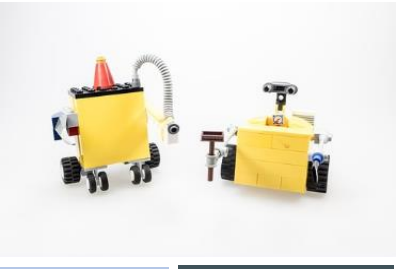
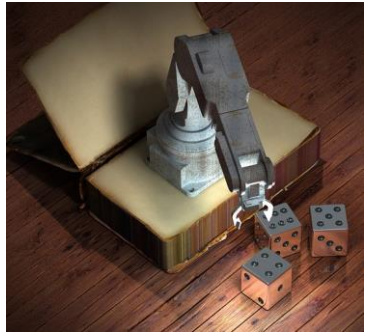
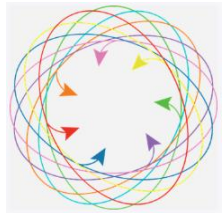
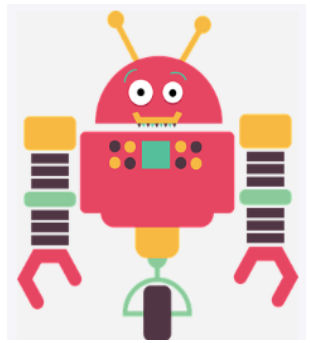
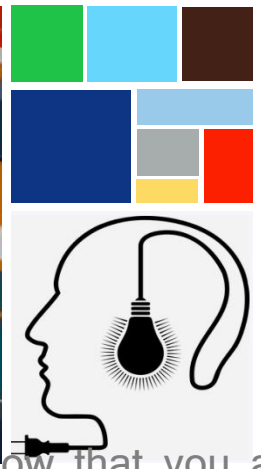








In the future, an AI agent will know that you are at work and have ten minutes free, and then help you accomplish something that is high on your to-do list.

딥러닝 알아보기

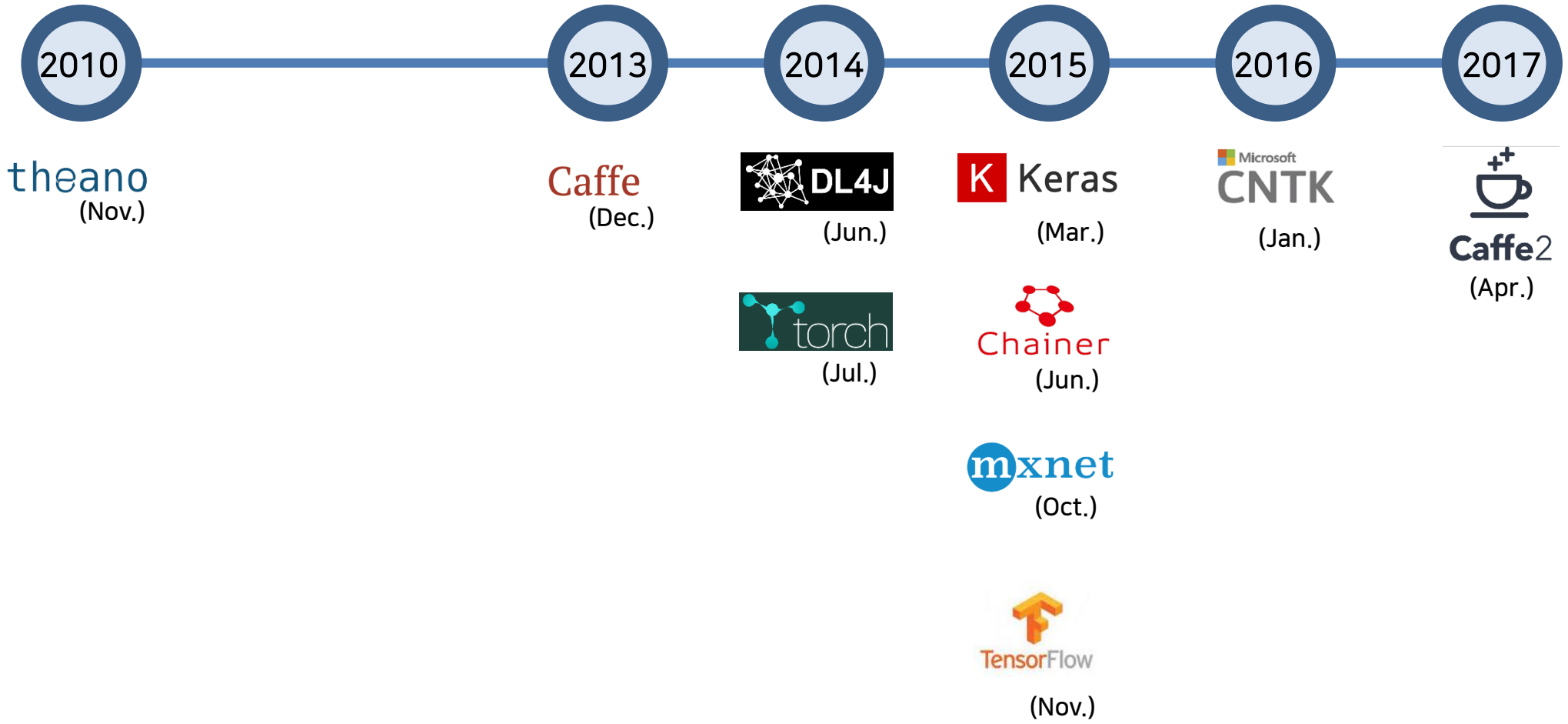



PART1. 딥러닝 대한 이해

Enjoy your possibility

Tensorflow-keras 로 딥러닝 모델 구성해보기

딥러닝 프레임워크 Timeline



딥러닝 프레임워크 비교

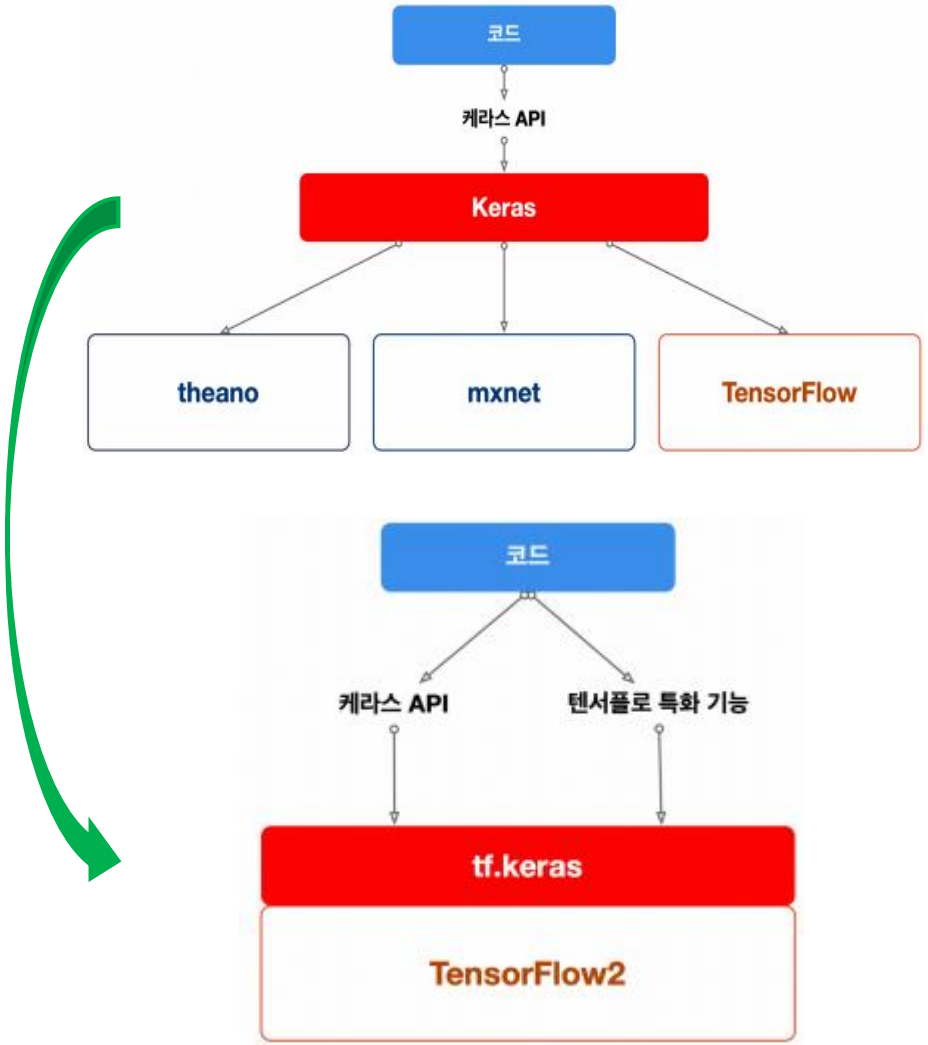
기준	PyTorch	TensorFlow	Keras
주요 사용 분야	연구, 학계, 실험적 모델 개발	산업 및 대규모 배포 환경	프로토타입 개발 및 교육
성장률	연구 분야에서 25% 성장	최근 산업 응용에서 1.4% 성장	사용 비중 약 4.8% 감소
강점	- 동적 그래프 제공 - 빠른 디버깅 및 실험에 유리	- 대규모 학습 및 배포에 최적화 - 다양한 배포 도구 제공	- 간단한 코드로 빠른 개발 가능 - 교육에 적합
사용 사례	OpenAI, Tesla와 같은 연구 및 응용 프로그램	Google 및 금융, 의료 등 대기업의 프로덕션 모델	Kaggle 및 교육 기관
주요 도구 및 생태계	PyTorch Lightning, TorchServe	TensorFlow Serving, TensorBoard	TensorFlow와 통합된 API
커뮤니티	연구자 중심의 커뮤니티, 빠른 피드백	대규모 산업 중심 커뮤니티와 이벤트 (예: TensorFlow Summit)	교육 및 초보자 중심 커뮤니티

Keras

Keras

케라스는 많은 이들이 딥러닝을 쉽게 접할 수 있도록 다양한 플랫폼 위에서 딥러닝 모델을 만들 수 있는 API이다.

2019.9.30



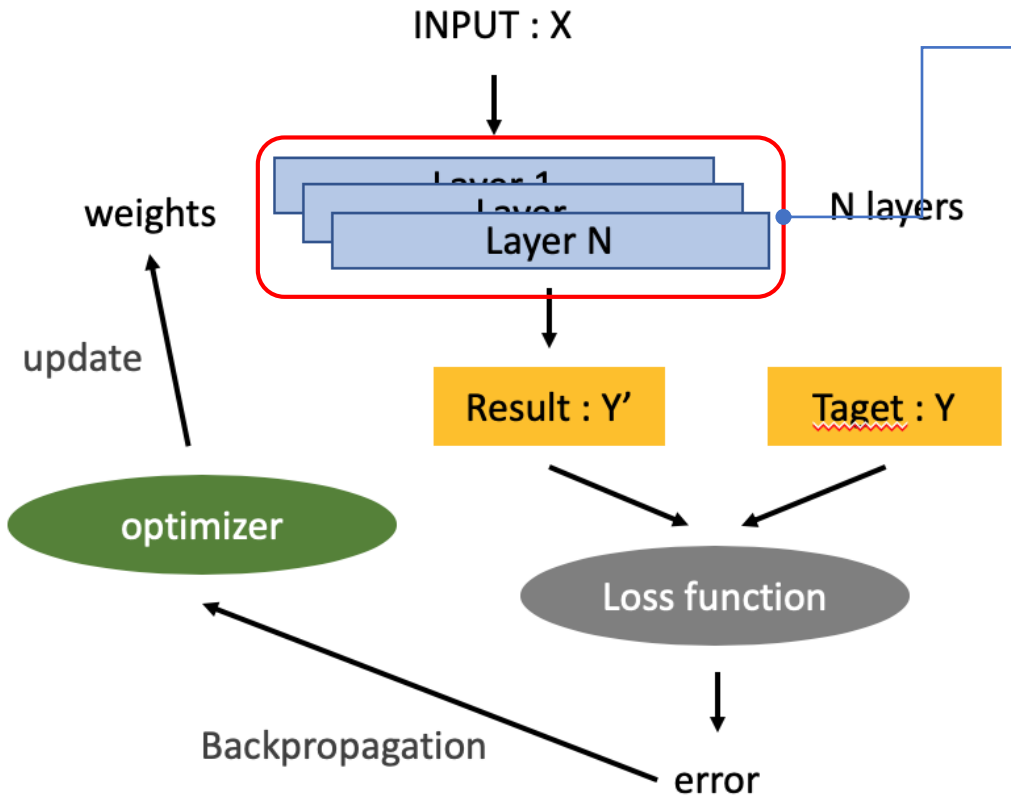
Keras를 이용하여 딥러닝 학습하기 프로세스

케라스는 많은 이들이 딥러닝을 쉽게 접할 수 있도록 다양한 플랫폼 위에서 딥러닝 모델을 만들 수 있는 API이다.

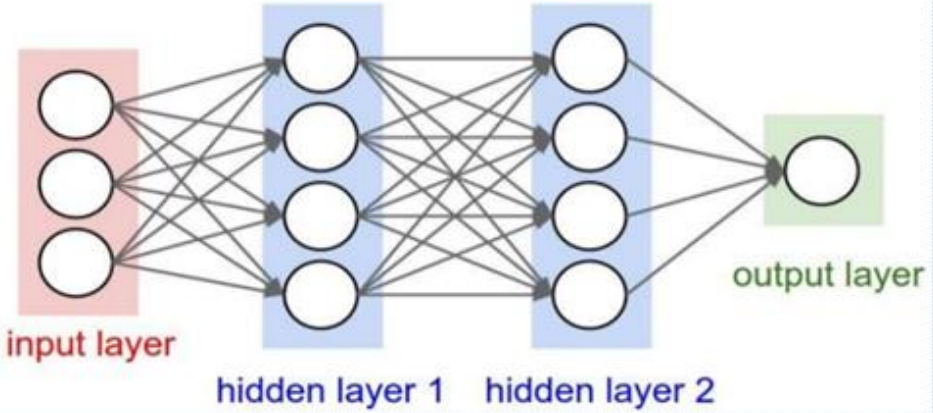
1	모듈 포함하기	• 학습에 필요한 모듈 가져오기 - keras, matplotlib, cv2
2	데이터셋 생성하기	• 훈련셋과 시험셋 불러오기 (train_image, test_image) • 데이터셋 전처리 (2차원 ➔ 1차원으로 : reshape) • 원핫인코딩 처리
3	신경망 모델 구성하기	• 모델 층 만들기 : Sequential() 함수
4	모델 학습과정 설정하기	• 신경망 최적화하기, 손실함수 정의하기 : compile()함수 - optimizer , loss
5	모델 학습하기	• 모델 학습하기 : fit() 함수
6	학습 과정 살펴보기	• 학습 과정의 손실(loss) 및 및 정확도(accuracy) 나타내기 - matplotlib 의 pyplot 이용하여 그래프로 표현
7	모델 평가 및 예측하기	• 시험셋으로 모델 평가하기 : evaluate() 함수 • 모델 사용하여 예측하기 : predict() 함수

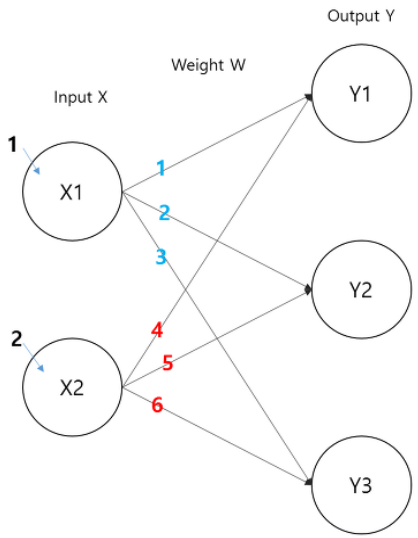
딥러닝 작동 원리

- 층 구조 및 오차 역전파



- Hidden layer의 수 <= 1 → shallow network
- Hidden layer의 수 >= 2 → deep network





X * W = Y

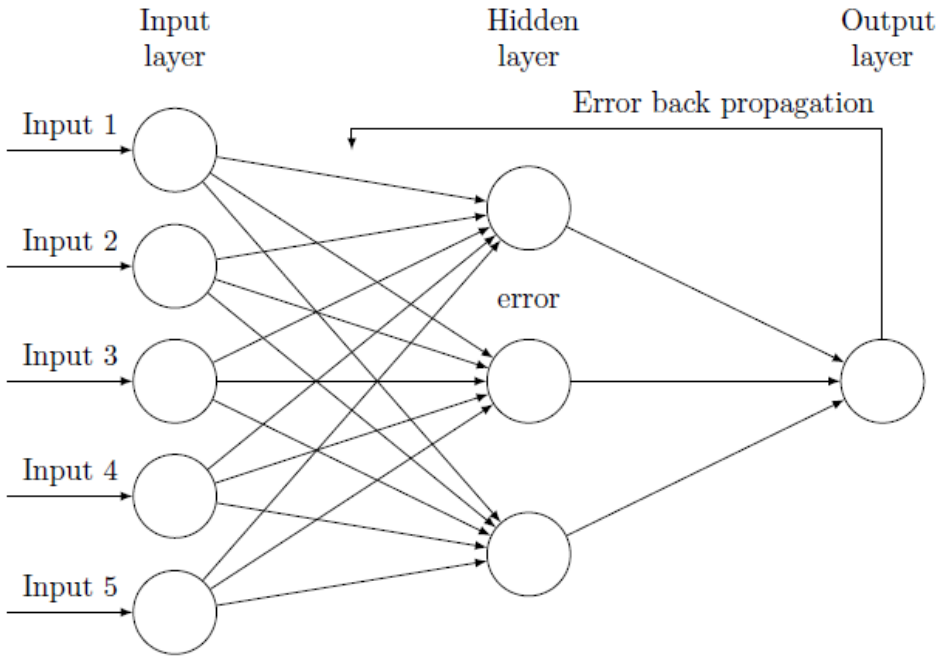
1 x 2 * 2 x 3 = 1 x 3

$(1 \quad 2) \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} (9 \quad 12 \quad 15)$

MLP의 한계

MLP 층의 수가 많아지면서 늘어나는 Weight와 Bias를 수동으로 설정하여 학습시키기는데 한계

오차 역전파 알고리즘
Back Propagation Algorithm , 1986년



- Vanishing gradient, Non convex optimization problem - 최적값이 아닌 지점에서 학습 멈춰버린 문제
- Curse of dimensionality, Overfitting - 고차원의 저주와 과적합 문제
- Low learning time - 느린 학습시간

1974년 Paul Werbos이 처음 제안한 알고리즘으로
후진 방식 자동 미분에서 영감을 받아
오차(실제값 - 예측값)를 출력층에서 입력층으로 역으로 전파시켜
각 층의 가중치와 편향치를 계산하는 방법

MLP의 한계 극복

- 1980~1990년대 후반까지 MLP의 한계를 극복하지 못했지만, 이후 딥러닝 연구와 함께 다음의 혁신적인 기술들로 해결됨

Vanishing Gradient 문제

- **ReLU 활성화 함수**:
  - ✓ Sigmoid 함수 대신 ReLU를 사용.
  - ✓ ReLU는 기울기가 0이 되지 않으며, Vanishing Gradient 문제를 크게 완화.
- **가중치 초기화 방법 개선**:
  - ✓ 가중치를 적절히 초기화하는 기술(예: He 초기화, Xavier 초기화)을 통해 학습 시작 시 기울기 소멸 문제를 줄임.
- **배치 정규화 (Batch Normalization)**:
  - ✓ 각 층에서 입력값을 정규화하여 기울기 흐름이 안정적이게 만듦.

Overfitting 문제

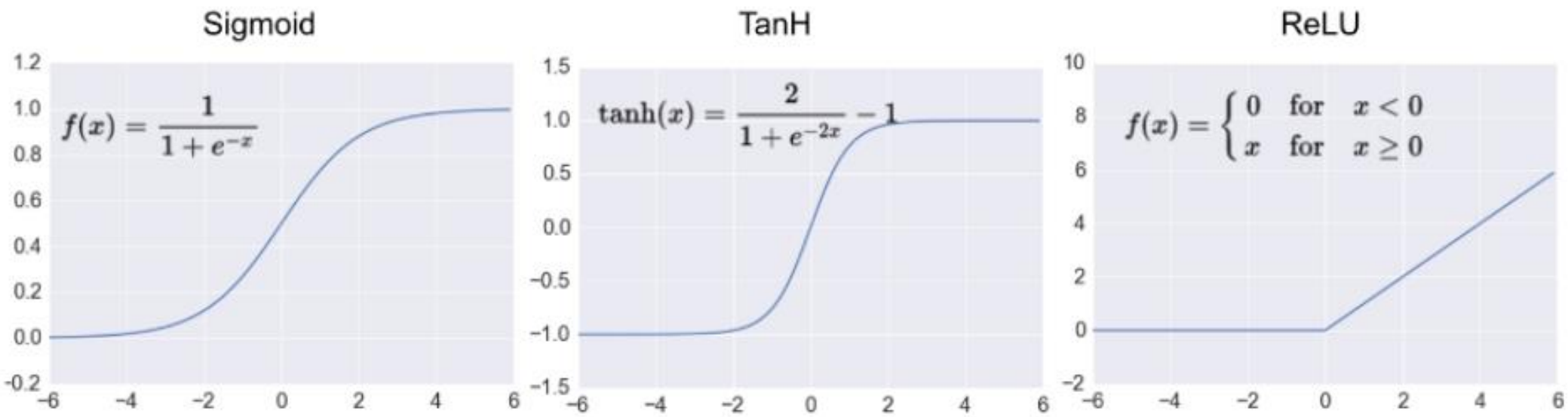
- **드롭아웃(Dropout):**
  - ✓ 학습 과정에서 일부 뉴런을 임의로 비활성화하여 과적합을 방지.
- **데이터 증강 (Data Augmentation):**
  - ✓ 훈련 데이터를 증대하여 일반화 성능을 높임.
- **정규화 기법:**
  - ✓ L1, L2 정규화를 사용하여 모델 복잡도를 제한.

느린 학습 속도 문제

- **GPU 및 병렬 계산**:
  - ✓ 그래픽 처리 장치(GPU)를 활용하여 병렬 연산을 수행, 학습 시간을 획기적으로 단축.
- **효율적인 알고리즘**:
  - ✓ 미니배치 경사 하강법(Mini-batch Gradient Descent)을 사용하여 더 빠르고 안정적으로 학습.
- **대규모 데이터셋의 등장**:
  - ✓ 빅데이터 시대가 도래하며, 신경망이 제대로 학습할 수 있는 양질의 데이터가 늘어남.

딥러닝 작동 원리 - 활성화함수

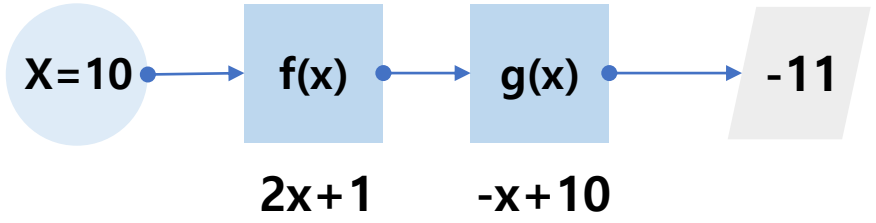
- 활성화 함수(activation function) : 생물학적 뉴런(neuron)에서 입력 신호가 일정 크기 이상일 때만 신호를 전달하는 메커니즘을 모방한 함수

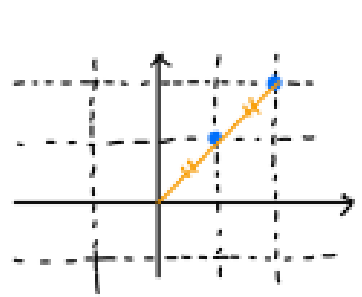


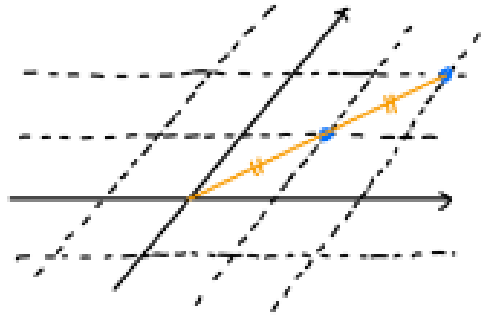
활성화 함수 비교

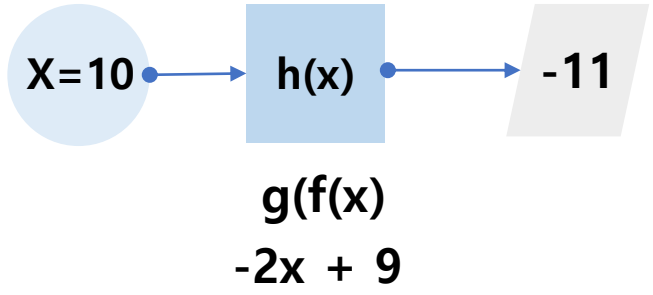
(http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/)

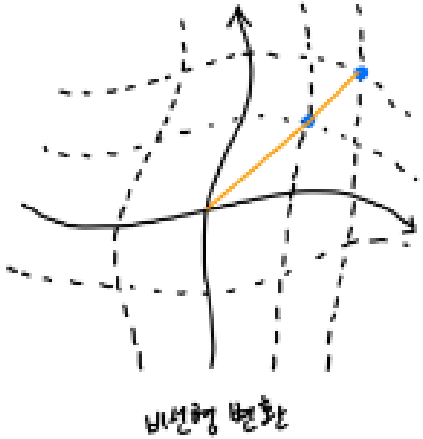
딥러닝 작동 원리 – 선형변환



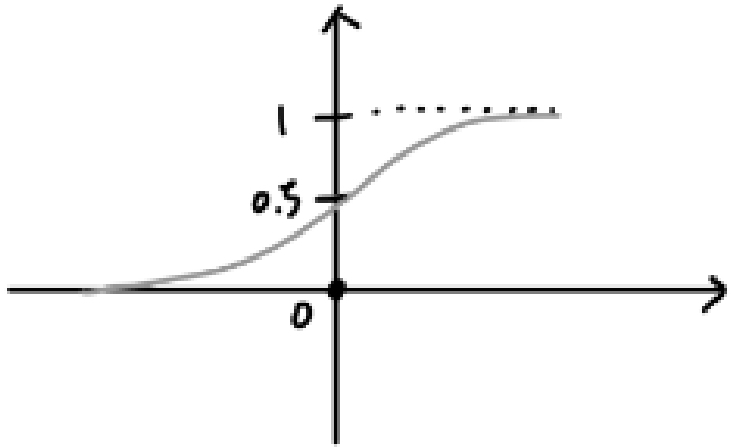








딥러닝 작동 원리 – 비선형변환



$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

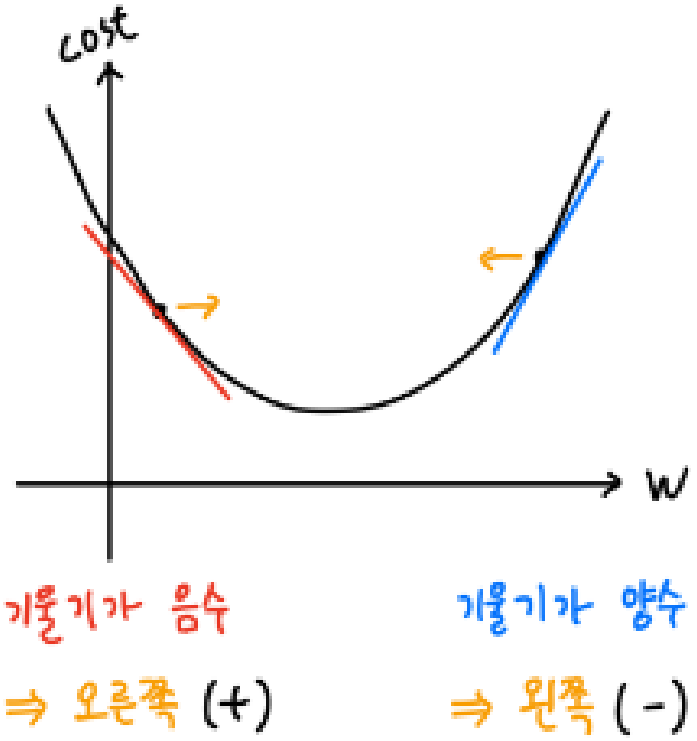
$$e \approx 2.718$$

```
>>> np.exp(-10)
4.5399929762484854e-05
```
Sigmoid 적용 → 0

```
>>> np.exp(0)
1.0
```
Sigmoid 적용 → 1/2

```
>>> np.exp(10)
22026.465794806718
```
Sigmoid 적용 → 1

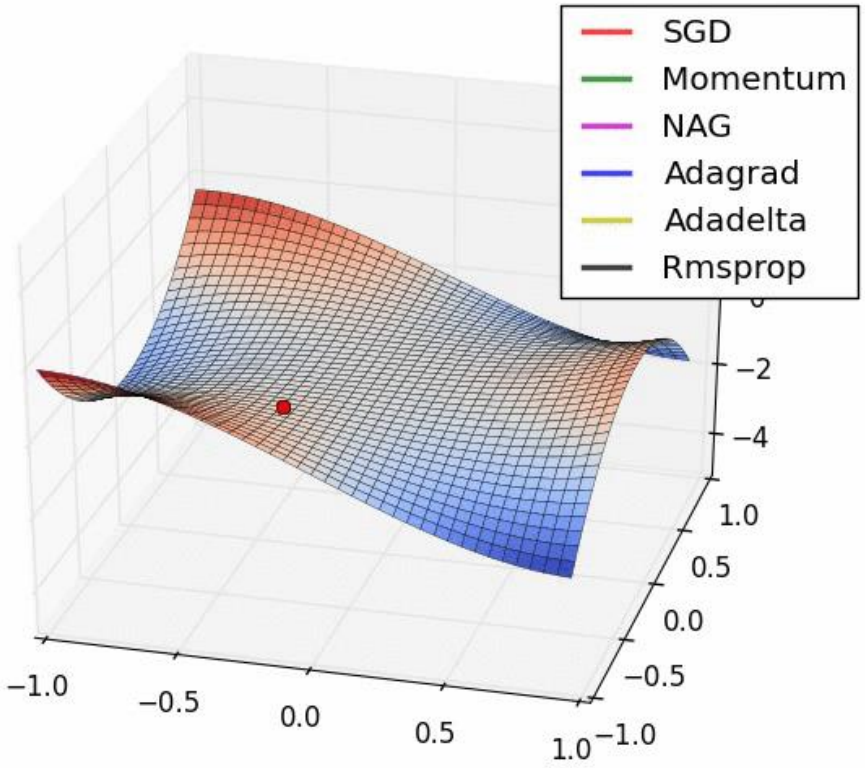
딥러닝 작동 원리 – 비용함수 업데이트 &학습률

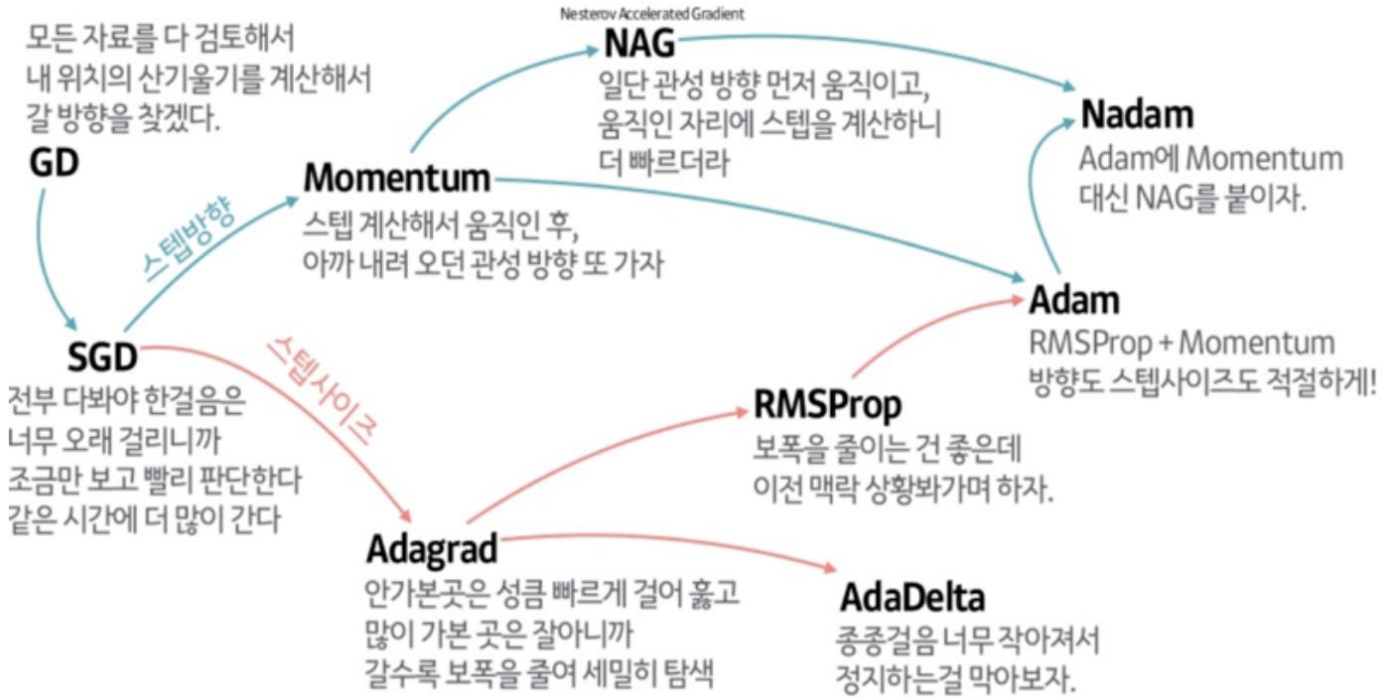


새로운 w = 원래 w + (기울기 반대 방향)

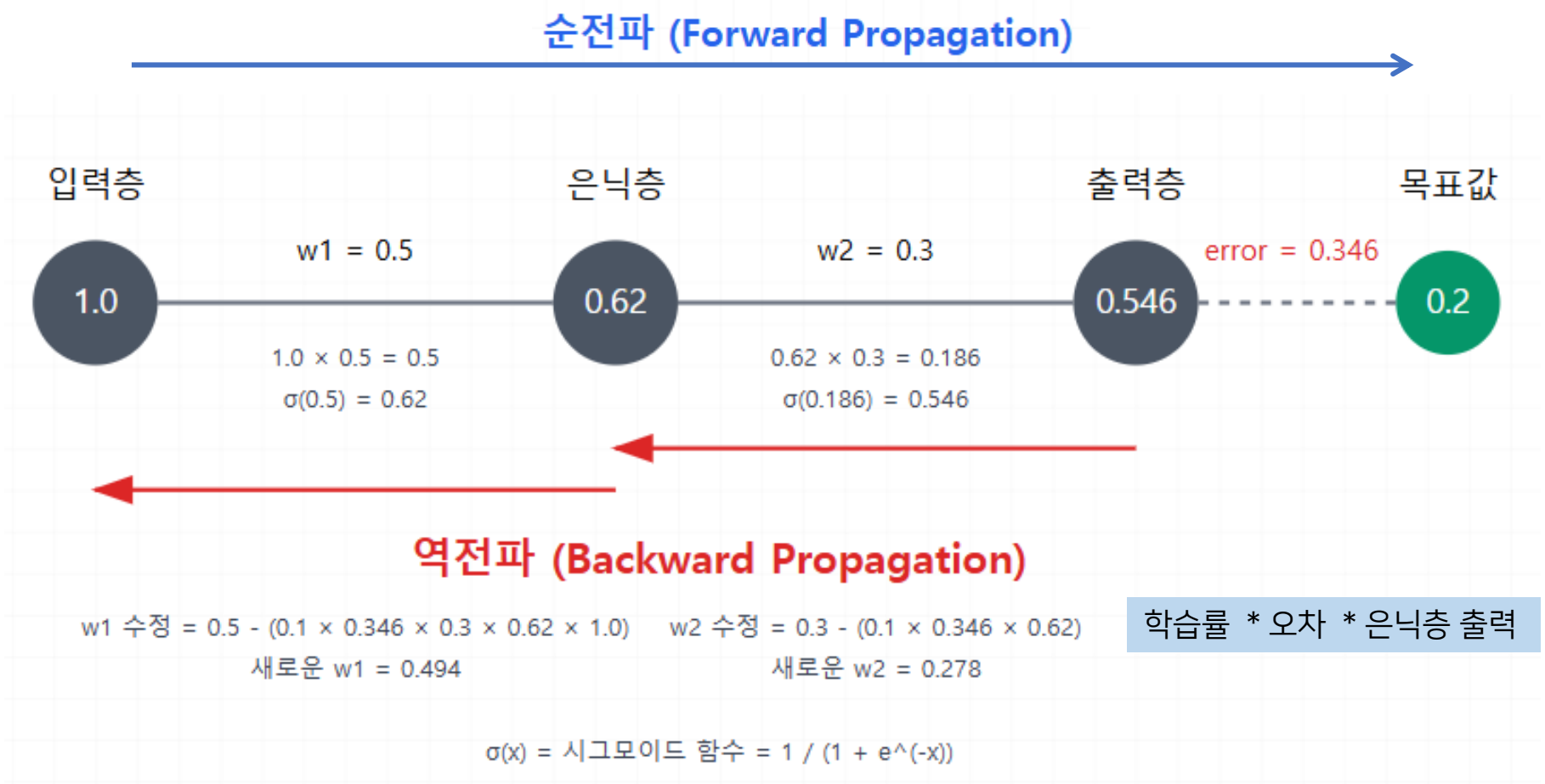
$$w := w - \alpha \frac{\partial C}{\partial w}$$

딥러닝 작동 원리 – 옵티마이저





딥러닝 작동 원리 – 순전파/역전파



딥러닝 작동 원리 – 역전파에 체인룰 적용

체인룰의 핵심 : "합성 함수의 미분"

> 최종 출력 = f3(f2(f1(x)))
> x에 대한 미분:
df3/dx = (df3/df2) × (df2/df1) × (df1/dx)

- **순전파 과정**

z1 = w1 × x (x=1.0)
a1 = σ(z1) (은닉층 활성화)
z2 = w2 × a1
y = σ(z2) (최종 출력)
E = (y - target)²/2 (오차)

- **역전파 과정 : w2 업데이트를 위한 체인룰**

∂E/∂w2 = ∂E/∂y × ∂y/∂z2 × ∂z2/∂w2

∂E/∂y = (y - target)
∂y/∂z2 = y × (1-y) [시그모이드 미분]
∂z2/∂w2 = a1

새로운w2 = w2 - 학습률 × (y-target) × y×(1-y) × a1

- **역전파 과정 : w1 업데이트를 위한 체인룰**

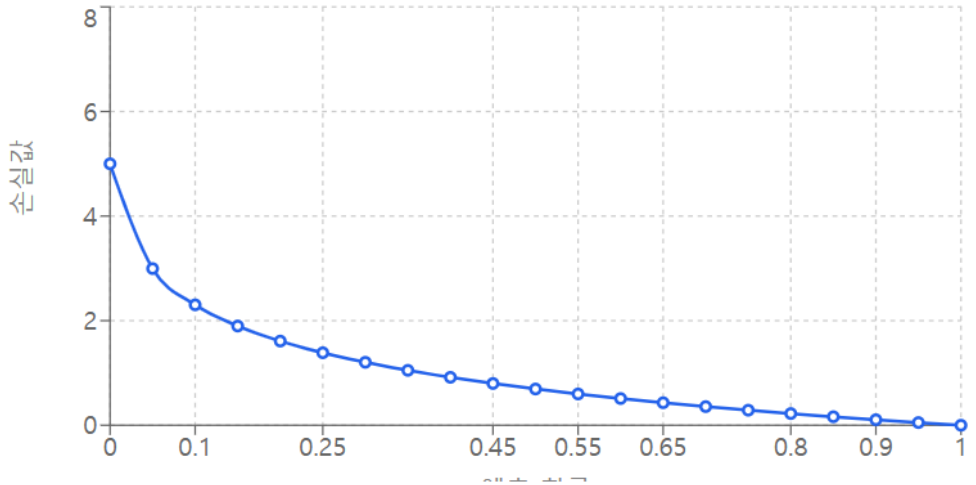
∂E/∂w1 = ∂E/∂y × ∂y/∂z2 × ∂z2/∂a1 × ∂a1/∂z1 × ∂z1/∂w1

∂E/∂y = (y - target)
∂y/∂z2 = y × (1-y)
∂z2/∂a1 = w2
∂a1/∂z1 = a1 × (1-a1) [시그모이드 미분]
∂z1/∂w1 = x

w1_새로운 = w1 - 학습률 × (y-target) × y×(1-y) × w2 × a1×(1-a1) × x

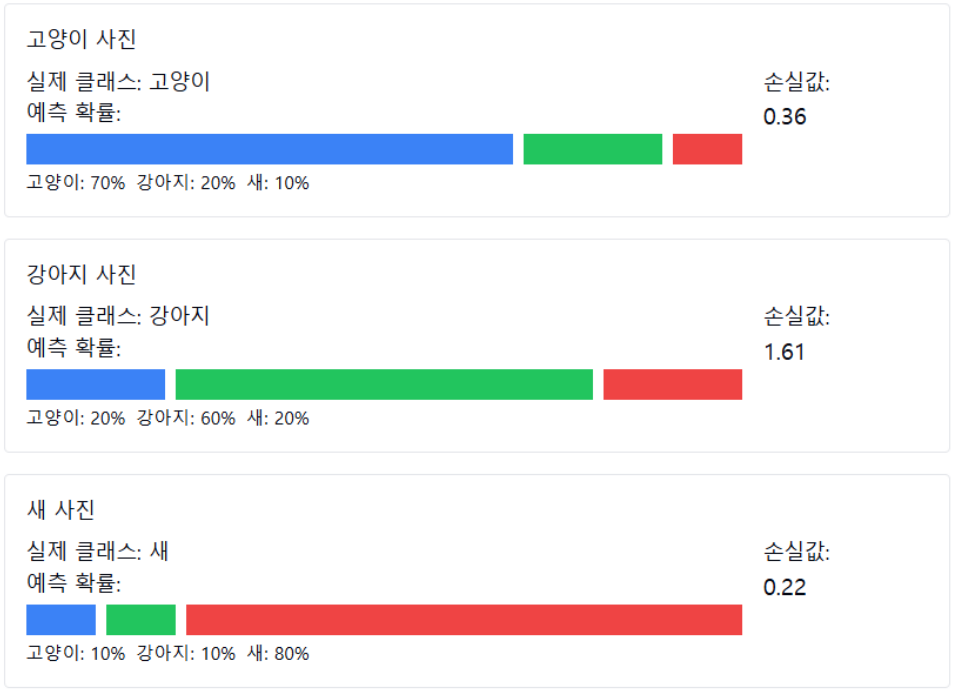
딥러닝 작동 원리 – 분류:손실함수

- 이진분류 - Binary Cross Entorpy



실제값(1)과 가까울수록 손실이 0에 가까워짐
하나의 확률 값
(예: y^=0.9₩hat{y} = 0.9y^=0.9)

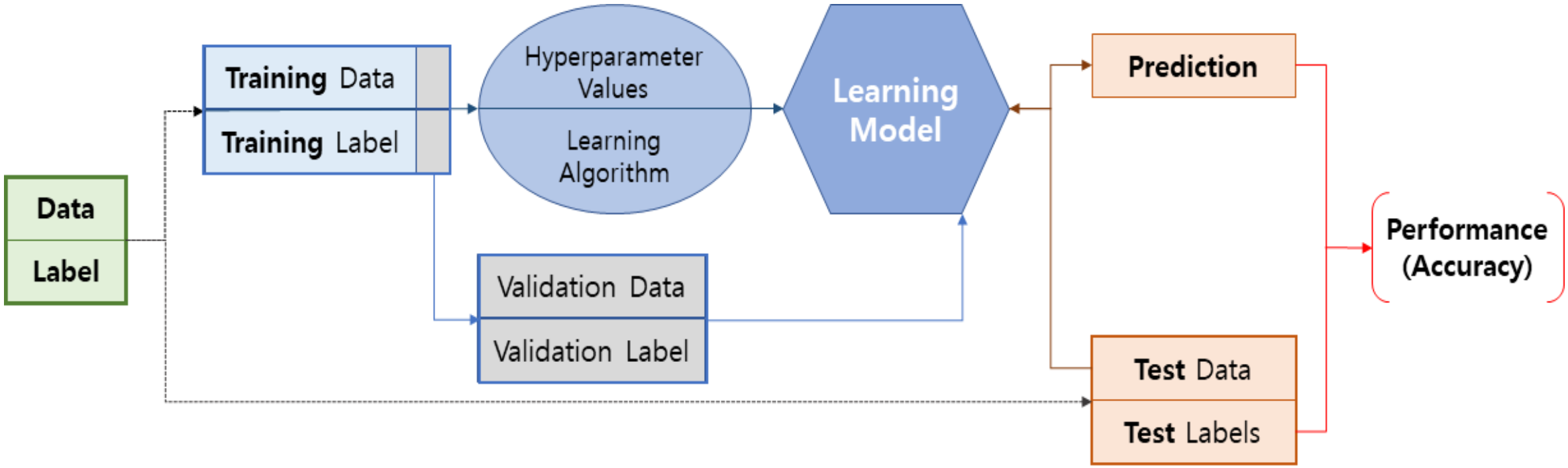
- 다중 클래스 분류 - Categorical Cross Entorpy



각 클래스별 확률 값 (예: y^=[0.8,0.1,0.1]
각 클래스별 확률의 로그 값을 합산하여 계산

데이터셋 사용

- 훈련데이터셋, 테스트 데이터셋, 검증데이터셋





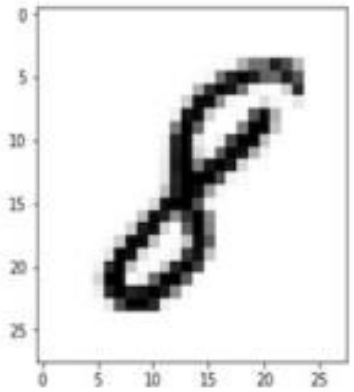

PART2. 이미지 분류에 대한 이해

Enjoy your possibility

MNIST 데이터셋을 사용한 이미지 분류 이해 및 Keras로 직접 모델 구성해보기

손글씨 분류 프로젝트

- 손글씨 분류 프로젝트



- 1990년 미국 우편 서비스의 우편 봉투의 우편 번호 코드를 자동으로 읽기 위해 사용함.
- 손글씨 이미지 분류에 합성곱 신경망과 역전파를 이용한 LeNet이라는 신경망을 만들어 적용함.

- 학습, 분류 및 컴퓨터 비전 시스템의 표준 벤치마크
- 손글씨 숫자 이미지 집합
- 0~9까지 (10개 클래스) 28×28 회색조 이미지 크기
- 훈련데이터 – 6만장, 테스트데이터-1만장

숫자 이미지 분류하기 – DNN

- 1) 필요한 모듈 가져오기

```
1  import tensorflow as tf
2  from tensorflow import keras
3  from tensorflow.keras import layers
4  from tensorflow.keras.datasets import mnist
5  import matplotlib.pyplot as plt
6  import sys
7  import numpy as np
```

숫자 이미지 분류하기 – DNN

- 2) MNIST 데이터셋을 가져와서 데이터 형태 살펴보기

```
# mnist 데이터 가져오기
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
# 가져온 데이터 형태 확인하기
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

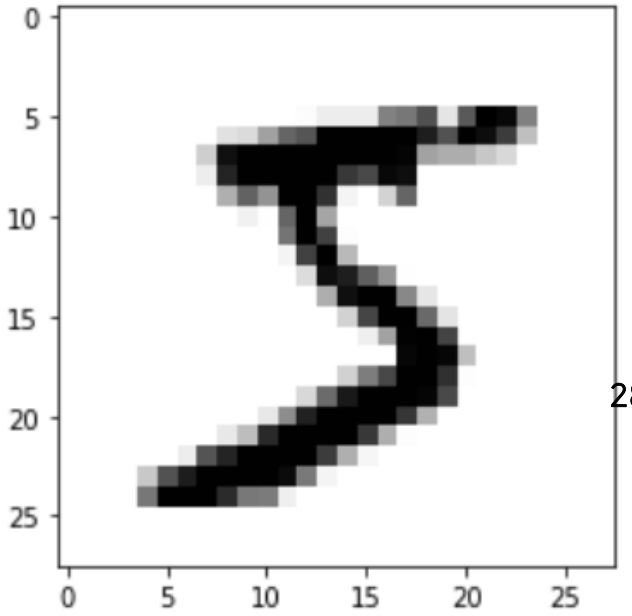
```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

숫자 이미지 분류하기 – DNN

- 3) 가져온 MNIST 데이터의 이미지 확인해보기

```
1  plt.imshow(x_train[0], cmap='binary')
2  print('Label: ', y_train[0])
```

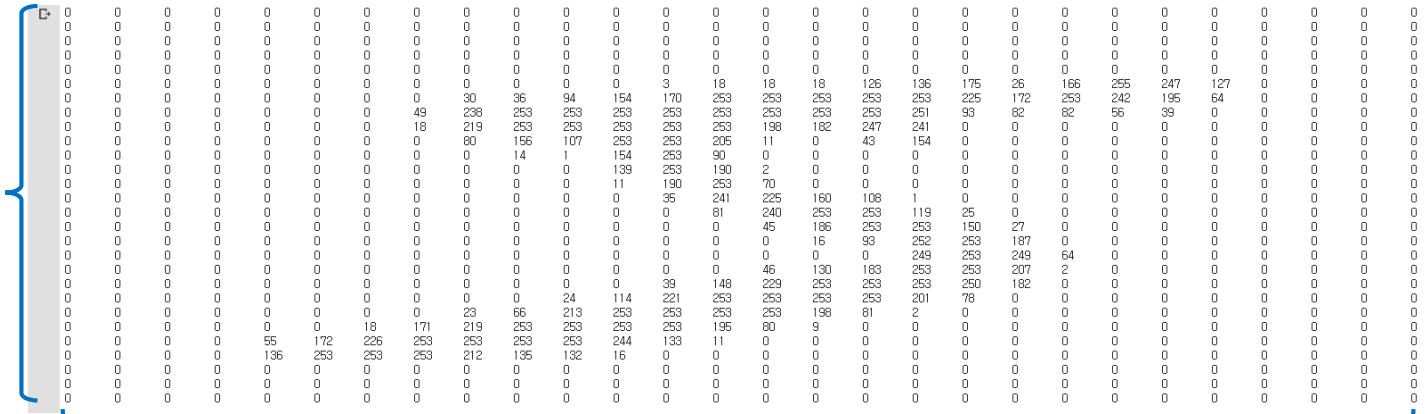
Label: 5



28

```
[ ]  1  # 코드로 이미지가 어떻게 들어가 있는지 확인
     2  for x in X_train[0]:
     3      for i in x:
     4          sys.stdout.write('%d₩t' % i)
     5      sys.stdout.write('₩n')
```

28 * 28 bytes gray



28

숫자 이미지 분류하기 – DNN

- 4) 데이터 스케일링하기

```
x_train, x_test = x_train / 255.0, x_test / 255.0   # 정규화
```

- 5) 학습 모델 만들기

```
1  model = keras.models.Sequential([
2                                   layers.Flatten(input_shape=(28, 28)),
3                                   layers.Dense(128, activation='relu'),
4                                   layers.Dense(10, activation='softmax')
5  ])
6
7  model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_5 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               100480
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
```

숫자 이미지 분류하기 – DNN

- 6) 모델 컴파일하기

```
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# optimizer='adam',
```

- 7) 모델 학습하기

```
history = model.fit(x_train, y_train, epochs=5, validation_split=0.2 )
```

숫자 이미지 분류하기 – DNN

- 8) 학습과정 그래프로 살펴보기

```
1  import numpy as np
2  # 테스트 셋의 오차
3  y_vloss = history.history['val_loss']
4
5  # 학습셋의 오차
6  y_loss = history.history['loss']
7
8  # 그래프로 표현
9  x_len = np.arange(len(y_loss))
10 plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
11 plt.plot(x_len, y_loss, marker='.',  c="blue", label='Trainset_loss')
12
13 # 그래프에 그리드를 주고 레이블을 표시
14 plt.legend(loc='upper right')
15 # plt.axis([0, 20, 0, 0.35])
16 plt.grid()
17 plt.xlabel('epoch')
18 plt.ylabel('loss')
19 plt.show()
```

숫자 이미지 분류하기 – DNN

- 9) 모델 평가하기

```
1  model.evaluate(x_test,  y_test, verbose=2)
```

사전학습 : 이미지 데이터 가져와 특성 살피고 변형해보기

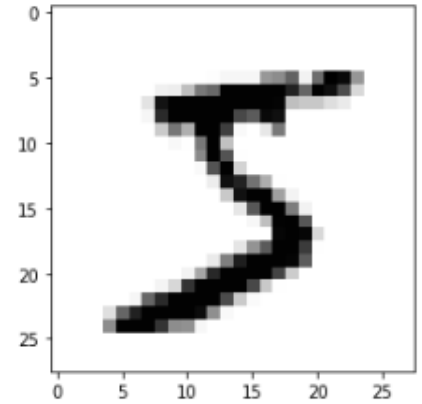
- # 카테고리화 하기

5. 라벨(y data)를 카테고리화 : 원 핫 인코딩(one-hot encoding)

```
[36]  1  # 바이너리화 과정
      2  y_train = np_utils.to_categorical(y_train, 10)
      3  y_test = np_utils.to_categorical(y_test, 10)
      4
      5  print(y_train[0])
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

0	1	2	3	4	5	6	7	8	9





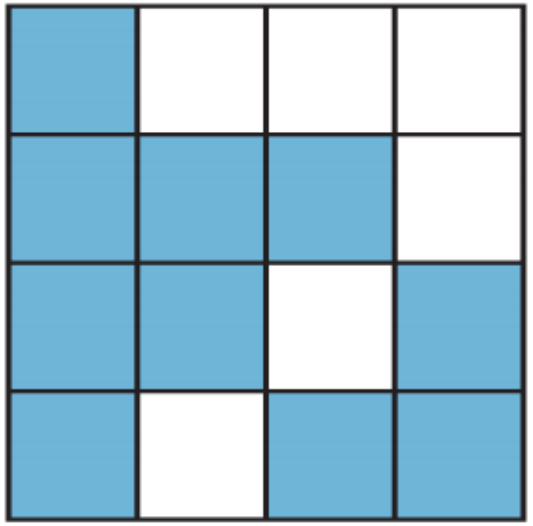

PART3. CNN 대한 이해

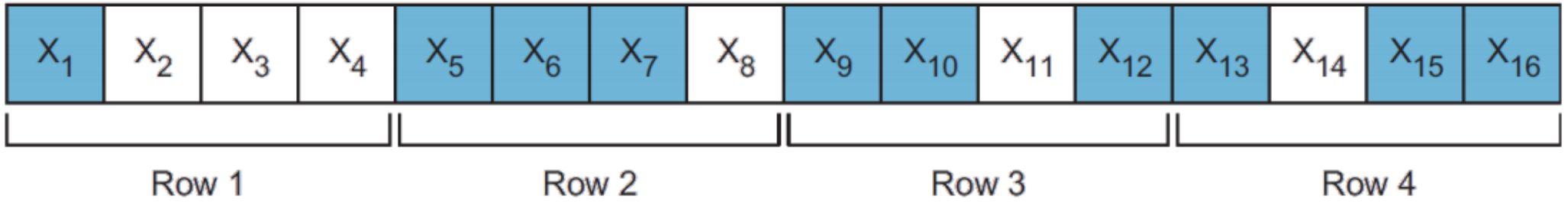
Enjoy your possibility

Tensorflow-keras 로 CNN 알고리즘 구현하기

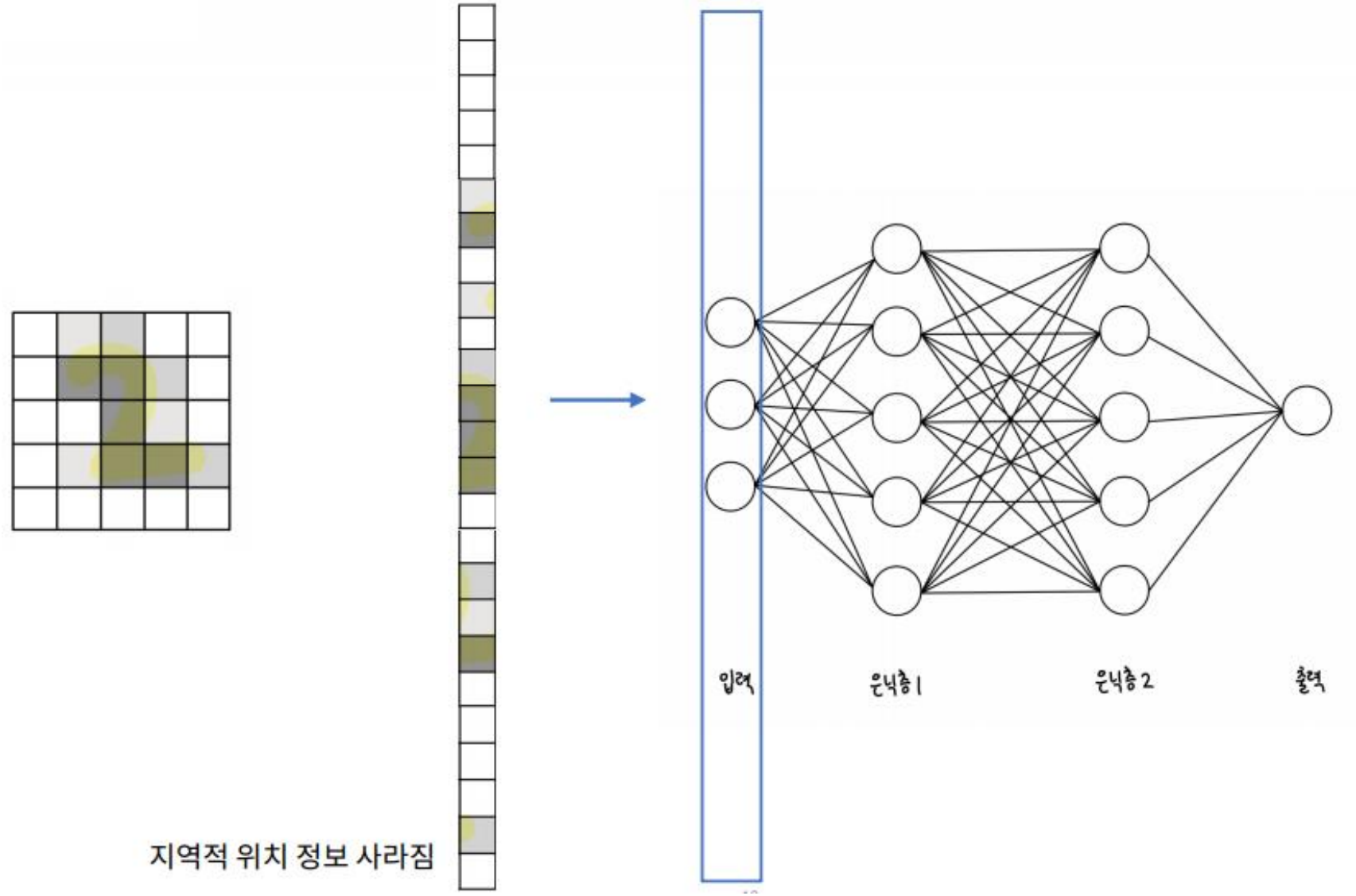
왜 CNN인가?

- 이미지를 1D 벡터 입력으로 평면화하면 2D 이미지의 공간적 특징이 손실됨



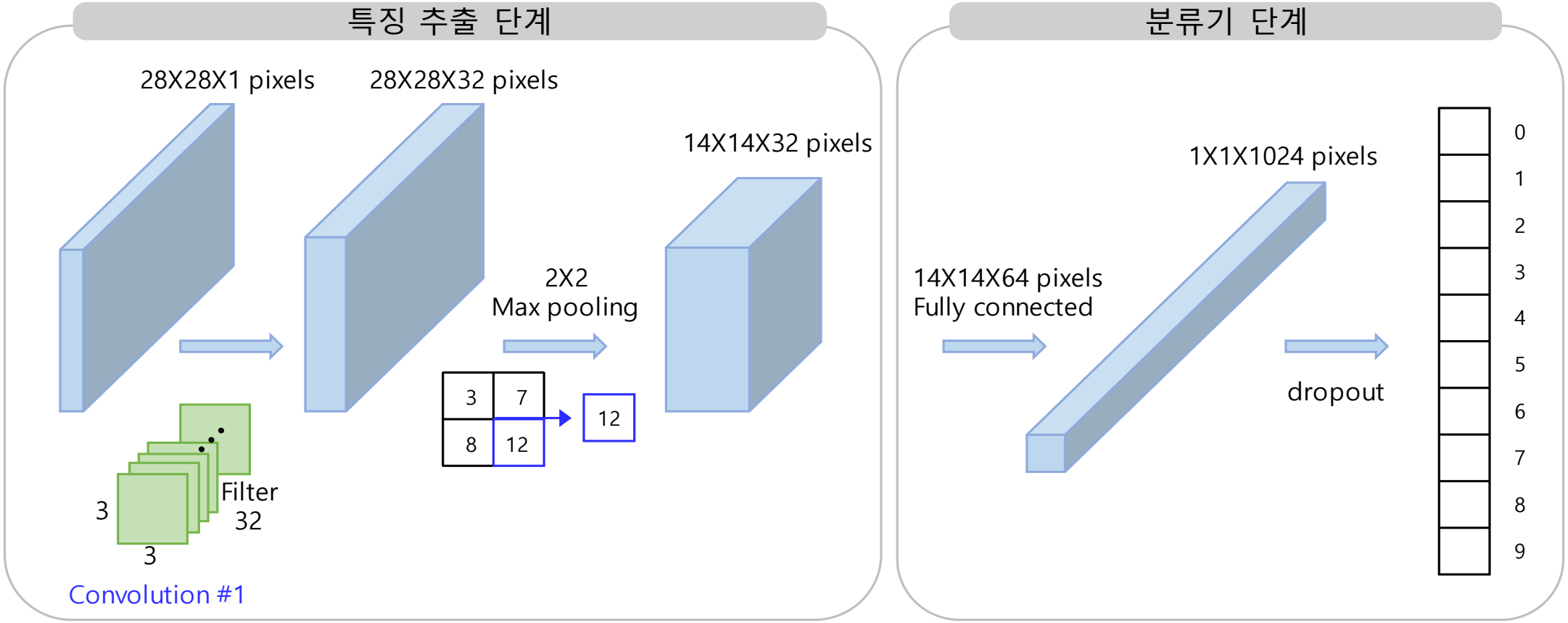


왜 CNN인가?



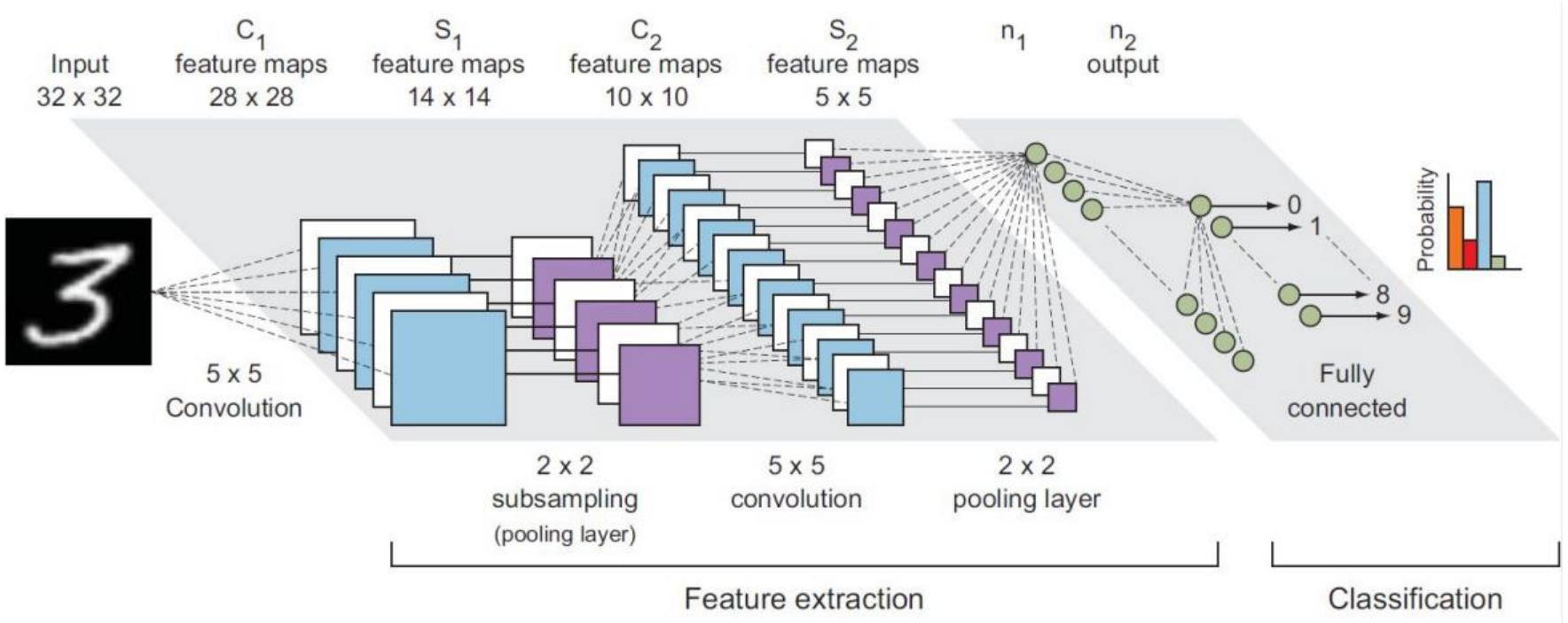
CNN(Convolution Neural Network, 합성곱신경망)

- CNN 구조



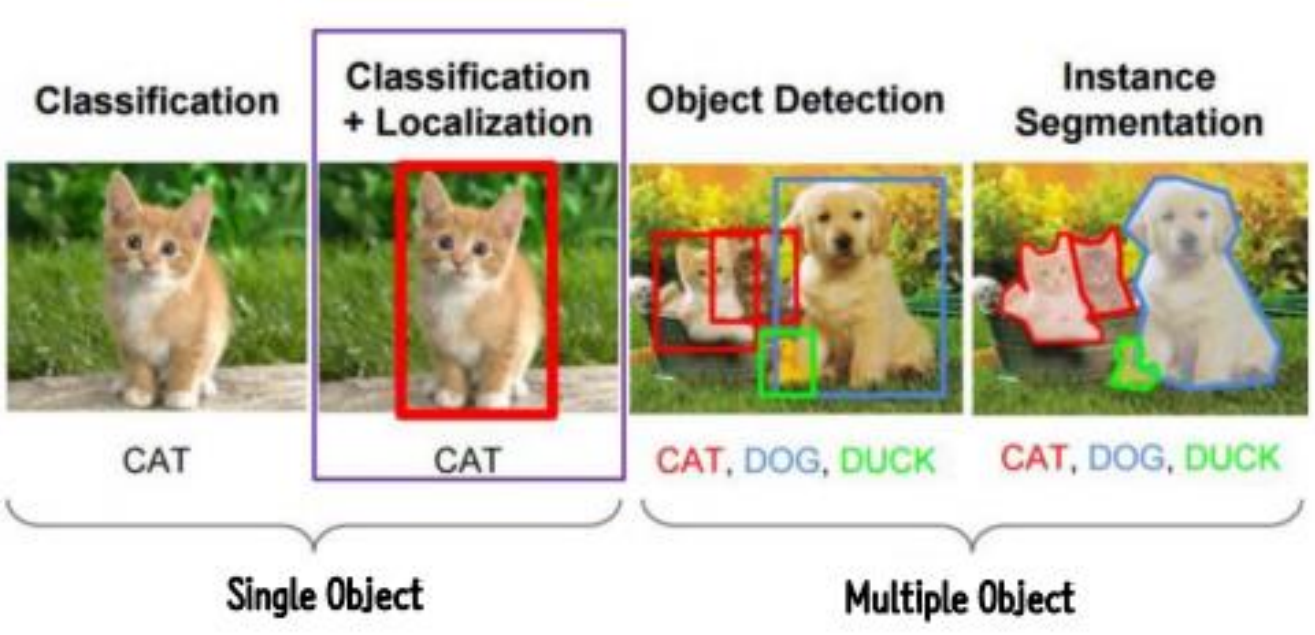
CNN 개요

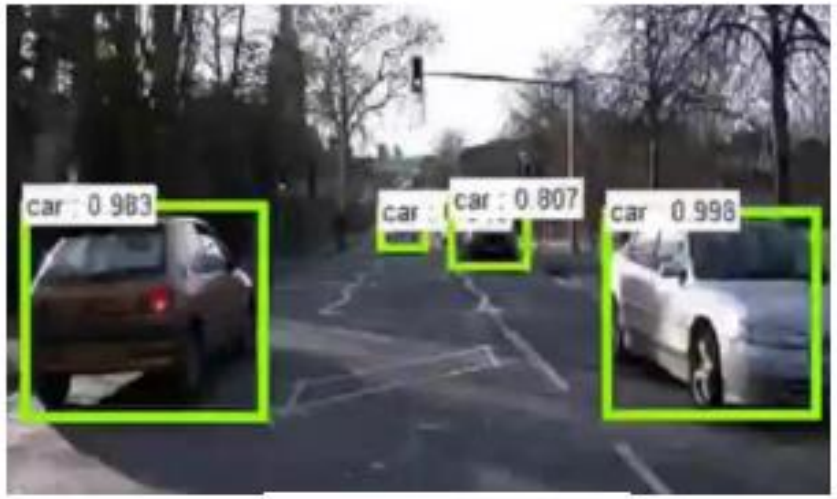
- Input Image : 이미지를 하나의 입력으로 취함
- Convolution Layers : Feature Extraction을 수행하는 Layer
  - ✓ Convolution Layer + ReLU : Feature 추출, 의미 없는 특징을 zero화
  - ✓ Pooling Layer : Feature 개수 축소, 중요한 Feature 만 유지 (선택적 작업)
- Fully-Connected Layer : 비선형 조합 학습 및 분류 작업 수행하는 Layer
- Output Class : 작업의 결과



CNN 활용분야

- 분류(Classification)
- 지역화(Localization)
- 이미지 세분화(Image Segmentation)
- 물체 감지(Object Detection)

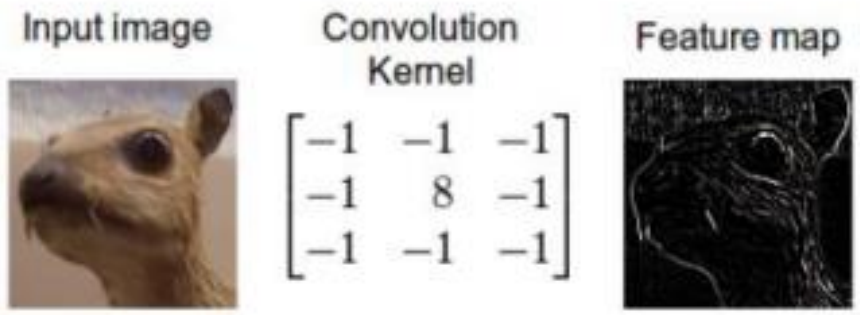


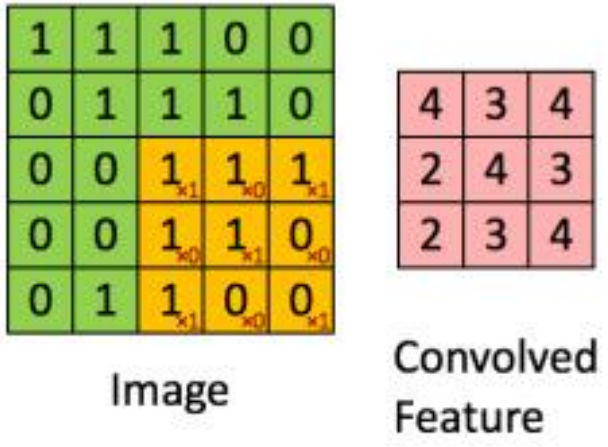


Object Detection

CNN 이해 - 필터 커널(Filter Kernel)

- Filter(Convolution Kernel Matrix)를 적용하여 입력에 대해 특정 성분에 대해서만 뽑아내는 작업
  - ✓ 예) 사선 정보, 직선 정보, 동그란 정보, 각진 정보….
  - ✓ 알고 싶은 특정 성분에 따라 Filter 의 모양이 다름
  - ✓ CNN은 Filter를 갱신하면서 학습하는 것임
- Image에 특정 Filter를 Convolution한 결과를 Feature Map 이라고 함
  - ✓ Feature Map 은 Image에 적용된 Filter 개수 만큼의 Channel을 갖게 됨
  - ✓ n개의 Filter 가 적용된 경우 n개 Channel
- Stride: Filter를 순회하는 간격, Stride가 2로 설정되면 2칸씩 이동하면서 convolution 하게됨





http://deeplearning.stanford.edu/wiki/images/6/6c/Convolution_schematic.gif

CNN 이해 – 필터 커널(Filter Kernel)

- https://setosa.io/ev/image-kernels/

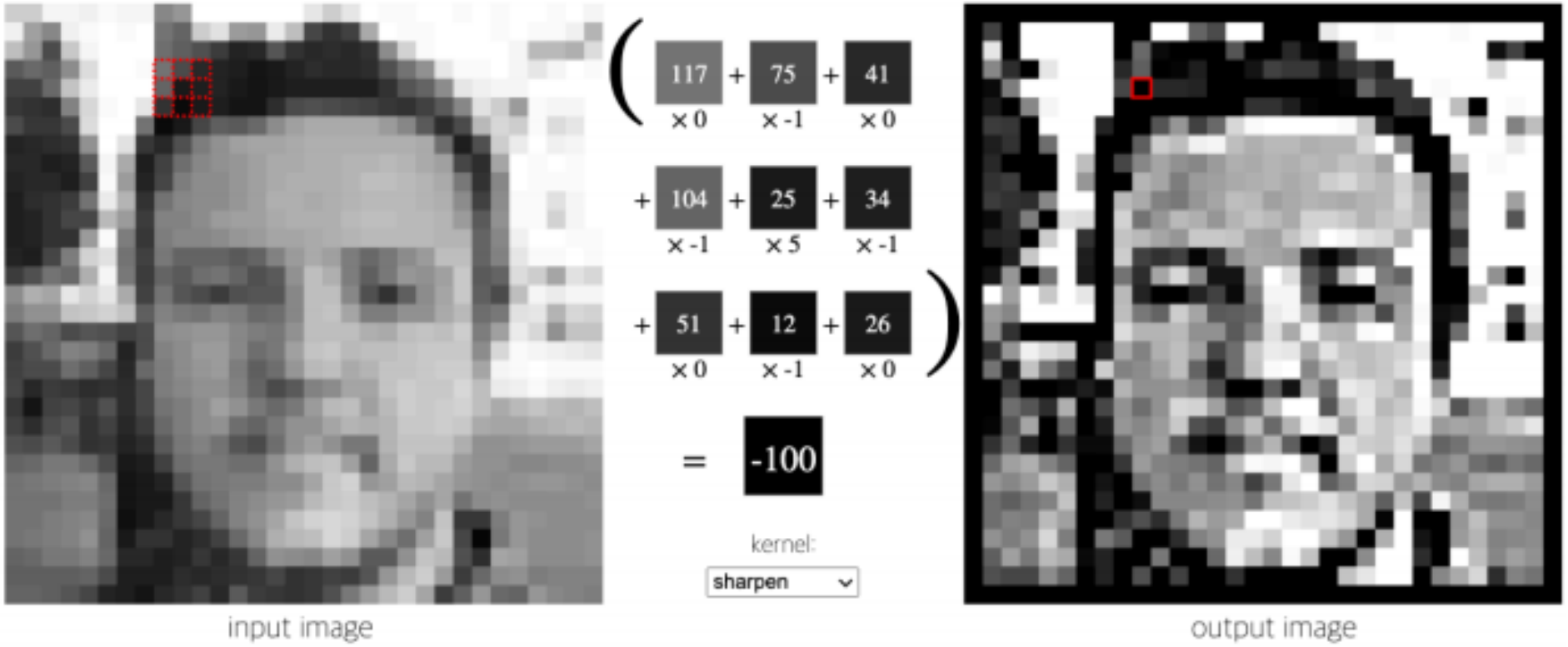
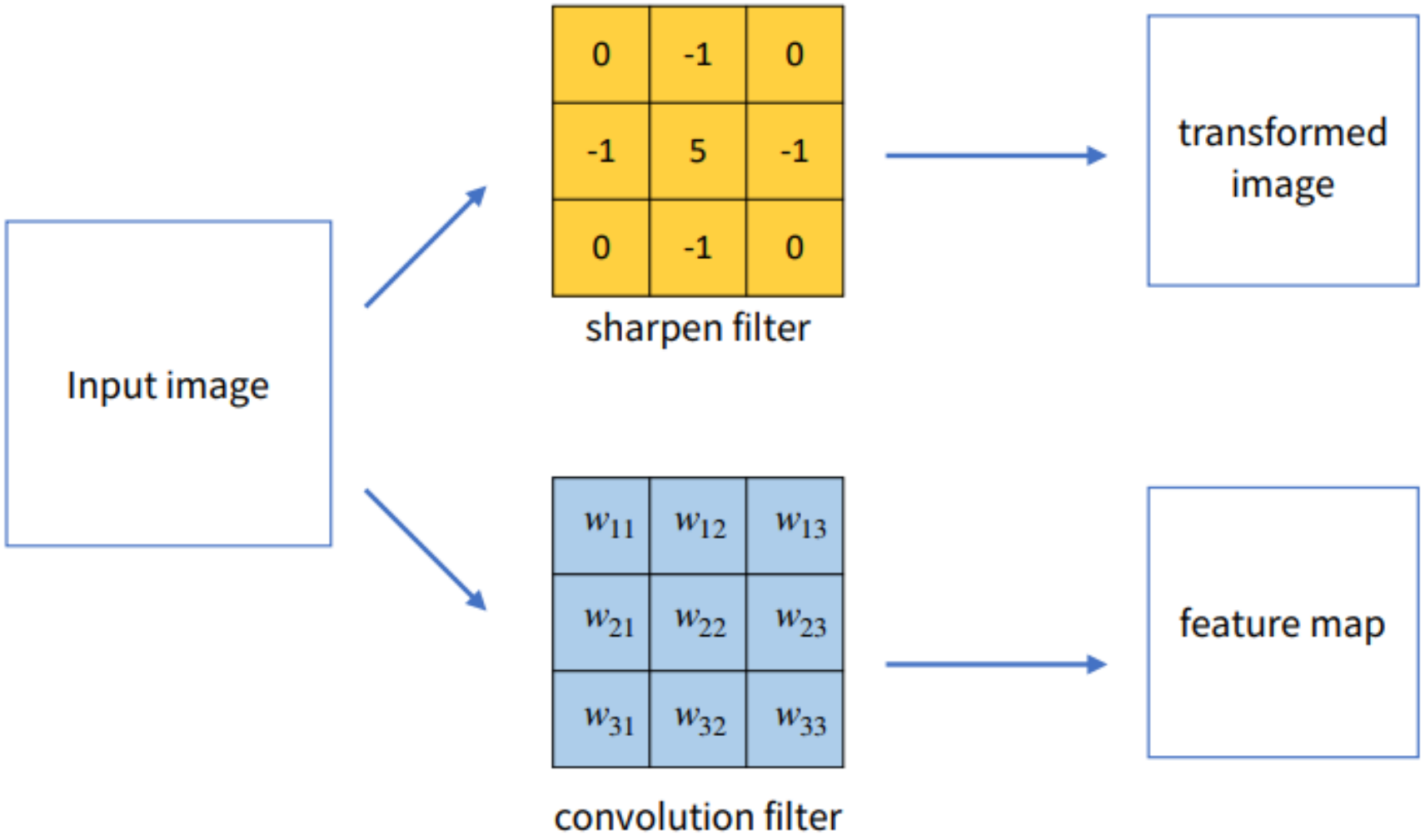


Image Kernels Explained Visually (By Victor Powell)

CNN 이해 – 필터 커널(Filter Kernel)



CNN 이해 - 필터 커널(Filter Kernel)

- 원본 이미지에 특수한 행렬로 컨볼루션을 취함
- 행렬의 특성에 따라 원본 이미지로부터 특성이 강조된 이미지를 얻을 수 있음

①

1	1	1	2	2	2
1	1	1	2	2	2
1	1	1	2	2	2
1	1	1	2	2	2

*

필터 1

-1	0	1
-1	0	1
-1	0	1

*

필터 2

-1	-1	-1
0	0	0
1	1	1

②

1	1	1	1
1	1	1	1
1	1	1	1
2	2	2	2
2	2	2	2
2	2	2	2

*

필터 1

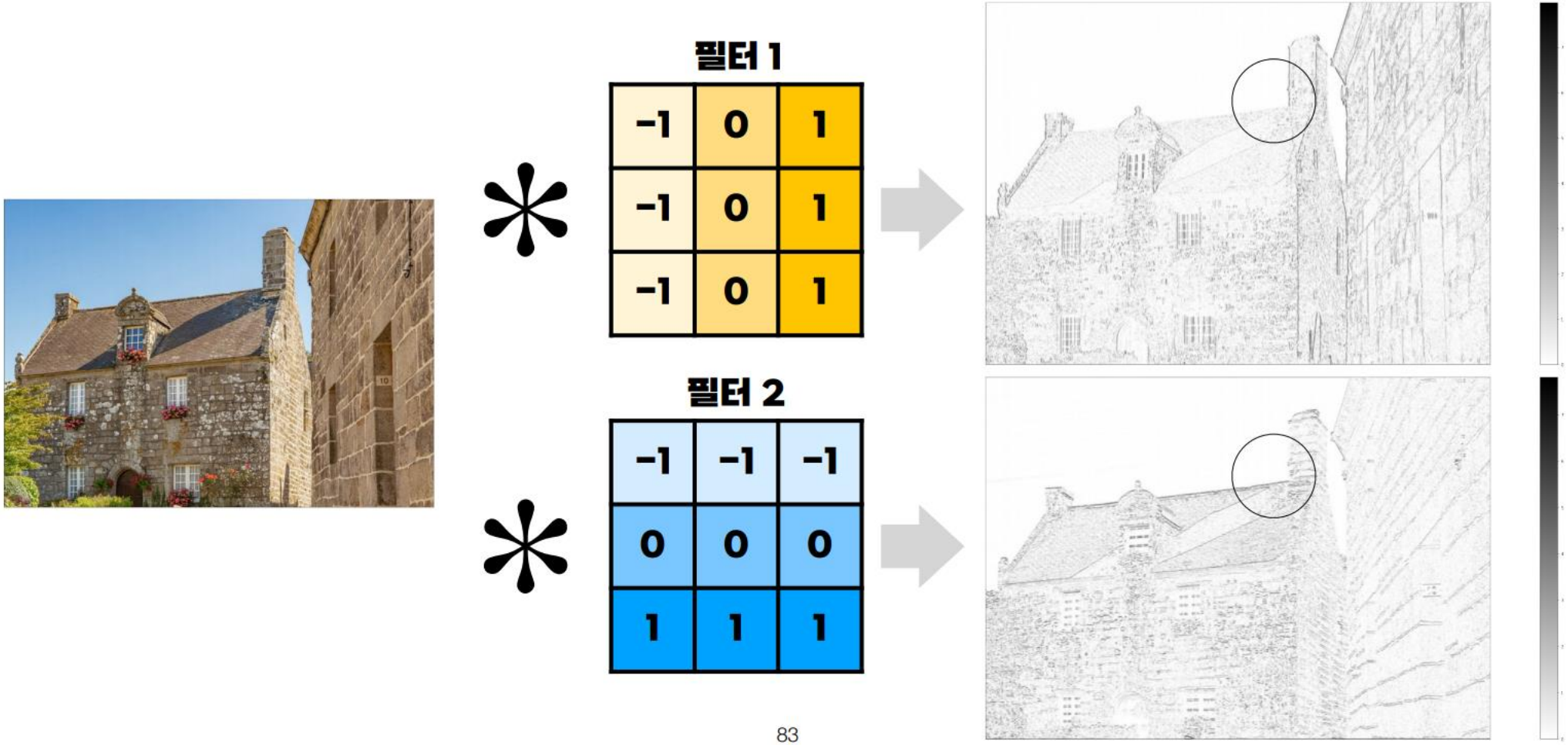
-1	0	1
-1	0	1
-1	0	1

*

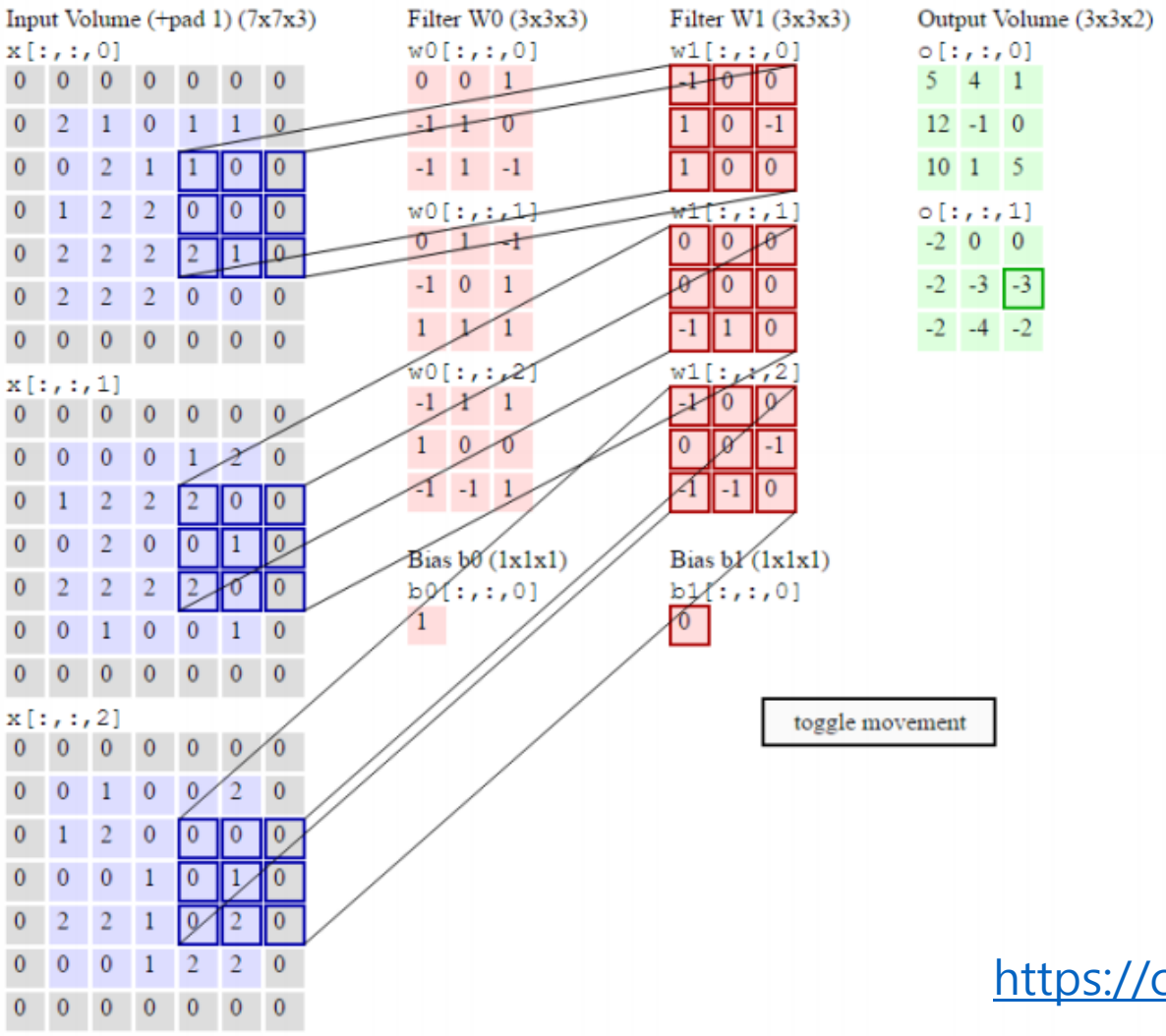
필터 2

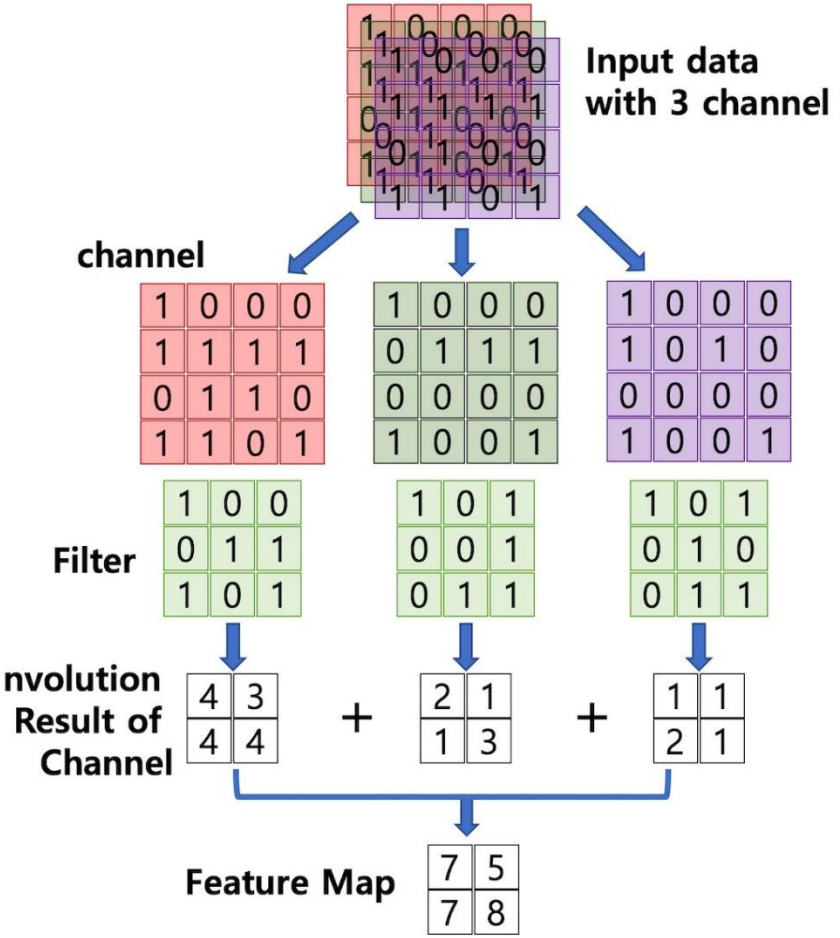
-1	-1	-1
0	0	0
1	1	1

CNN 이해 - 필터 커널(Filter Kernel)



CNN 이해- 필터 커널(Filter Kernel)



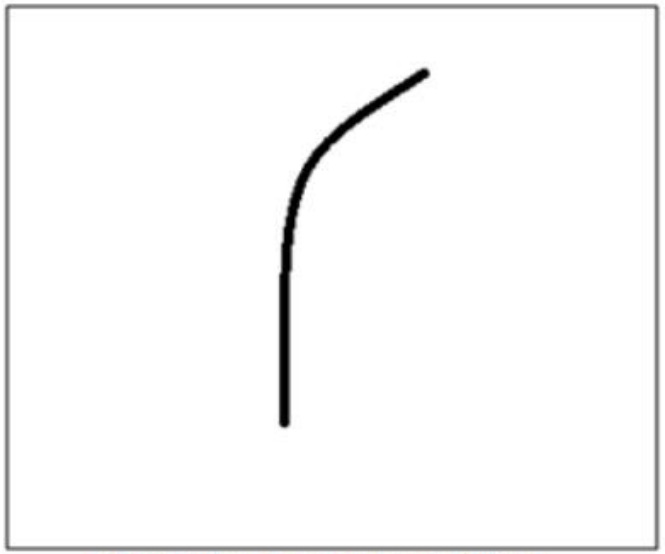


https://cs231n.github.io/convolutional-networks/

CNN 이해 - 필터 커널(Filter Kernel)

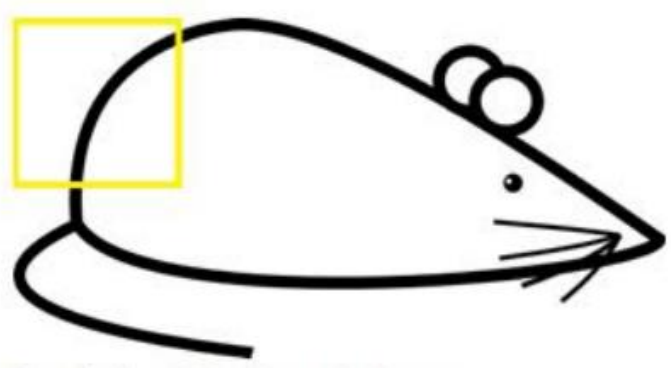
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Original image



Visualization of the filter on the image

CNN 이해 - 필터 커널(Filter Kernel)

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

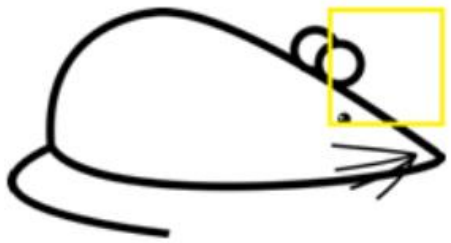
Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

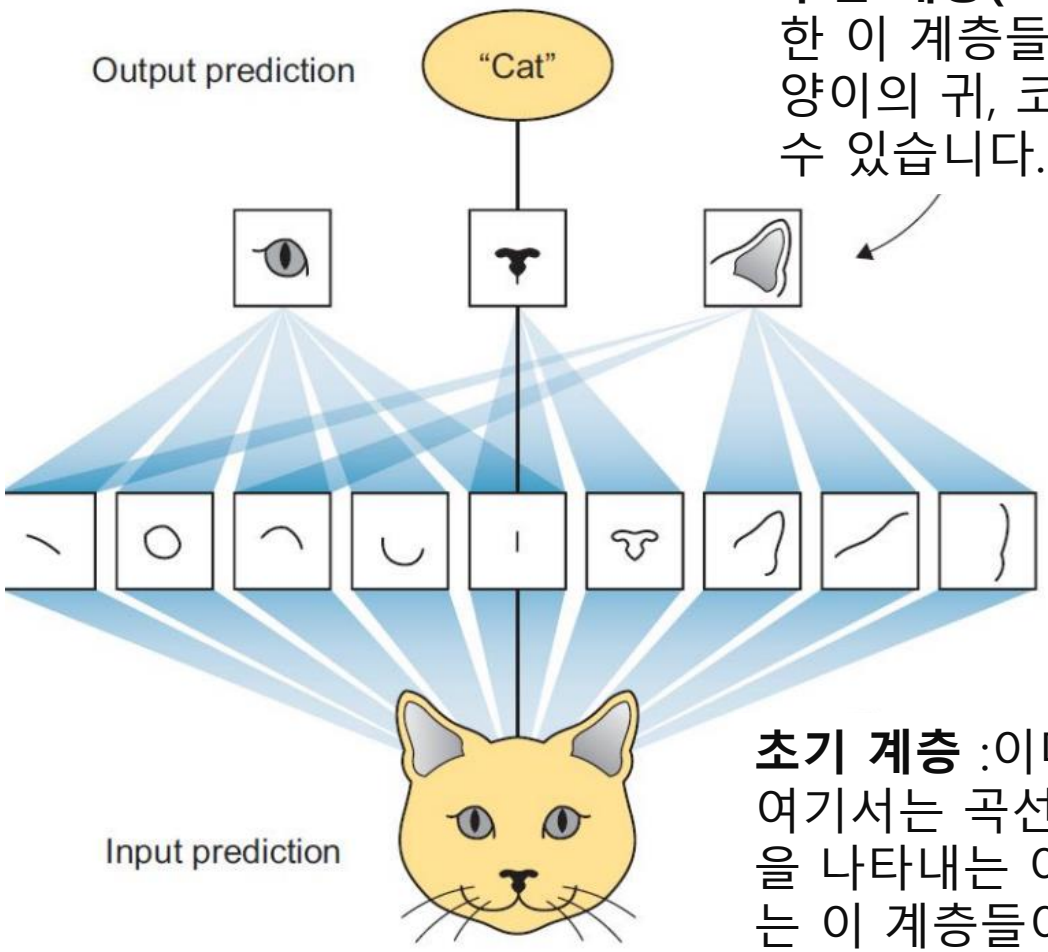
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

CNN 이해



후반 계층(Later layers): 신경망의 깊은 부분에 위치한 이 계층들은 보다 복잡한 특징을 학습합니다. 고양이의 귀, 코, 눈과 같은 특정 객체의 일부를 인식할 수 있습니다.

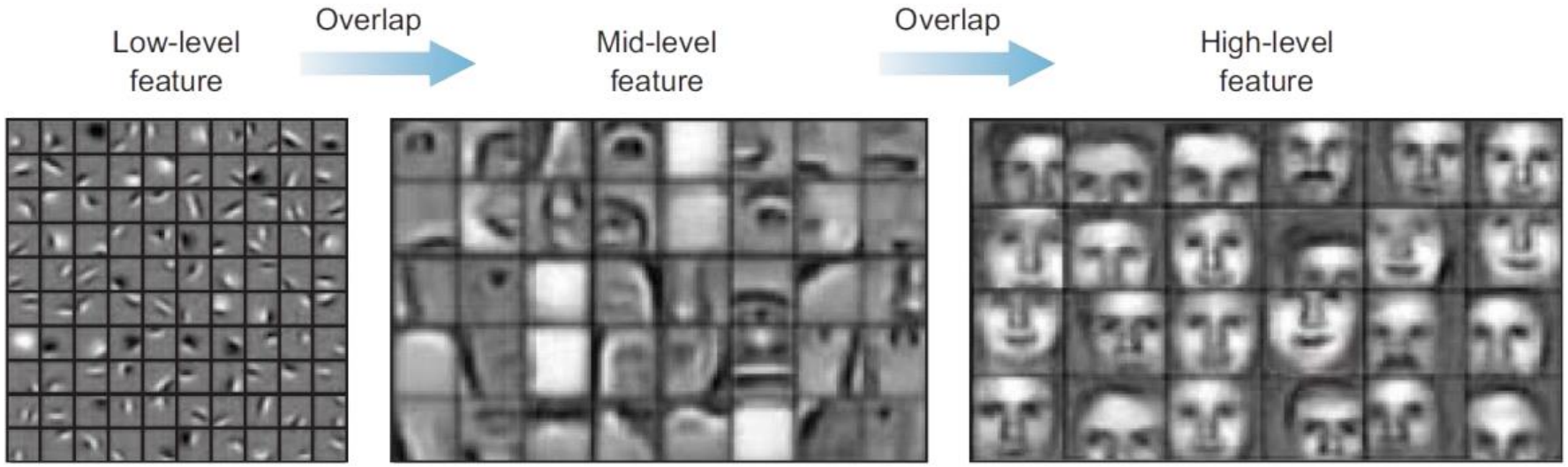
초기 계층 :이미지의 간단한 특징을 학습합니다. 여기서는 곡선, 가장자리 등 기본적인 형태와 패턴을 나타내는 아이콘들이 보여지며, 실제 CNN에서는 이 계층들이 이미지에서 수직선, 수평선 등의 기본적인 형태를 감지합니다.

CNN 이해

신경망의 높은 계층이 낮은 계층의 출력에 기반한 가중치가 적용된 합으로 형성됨

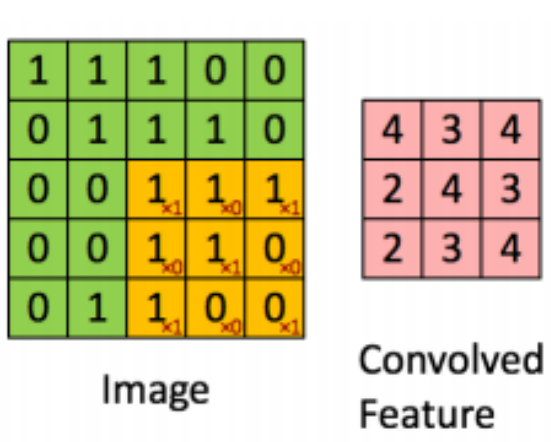
각 높은 계층의 뉴런은 낮은 계층의 특징들을 받아 이들의 가중합을 계산하고, 이를 바탕으로 더 복잡한 특징을 인식

- Tr a n s f e r Learning : 이미 대규모 데이터셋에서 사전 훈련된 모델의 지식을 새로운 문제에 적용하는 방법론으로 이때 낮은 계층의 일반적인 특징은 유지되고, 높은 계층의 특징은 새로운 문제에 맞게 조정
- Capsule Network : 개별 뉴런 대신 "캡슐"이라 불리는 뉴런의 그룹을 사용하여 이미지 내 객체의 다양한 속성과 공간적인 관계를 보다 효과적으로 인식하는 신경망 구조

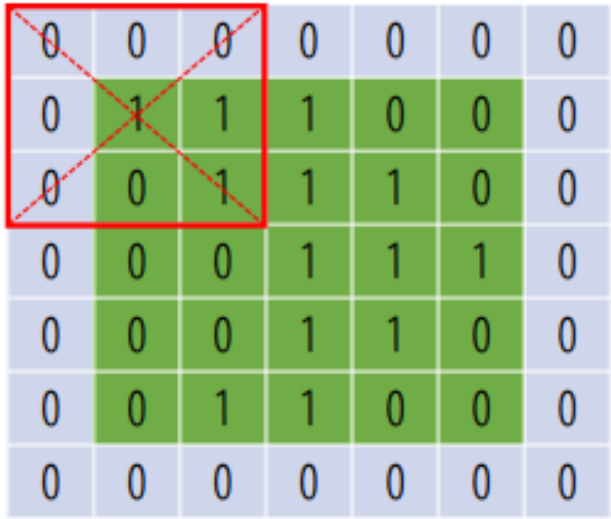


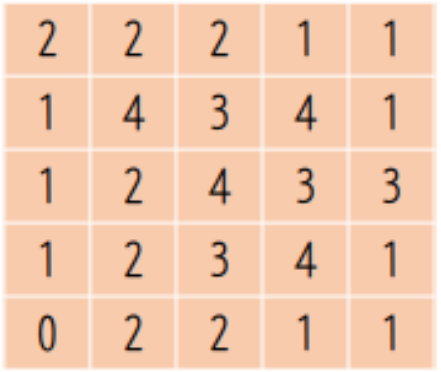
CNN 이해 - Padding

- Convolution Layer에서 Filter를 사용하여 Feature Map을 생성할 때, 이미지 크기가 작아지는 것을 막기 위해 테두리에 Filter 크기를 고려하여 특정 값(일반적으로 0)으로 채우는 작업
- 5X5 이미지에서 3X3 Filter를 사용하면 3X3 크기의 Feature Map이 만들어짐
  즉, 5X5 이미지의 둘레에 0을 채워 7X7의 이미지로 만들어서 3X3 Filter를 적용해 5X5 의 Feature Map을 얻음
- Padding 작업을 통해 인공신경망 이미지 외곽을 인식하도록 하는 효과도 있음 (필수 작업은 아님)



padding 없음



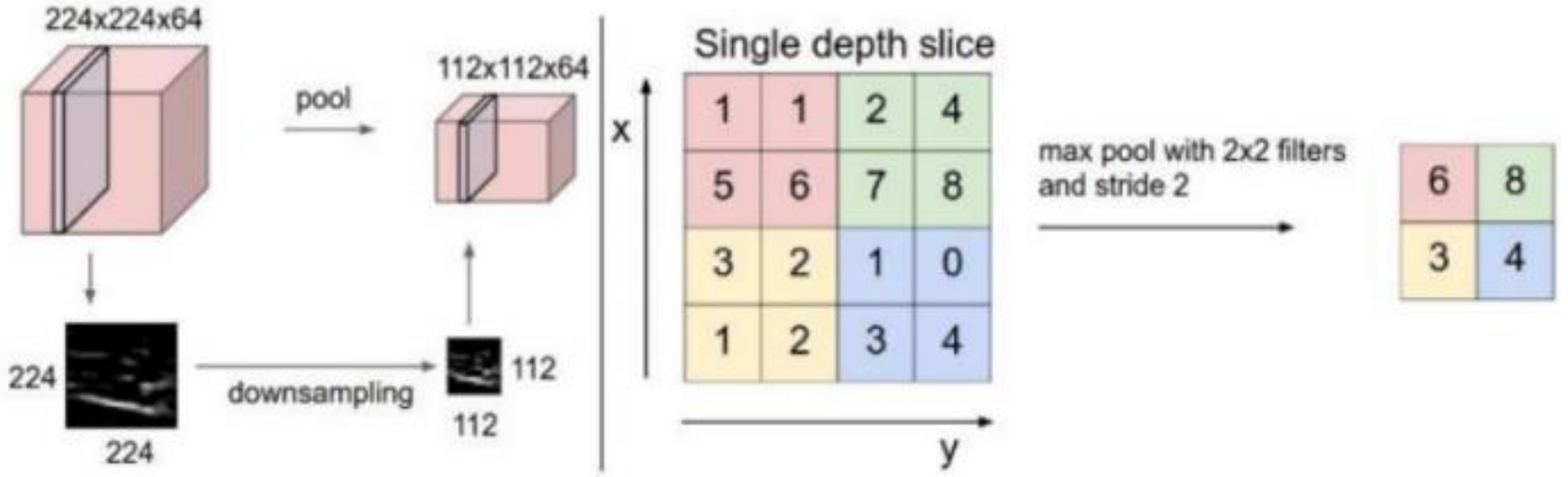


padding 부여

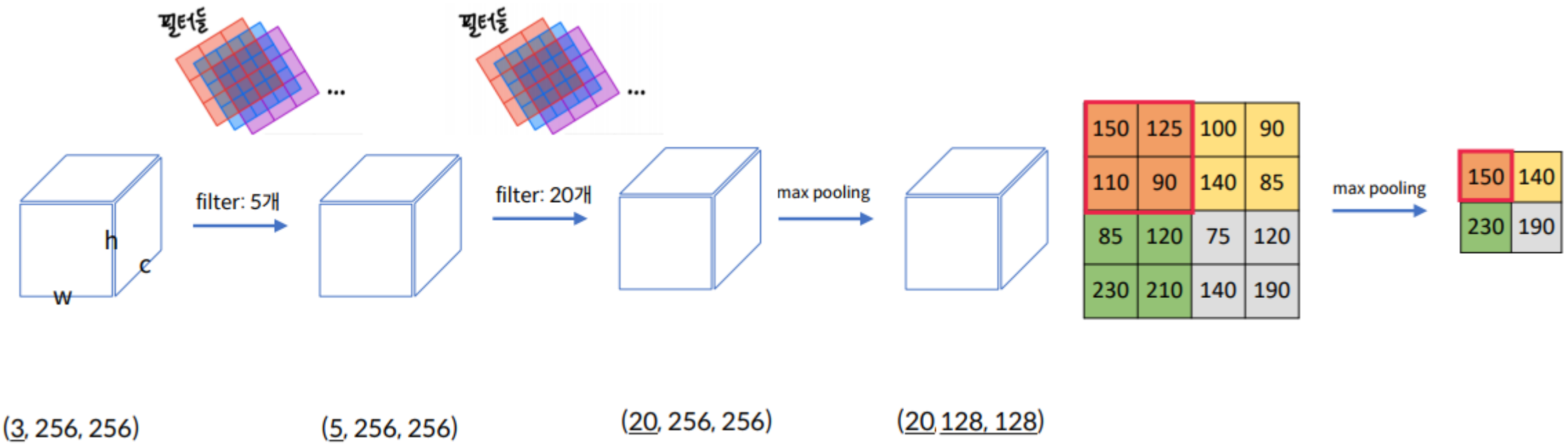
CNN 이해 - Pooling

- Convolution Layer의 Output을 Input으로 받아 Feature Map의 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용
- Max Pooling, Min Pooling, Average Pooling 등의 종류가 있음
- Pooling Size를 Stride로 지정하며, 이 크기에 따라 줄어드는 양이 줄어듬
- 입력 데이터의 행, 열 크기는 Pooling 사이즈의 배수(나누어 떨어지는 수) 이어야 함

https://www.youtube.com/watch?v=U1KiC0AXhHg
https://www.youtube.com/watch?v=f1fXCRtSUWU

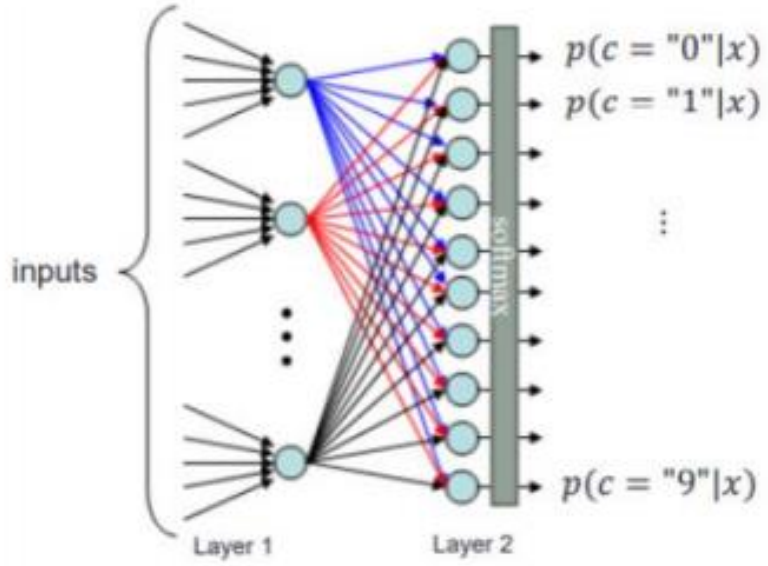


CNN 이해 – Filter & Pooling

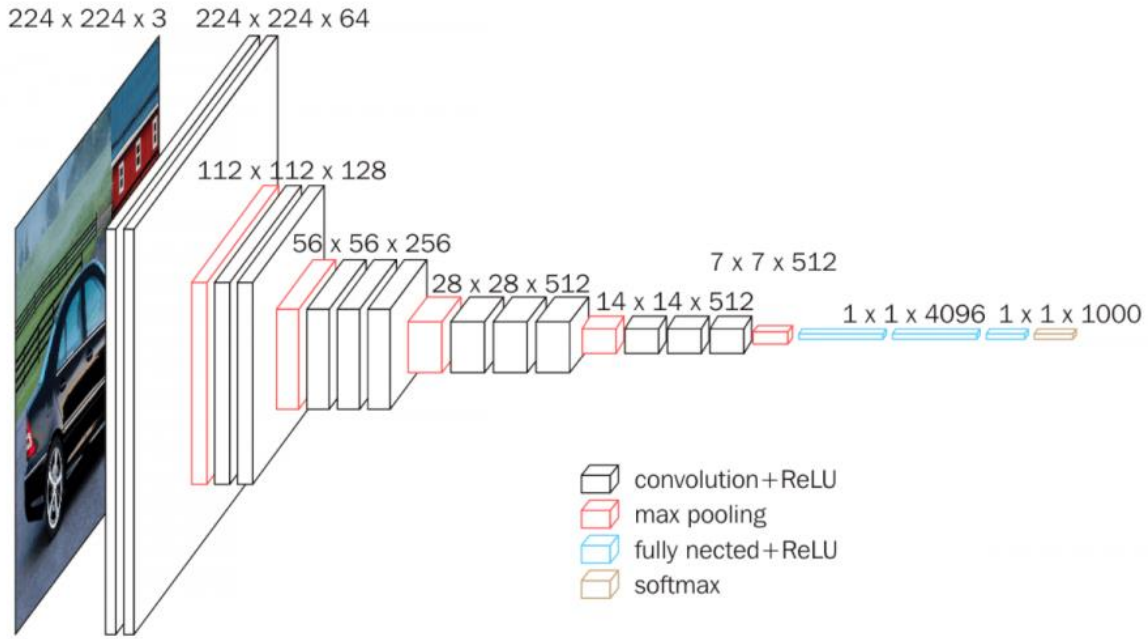


CNN 이해 - Fully Connected Layer

- Flatten Layer : CNN의 데이터를 Fully Connected Neural Network의 형태로 변경하는 Layer
  - 입력 데이터의 Shape 변경만 수행
  - 입력 Shape이 (8, 8, 10)이면 Flatten이 적용된 Shape 은 (640, 1)이 됨
- Softmax Layer : Flatten Layer의 출력을 입력으로 사용하며, 분류 클래스에 매칭 시키는 Layer
  - 분류 작업을 실행해 결과를 얻게됨
  - 입력 Shape이 (640, 1)이고 분류 클래스가 10인 경우 Softmax가 적용된 출력 Shape은 (10,1)이 됨. 이때 weight의 shape은 (10, 640)이며 Softmax Layer의 paramete가 6,400개임

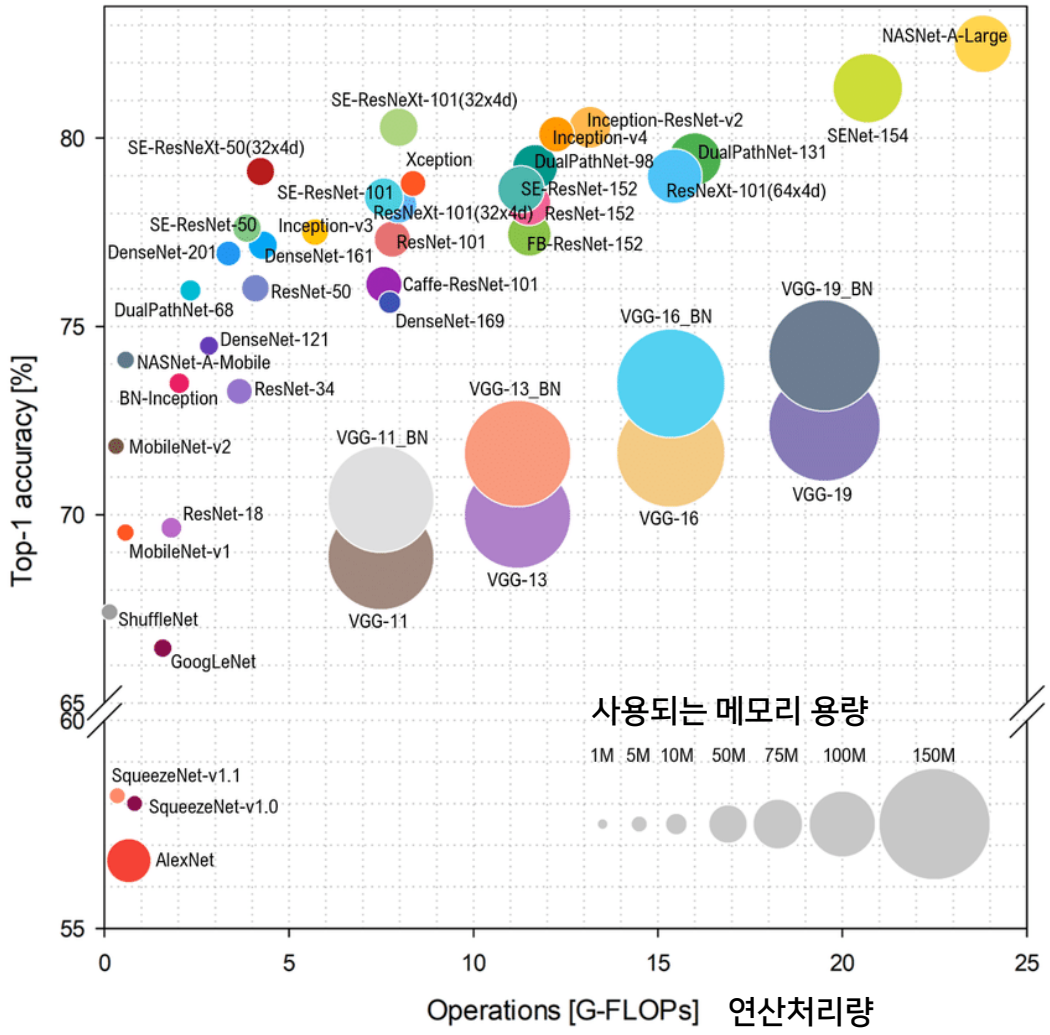


다양한 CNN 네트워크



VGG16 Architecture

출처 : https://neurohive.io/en/popular-networks/vgg16/



출처 : https://www.researchgate.net/figure/Ball-chart-reporting-the-Top-1-and-Top-5-accuracy-vs-computational-complexity-Top-1-and_fig1_328509150



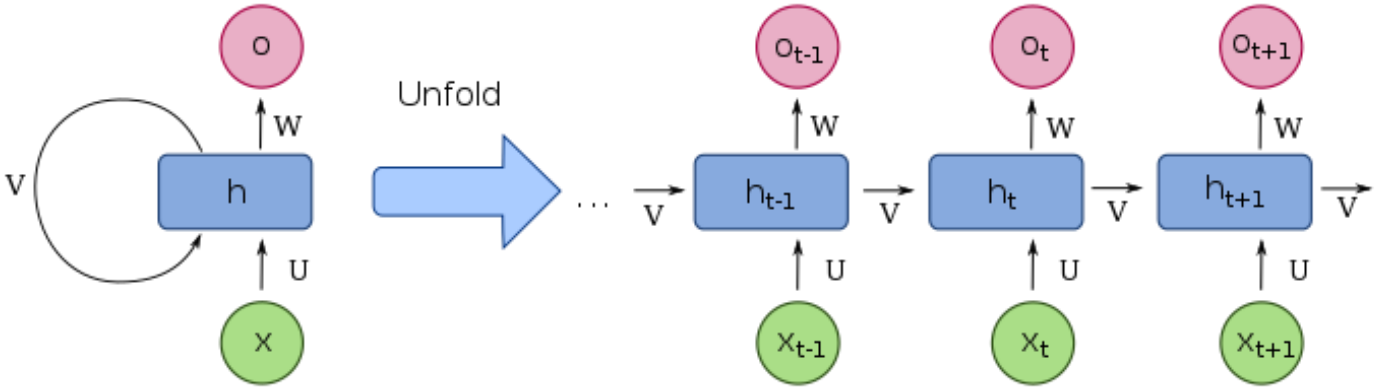

PART4. RNN 대한 이해

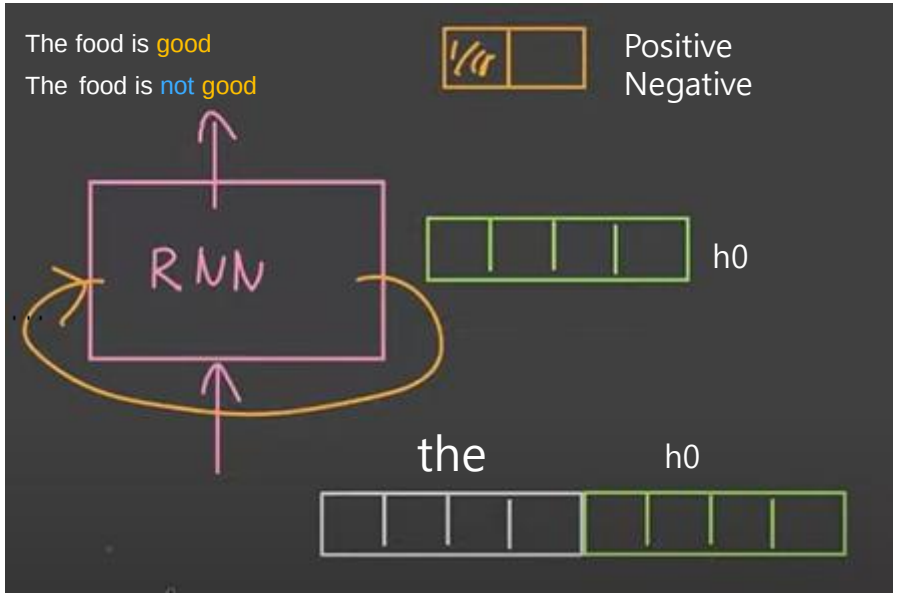
Enjoy your possibility

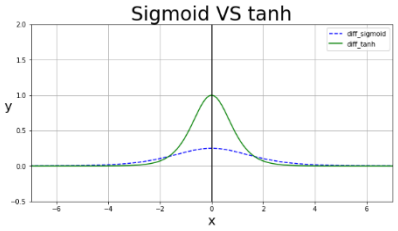
Tensorflow-keras 로 RNN 알고리즘 구현하기

RNN

- 입력과 출력을 시퀀스 단위로 처리하는 시퀀스(Sequence) 모델





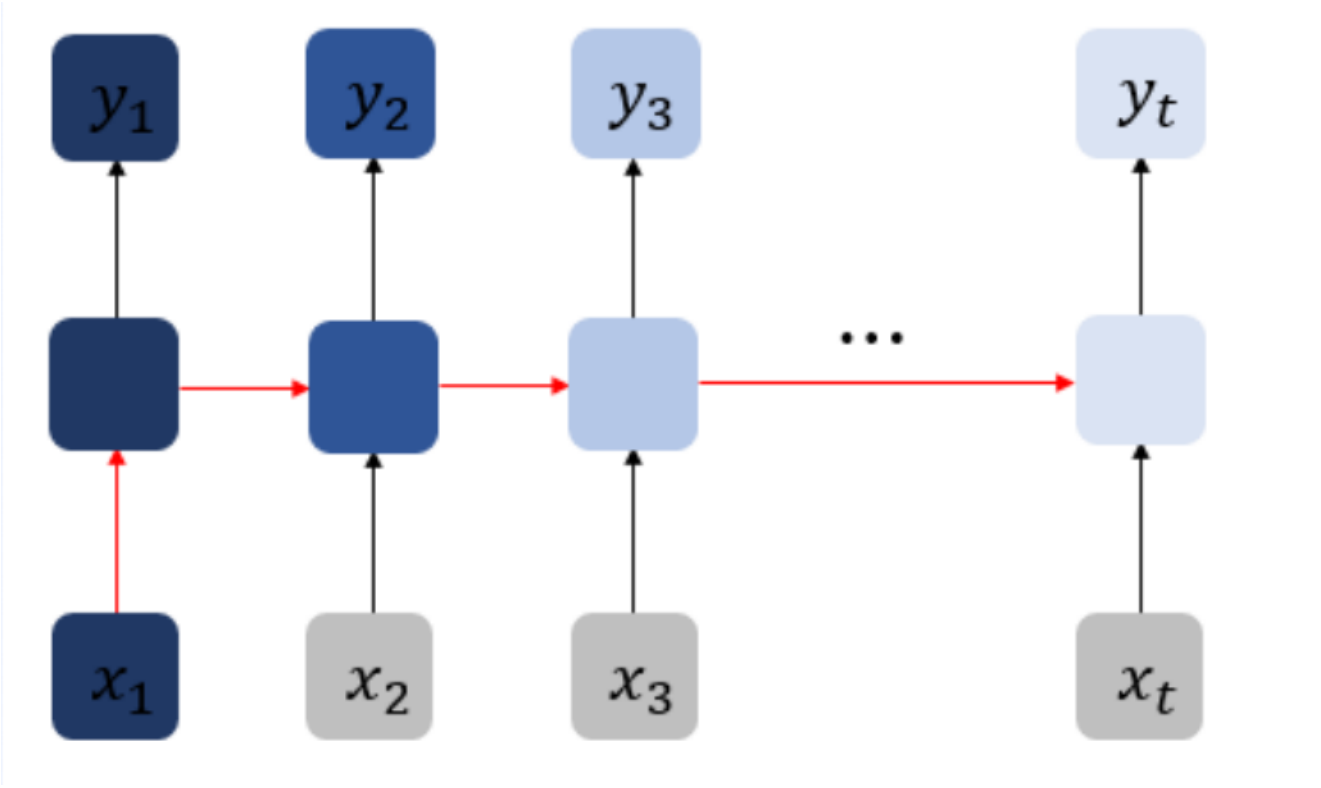


$$\tanh\left(W_h \times h_{t-1} + W_x \times x_t + b\right) = h_t$$

$D_h \times D_h$ $D_h \times 1$ $D_h \times d$ $d \times 1$ $D_h \times 1$ $D_h \times 1$

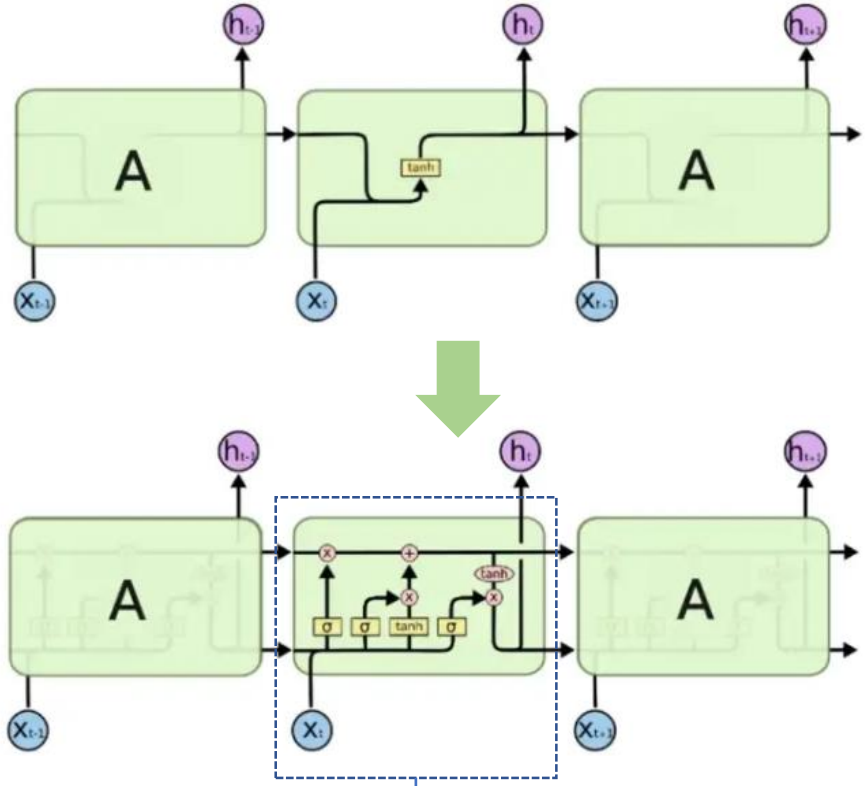
RNN의 장기의존성 문제

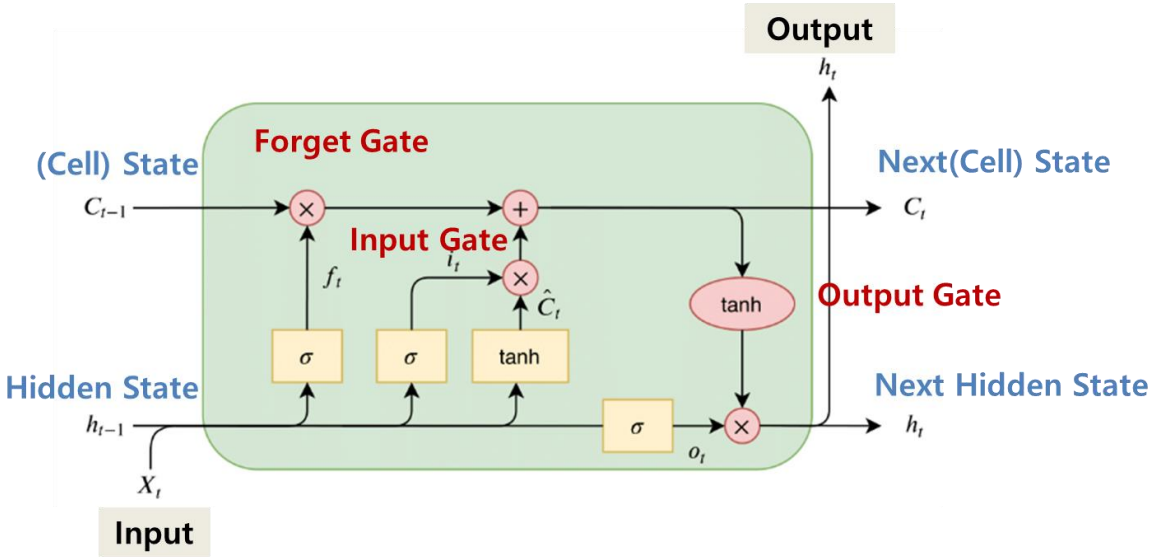
- RNN은 시점이 길어지면서 앞에 있던 정보가 소실되는 '장기 의존성 문제' 를 갖고 있음
  즉, X1의 정보량을 짙은 남색으로 표현하였을 때 뒤로 갈 수록 색이 얕아지는 것처럼 점점 정보가 소실되는되는 문제점



LSTM(Long Short-Term Memory)

- 기존 RNN의 장기 의존성 문제를 개선하여 기억력을 높인 RNN의 명칭임





게이트순환유닛(Gated Recurrent Unit, GRU)

- 뉴욕대학교 조경현 교수님이 집필한 논문에서 제안한 것으로 LSTM의 장기 의존성 문제에 대한 해결책을 유지하면서, 은닉 상태를 업데이트하는 계산을 줄임(즉, GRU는 성능은 LSTM과 유사하면서 복잡했던 LSTM의 구조를 간단화 시킴)