 아키텍처

- LeNet(1998)
  - 최초의 CNN은 얀 르쿤이 1998년에 개발한 LeNet입니다.
  - 르쿤은 CNN이 이미지 인식, 특히 필기 인식 분야에 강하다는 점을 최초로 증명했습니다.
  - 하지만 2000년대에는 르쿤의 뒤를 이은 연구자가 거의 없었고, CNN과 인공 지능 분야에도 획기적인 성과는 없었습니다.

- AlexNet(2012)
  - AlexNet은 알렉스 크리체프스키와 연구진이 개발해 2012년 ILSVRC 우승의 주역이 됐습니다.
  - AlexNet은 LeNet과 동일한 원리로 만들었지만 더 깊은 신경망을 사용했다는 점이 다릅니다.
  - AlexNet의 매개변수는 LeNet보다 1,000 배 이상 많은 6,000만 개에 달합니다.

최신 CNN 아키텍처

- VGG16(2014)
  - VGG16은 옥스포드 대학 VGG(Virtual Geometry Group)가 개발한 혁신적인 아키텍처로 필터 크기를 가급적 키우는 방식을 버리고 최초로 가로 3, 세로 3짜리 컨볼루션 필터를 사용했습니다.
  - VGG16은 2014 년 ILSVRC 이미지 인식 분야에서 2 위를 기록했습니다.
  - VGG16의 단점은 훈련시킬 매개변수가 훨씬 많아 훈련 시간이 오래 걸린다는 데 있습니다.

최신 CNN 아키텍처

- Inception(2014)
  - 구글이 개발한 Inception은 2014년 ILSVRC에서 우승을 차지했습니다.
  - Inception의 설계 원칙은 정확한 예측을 효율적으로 제공하는 것입니다.
  - 구글은 여러 서버에서 실시간으로 훈련하고 배포할 수 있는 CNN을 필요로 했고, 정확도를 유지하면서 훈련 시간을 크게 개선한 Inception 모델을 개발했습니다.
  - 그러나 Inception은 적은 매개 변수로도 VGG16 보다 높은 정확도를 달성해 2014년 ILSVRC에서 우승했습니다.
  - Inception은 그 후로도 발전을 거듭해 최근에는 네 번째 버전(Inception-v4)이 나왔습니다.

최신 CNN 아키텍처

- ResNet(2015)
  - ResNet(Residual Neural Network)은 키이밍 히와 연구진이 2015년 ILSVRC에 공개한 아키텍처입니다.
  - ResNet의 특징은 잔차 블록 기술로, 매개변수 개수를 적절히 유지하면서 신경망 깊이를 늘릴 수 있습니다.

Q&A