2주차 스터디

딥러닝 CNN 완벽 가이드 – 섹션4, 5, 6

섹션4 – CNN의 이해

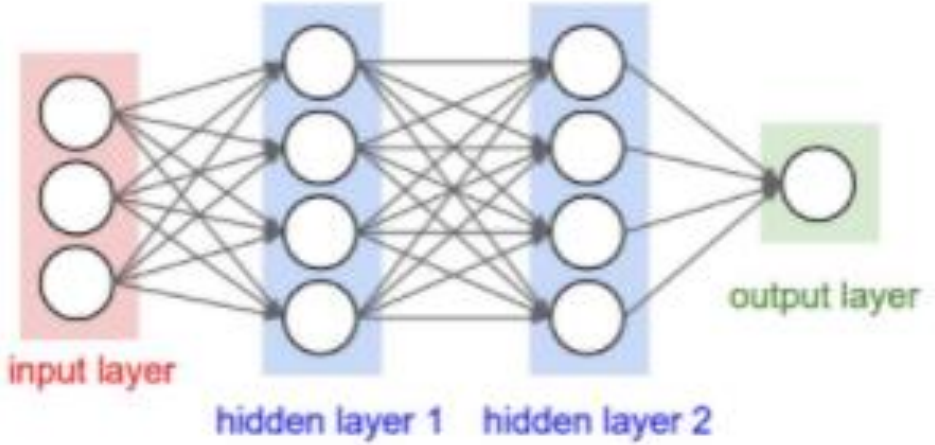
섹션5 – CNN 모델 구현 및 성능 향상 기본 기법 적용하기

섹션6 – 데이터 증강의 이해 – Keras ImageDataGenerator 활용

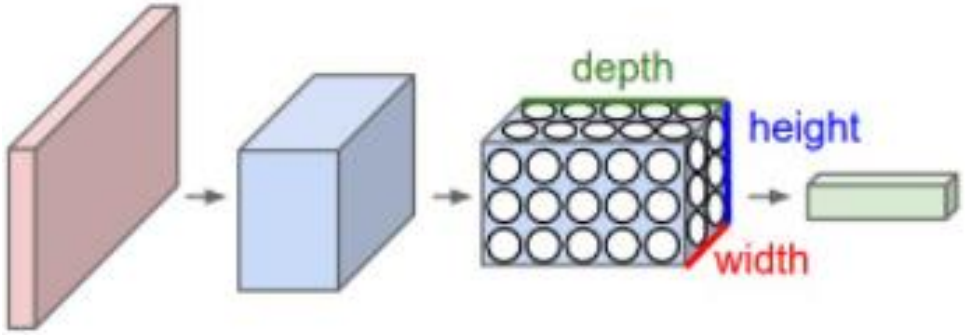
발표자 : 김다희, 이준규

1. CNN

- **Dense Layer 기반 Image 분류**
  - 이미지의 특성 -> 고려 못함
  - 이미지의 크기 -> 너무 많은 weight 필요

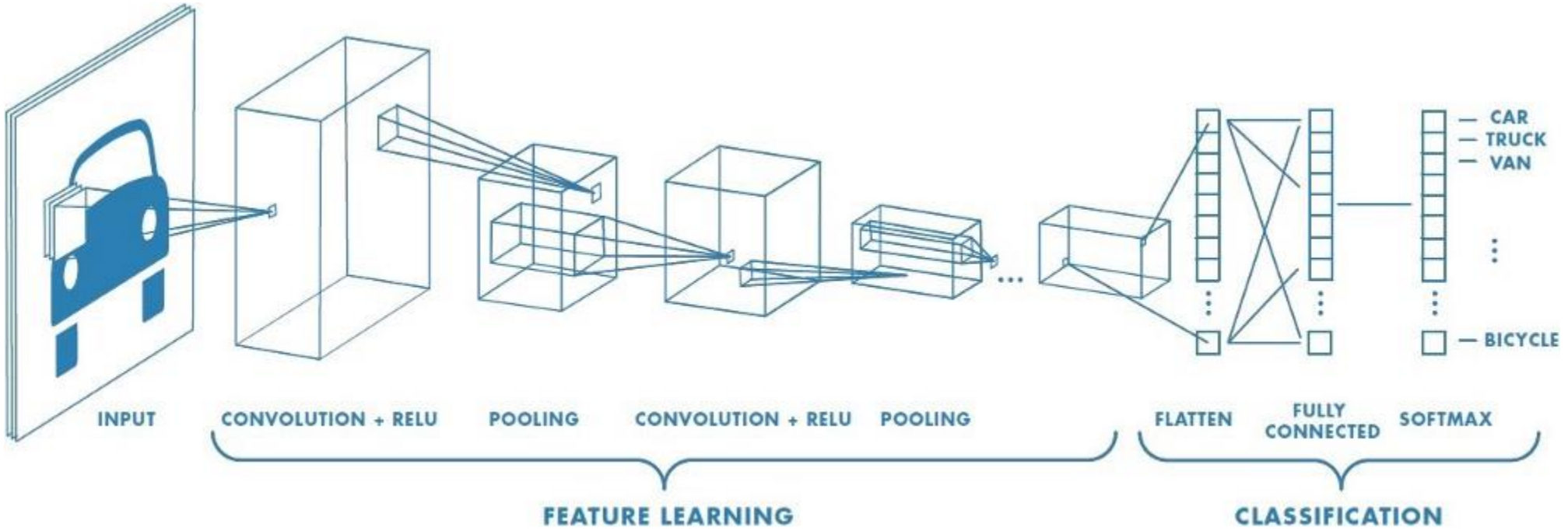


- **Feature Extraction 기반 Image 분류**
  - CNN: Feature Extraction + Classification
  - Feature Extraction: Convolution + Pooling



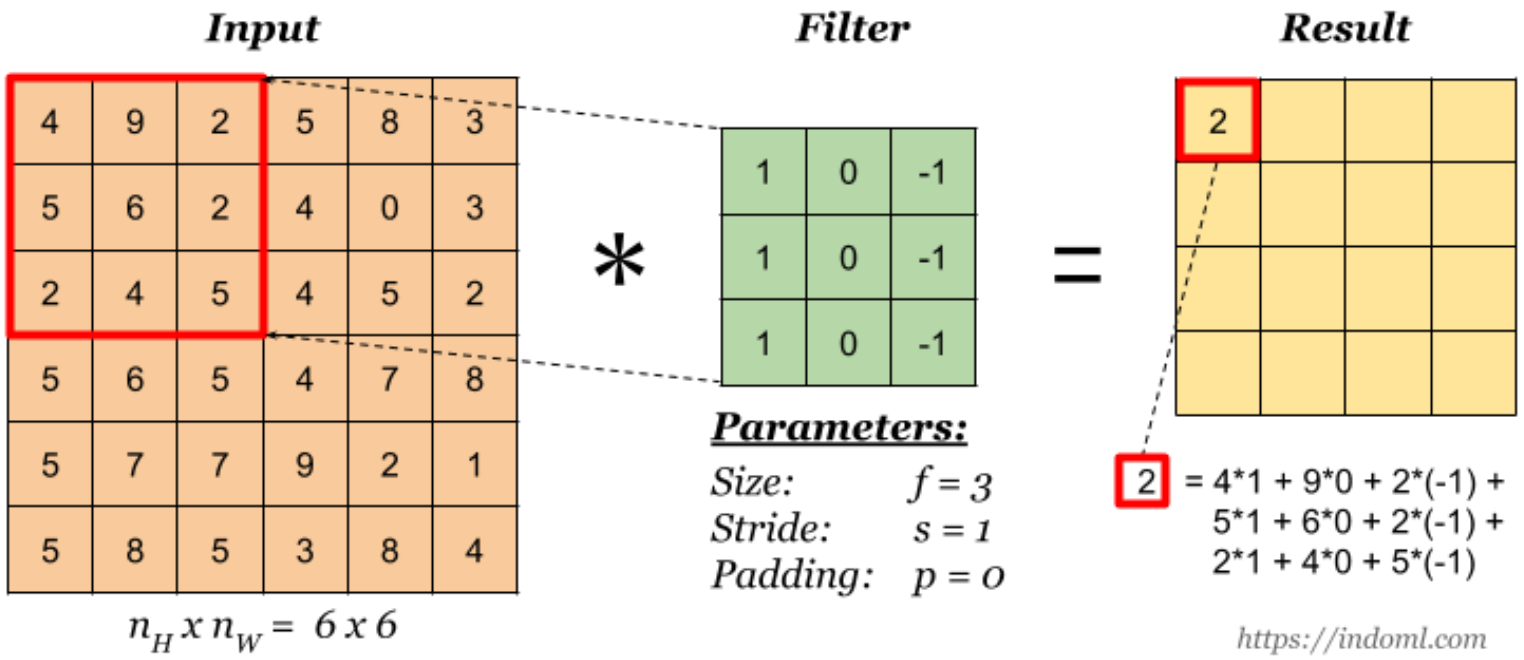
1. CNN

- CNN의 구조


CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION

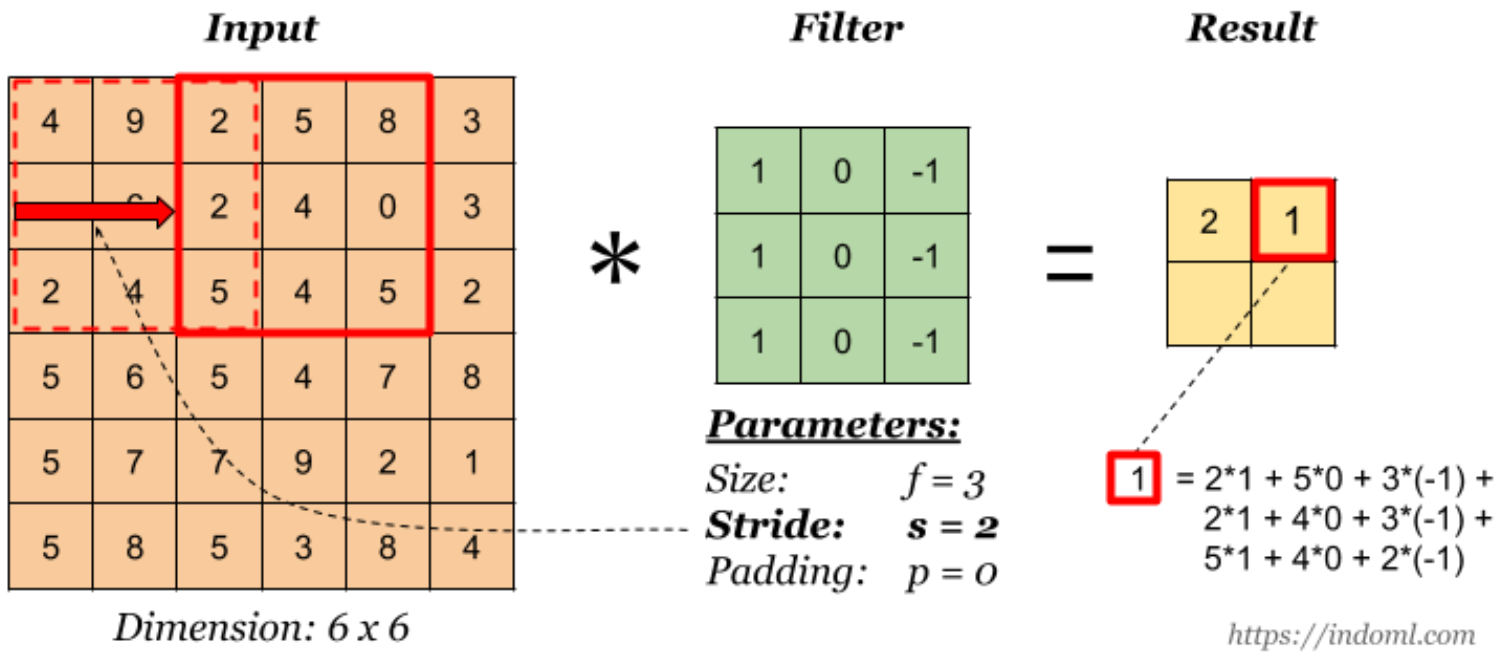
2. Feature Extraction

- Convolution
  - sliding 하면서 filter를 이용해 합성곱 연산 순차적으로 실행



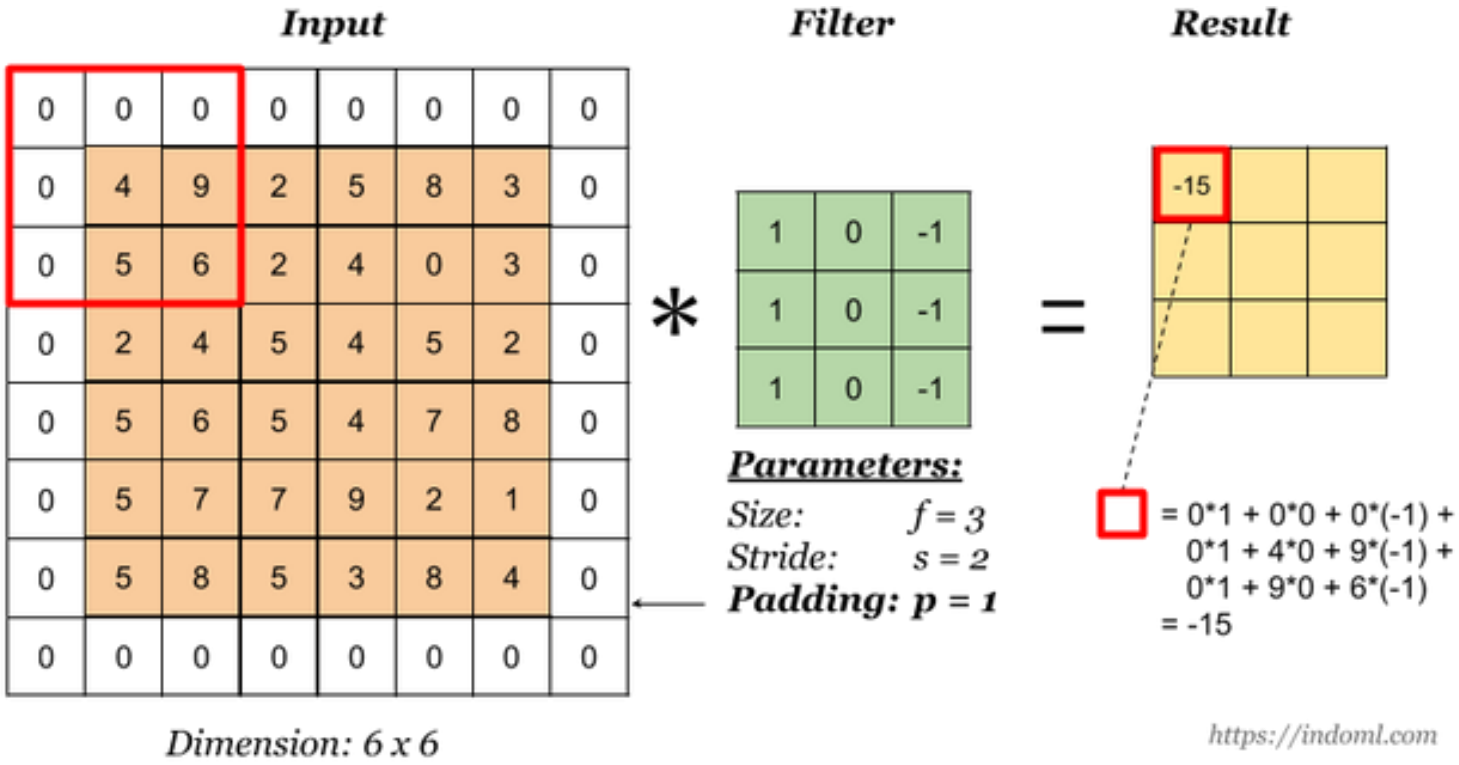
2. Feature Extraction

- **Stride**
  - sliding window가 이동하는 간격



2. Feature Extraction

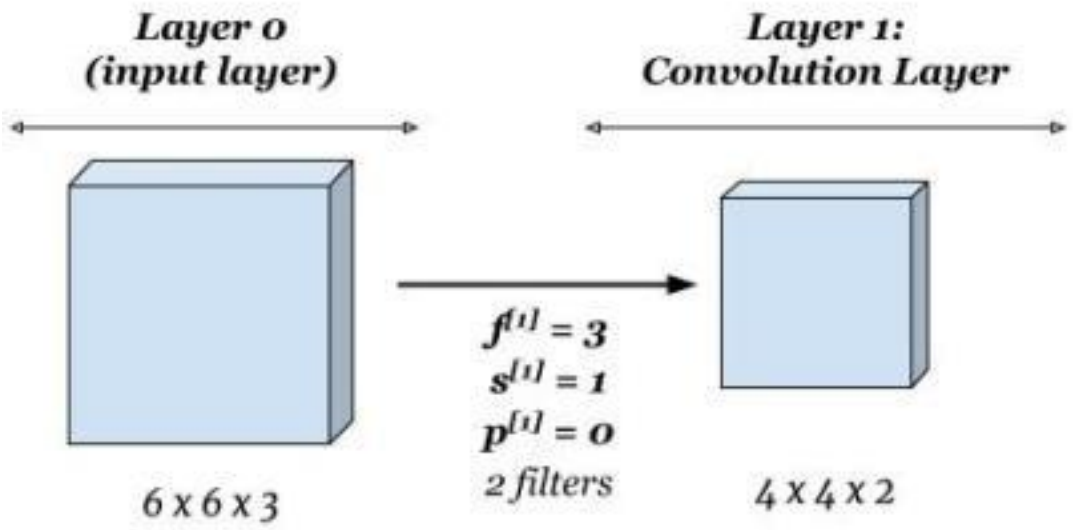
- **Padding**
  - Input을 0으로 둘러싸서 이미지 크기를 키워주는 작업



2. Feature Extraction

- **Feature Map**

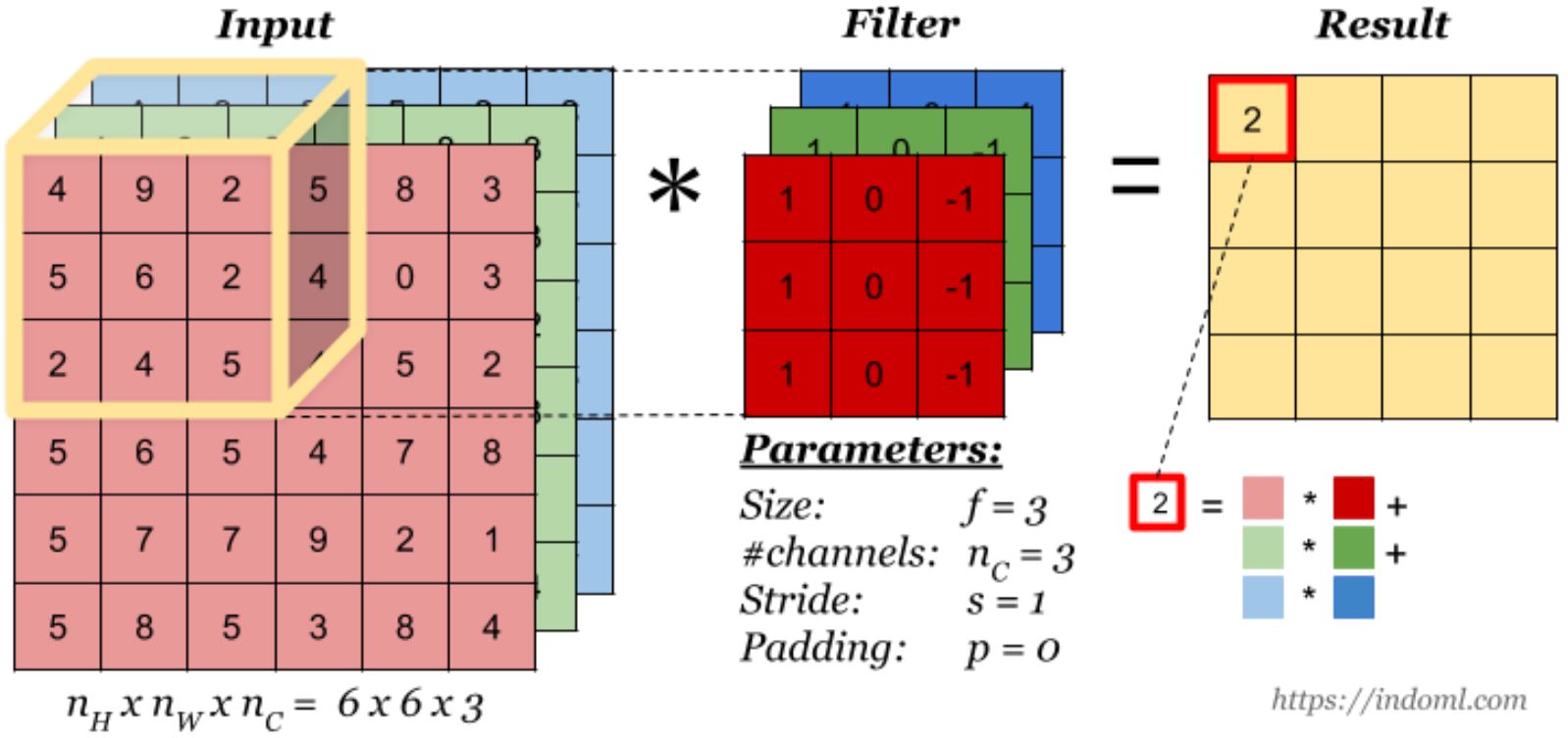
- Input * Convolution = Feature Map
- 피처맵 크기 공식: O = (I – F +2P) / S + 1



2. Feature Extraction

- **Filter**

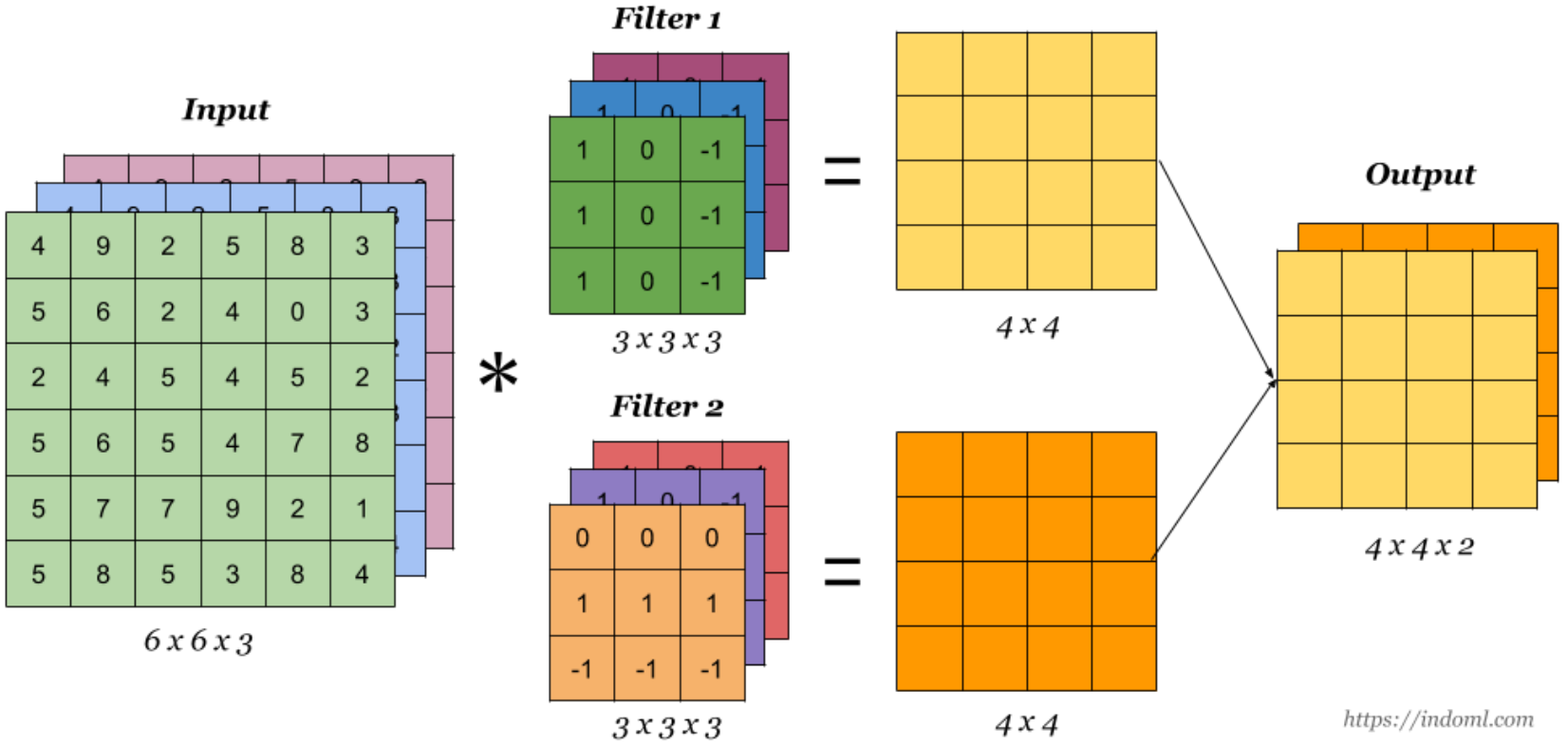
- Filter: 여러 개의 Kernel(거의 혼용해서 씀) -> 1개의 필터는 3차원
- 특징 추출을 위해 사용되는 파라미터 -> 우리가 선택하는 게 아니라 스스로 최적화
- 입력의 차원 = 필터의 채널 수



2. Feature Extraction

- **Channel**

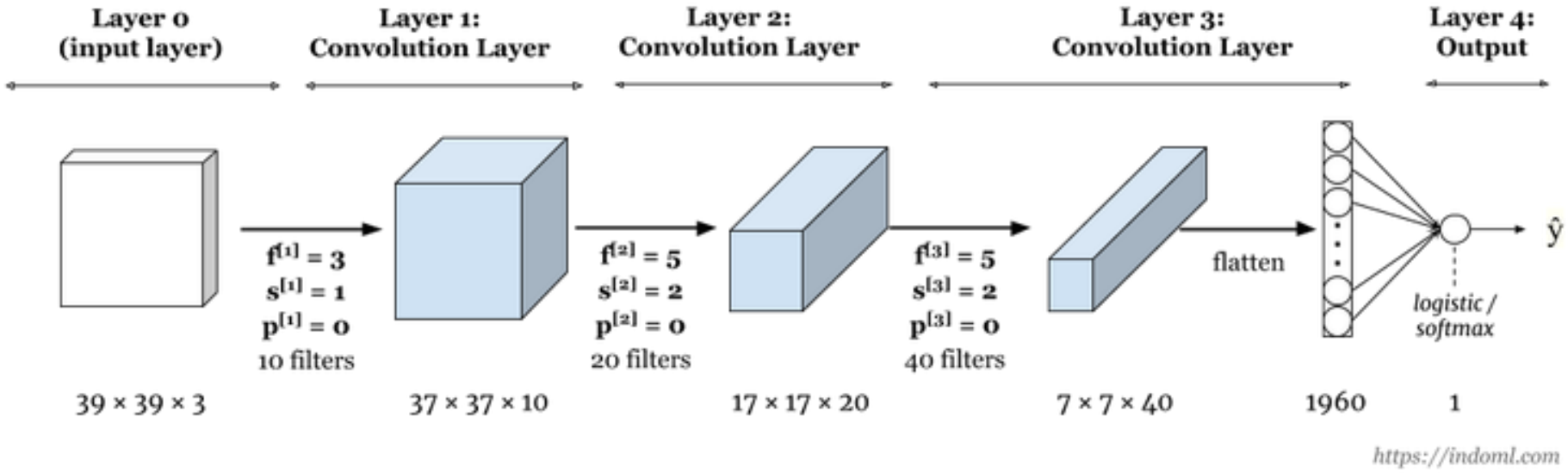
- # of filter = # of channel
- 채널별로 합성곱 연산을 실시해 채널 별 feature map 생성 -> 이를 합산한 feature map을 activation map이라고 함



2. Feature Extraction

- **차원 정리**

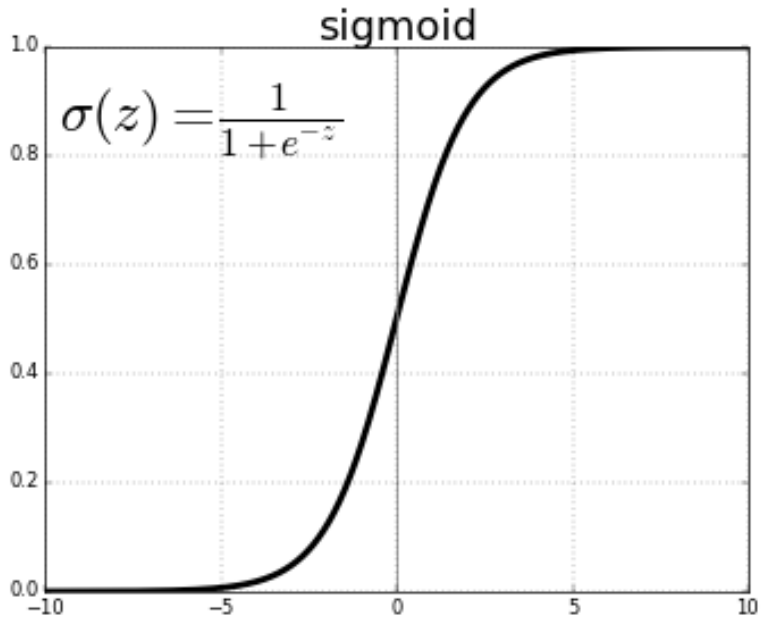
- Input image: 3차원 / Filter : 3차원 / Feature map: 3차원
- 여기서 말하는 3차원은 배치 크기를 제외한 3차원
- Conv 연산을 적용할 필터의 채널 수 = 입력 피처맵의 채널 수
- Conv 연산을 적용한 필터의 개수 = 출력 피처맵의 채널 수
- 즉, 피처맵의 채널 수는 Conv를 적용한 필터의 개수로 결정 -> 단일 필터의 채널 수가 아님

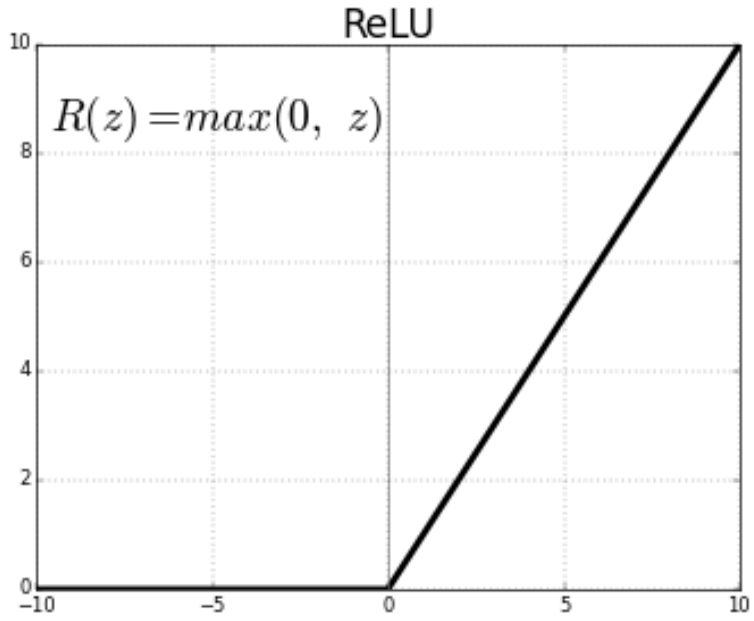


2. Feature Extraction

- Activation Function

- 추출된 feature map에 activation function을 적용

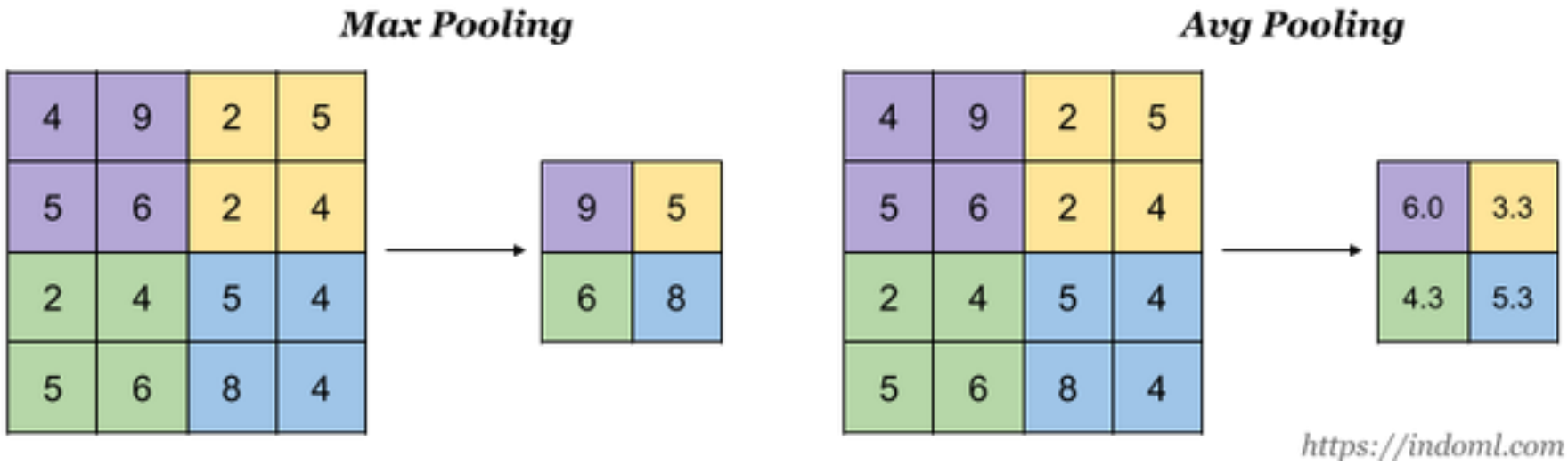




2. Feature Extraction

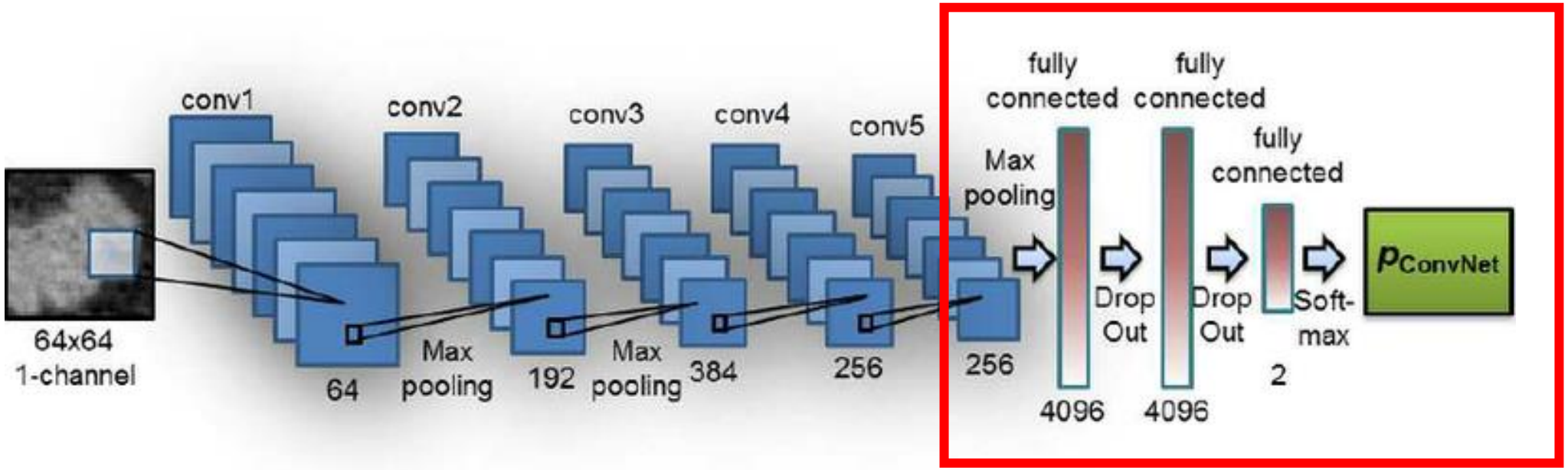
- **Pooling**

- Feature map의 일정 영역 별로 하나의 값 추출 -> 연산 중복을 방지하기 위해
- Pooling에서 학습할 파라미터는 따로 없음
- 최근 CNN은 pooling은 최대한 자제하고 stride를 이용하는 추세



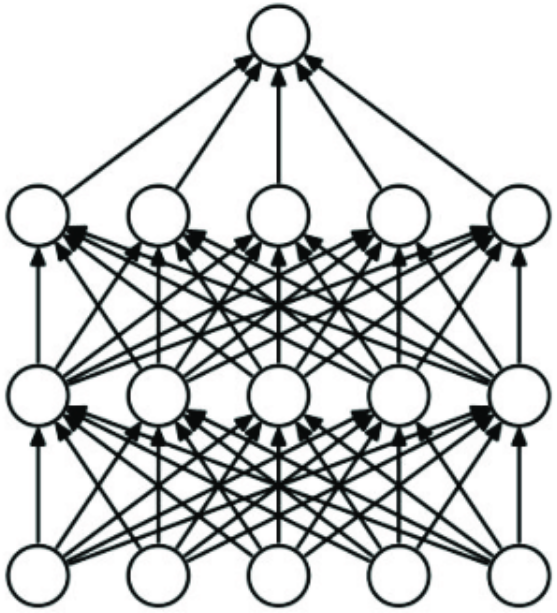
3. Image Classification

- **Fully Connected Layers**
  - Flatten()을 이용해 feature map을 1차원으로 펼침
  - FC는 Convolution Layer에서 추출한 특징을 기존 신경망에 적용해 분류하는 과정

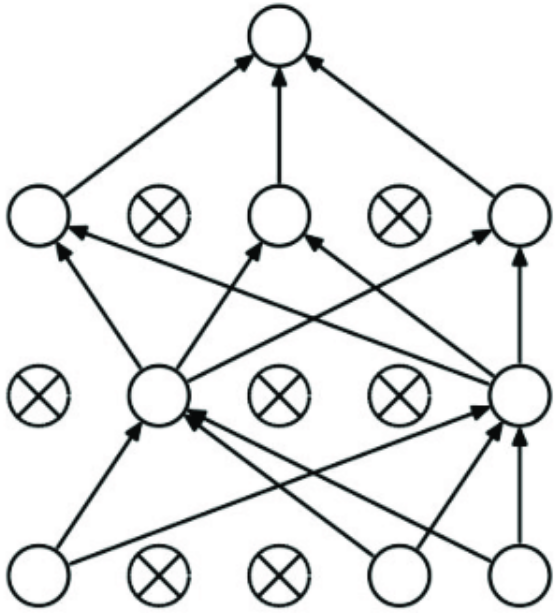


3. Image Classification

- Dropout
  - 과적합을 막기 위해 랜덤으로 노드 삭제



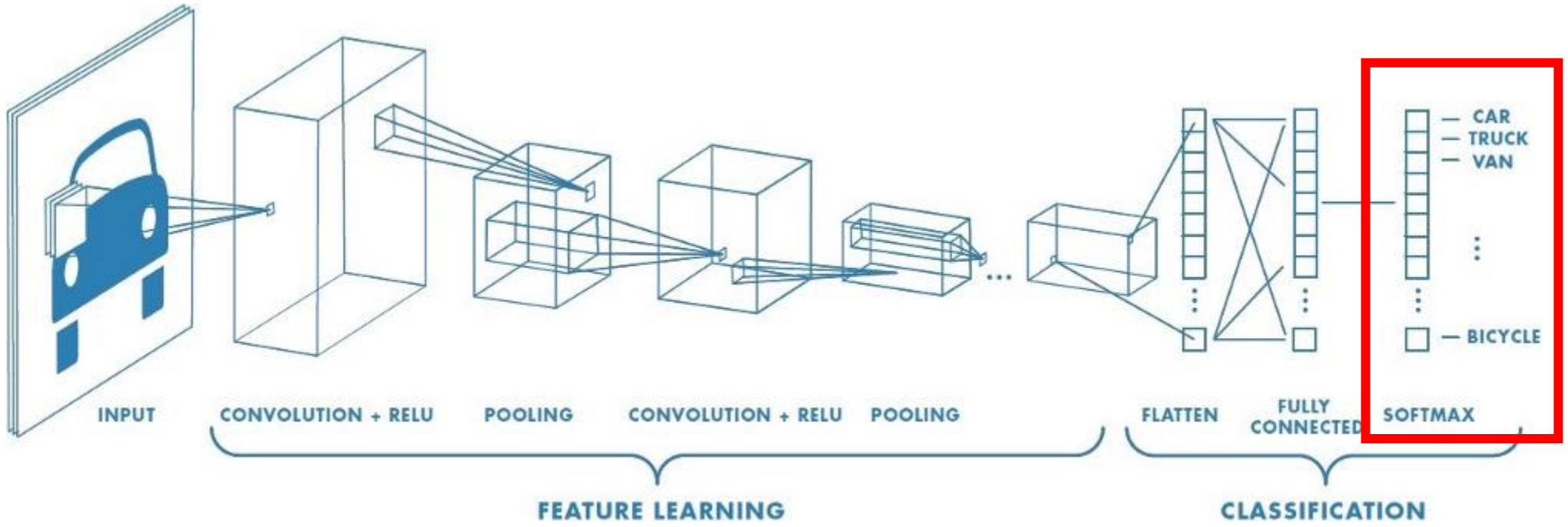
(a) Standard Neural Network



(b) Neural Net with Dropout

3. Image Classification

- Softmax
  - 최종적으로 Softmax 활성화 함수를 통과한 후 가장 큰 값으로 분류 대상 선정



4. Code

- **Input 차원의 이해**

- Input(): 배치 크기를 제외한 3차원을 입력으로 받음(keras에서 내부적으로 알아서 4차원으로 배치)

```
input_grey = Input(shape=(28, 28, 1)) # greyscale
input_rgb = Input(shape=(28, 28, 3)) # RGB
```

- **Padding 옵션**

- padding = 'same': 자동으로 입력 feature map과 출력 feature map의 크기를 맞춤
- padding = 'valid': 별도의 padding 수행 X

```
x = Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu')(x)
x = Conv2D(filters=32, kernel_size=3, strides=1, padding='valid', activation='relu')(x)
```

4. Code

- ## CNN 모델 생성

- 하나의 필터와 피처맵은 무조건 3차원
- Conv2D에서 말하는 3차원은 배치를 제외한 3차원
- 여러 개의 커널 = 필터 -> 필터의 채널 수는 keras가 자동적으로 맞춰 줌

```
input_tensor = Input(shape=(28, 28, 1))
x = Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu')(input_tensor)
x = Conv2D(filters=64, kernel_size=3, activation='relu')(x)
x = MaxPooling2D(2)(x)
x = Flatten()(x)
x = Dense(100, activation='relu')(x)
x = Dropout(0.4)(x)
output = Dense(10, activation='softmax')(x)

model = Model(inputs=input_tensor, outputs=output)
model.summary()
```

4. Code

- **CNN 모델 해석**

- Params: 학습할 weight의 개수
- Params = 3*3*32*64+64 = 18496(개)
- 3*3*32: 하나의 필터(3차원)
- *64: 필터의 개수
- +64: bias

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 26, 26, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 10816)             0
_________________________________________________________________
dense_9 (Dense)              (None, 100)               1081700
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1010
=================================================================
Total params: 1,101,526
Trainable params: 1,101,526
Non-trainable params: 0
_________________________________________________________________
```

* 가중치 초기화의 이해와 적용

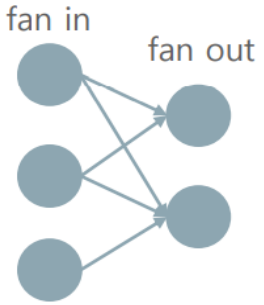
•좋은 가중치 초기화 조건

1.값이 동일 해서는 안됨
2.충분히 작아야 한다
3.적당한 분산(표준편차)를 가져야한다.

→
1) 모든 입력 노드에서 온 데이터가 동일한 가중치로 계산이 되면, 결국은 입력 값이 같은 비율로 변화한다는 것을 말합니다.
2)전체 식에서 가중치가 크면 곱 합을 진행했을 때 너무 큰 수가 나오게 되는데 그렇게 되면 마지막에 활성화함수를 통과할 때, 1로 수렴하는 현상이 일어날 수 있다. 정규분포나 균일분포 등에서 추출하는 방법이 있다.
3)1)과 비슷한 이유로 적당한 표준편차를 가져야 효율적인 학습이 진행된다.

* 가중치 초기화의 이해와 적용

•Xavier Glorot Initialization
fan in, fan out을 고려하여 동적으로 초기화



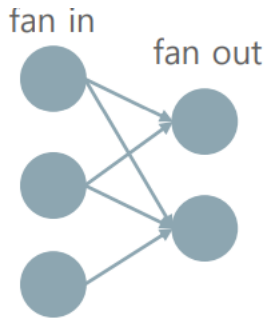
입력 노드와 출력 노드의 갯수를 감안하여 동적으로 Weight 초기화 수행.

glorot_uniform	Uniform Distribution(-한도값, +한도값)	**한도값** = $\sqrt{\frac{6}{fan\ in + fan\ out}}$
glorot_normal	Normal Distribution(평균=0, 표준편차)	**표준편차** = $\sqrt{\frac{2}{fan\ in + fan\ out}}$

→ 노드가 여러 개일 수록 곱 합이 가중치의 변화에 따라 더 크게 변화한다.
동일한 값들이 많거나 편차가 큰 값이 많을 때, 두 경우 모두 문제가 발생한다.
따라서 노드의 수에 따라 동적으로 값을 조정하는 방법이다.

* 가중치 초기화의 이해와 적용

•He Initialization
전반적으로 std가 Xavier보다 크다.



Relu에 보다 최적화된 가중치 초기화

he_uniform	Uniform Distribution(-한도값, +한도값)	**한도값** = $\sqrt{\frac{6}{fan\ in}}$
he_normal	Normal Distribution(평균=0, 표준편차)	**표준편차** = $\sqrt{\frac{2}{fan\ in}}$

* 가중치 초기화의 이해와 적용

•실습

```
input_tensor = Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3))

#x = Conv2D(filters=32, kernel_size=(5, 5), padding='valid', activation='relu')(input_tensor)
x = Conv2D(filters=32, kernel_size=(3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(input_tensor)
x = Conv2D(filters=32, kernel_size=(3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

x = Conv2D(filters=64, kernel_size=3, padding='same', activation='relu', kernel_initializer='he_normal')(x)
x = Conv2D(filters=64, kernel_size=3, padding='same', kernel_initializer='he_normal')(x)
x = Activation('relu')(x)
x = MaxPooling2D(pool_size=2)(x)

x = Conv2D(filters=128, kernel_size=3, padding='same', activation='relu', kernel_initializer='he_normal')(x)
x = Conv2D(filters=128, kernel_size=3, padding='same', activation='relu', kernel_initializer='he_normal')(x)
x = MaxPooling2D(pool_size=2)(x)
```

kernel_initializer='he_normal'로 변경이 가능한데, 변경을 해도 성능에 큰 차이가 없다는 것을 알 수 있다. → BN을 적용하면 큰 차이가 없다.

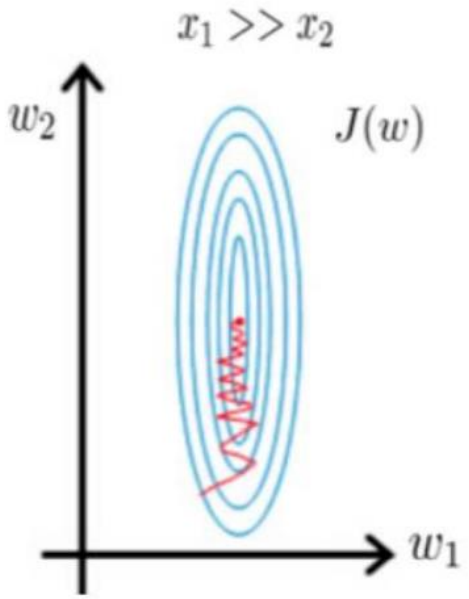
* Feature Scaling

서로 다른 Feature를 동일한 척도로 변환 : 나이, 월급, 몸무게 등은 서로 간의 값 차이가 유의미한 것은 아니다.

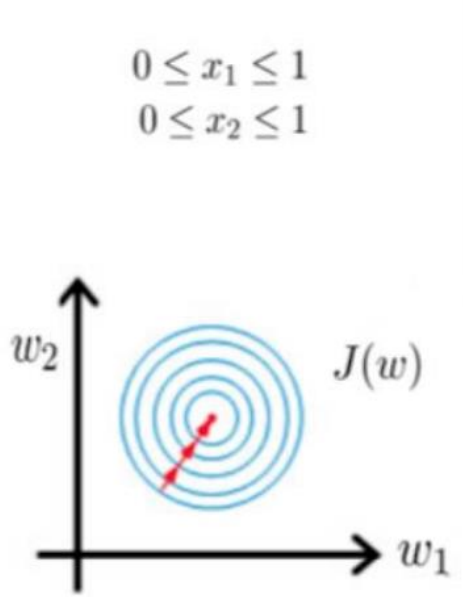
1.Min-Max : 0에서 1사이로 변환 (X-Min)/(Max-Min)
2.Z Score : 평균이 0, 표준편차가 1인 데이터 세트로 변환 X-Mean/std → 0 centered 로 만듦.

Feature x1, x2
w1은 x1의 가중치
w2는 x2의 가중치

Scaling을 적용하지 않은 경우



Scaling을 적용한 경우

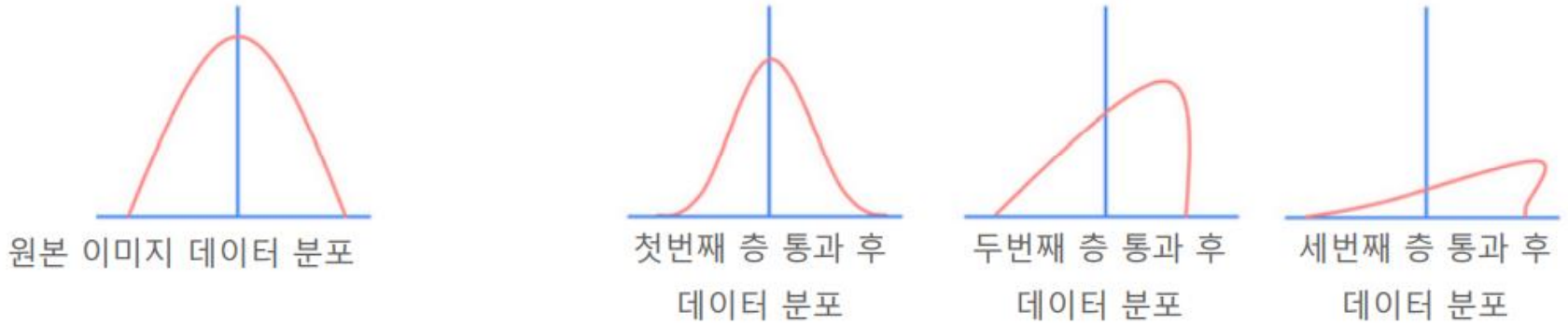


그림을 보면 왼쪽은 x1이 x2보다 크기에 w1은 상대적으로 작아지고 x2가 작기에 w2가 상대적으로 커지게 된다. 이렇게 되면 w2로만 loss를 줄이는 방식으로 학습이 되는 문제가 발생할 수 있다. 따라서 오른쪽처럼 scaling을 적용하여 학습한다.

Linear Regression: y = x1w1 + x2w2 + ...

* Batch Normalization 이해와 적용

신경망의 각 층을 통과할 때 마다 데이터의 분포가 조금씩 변경되는데,
이것이 누적되다 보면 처음 입력층과 마지막 층의 데이터 분포가 굉장히 많이 달라져 학습에 어려움이 생긴다.



→ 데이터 분포의 조절이 필요하다.

이해) 특정 가중치가 갑자기 튈 때 다음 층에서 그 값을 조금 덜 튀게 조정
해준다.
"너는 이 범위에서 놀았으면 좋겠다"

* Batch Normalization 이해와 적용

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

e : 0이 되는 것을 막기 위해 아주 작은 갑. 입실론

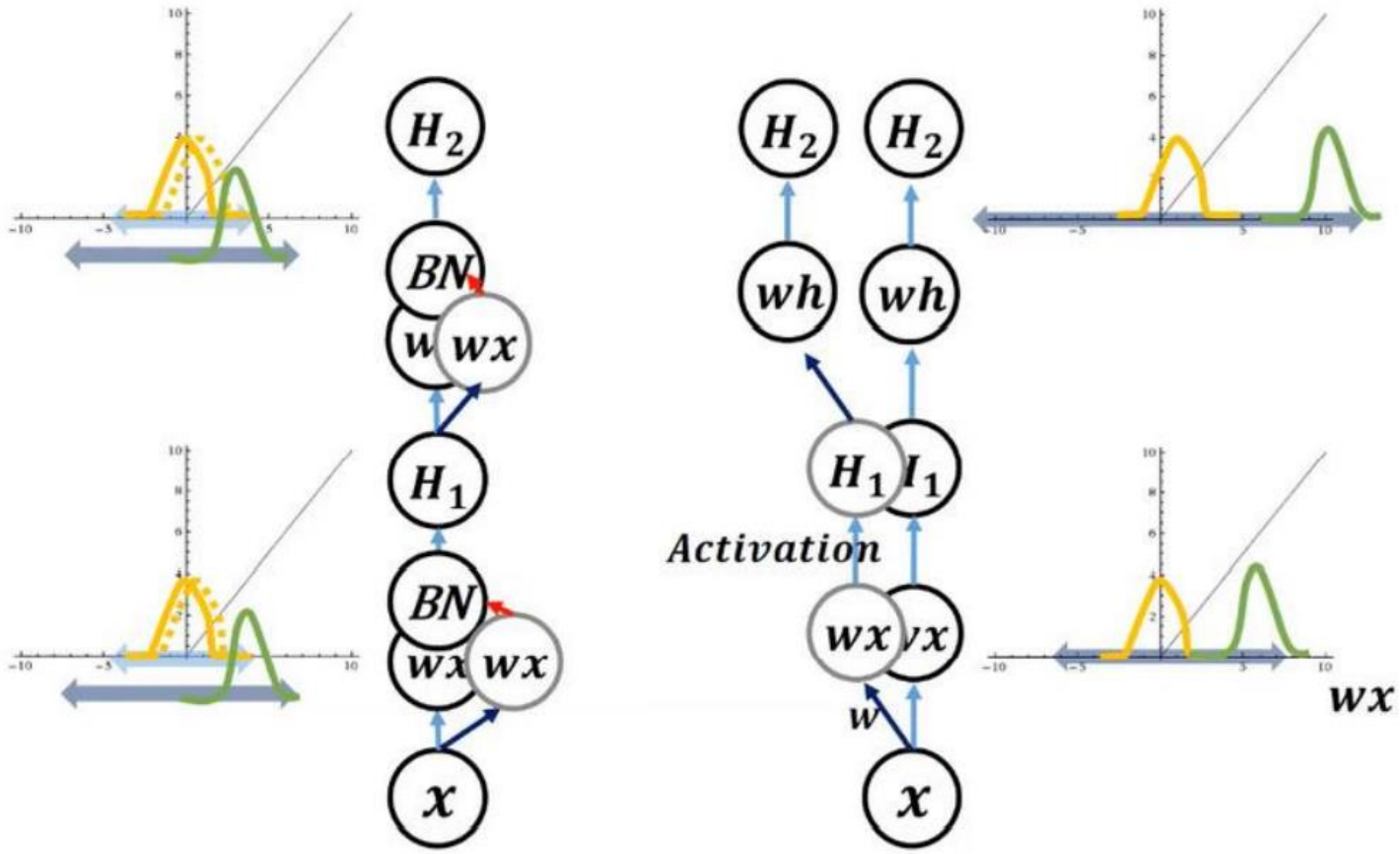
감마 : 스케일링 파라미터

B : Shift 파라미터

y = 감마 * x^ + B : 너무 타이트한 정규화 문제를 해결하기 위해
(많은 값이 0~1로 정규화 하면 0.5로 수렴하는 현상 발생)

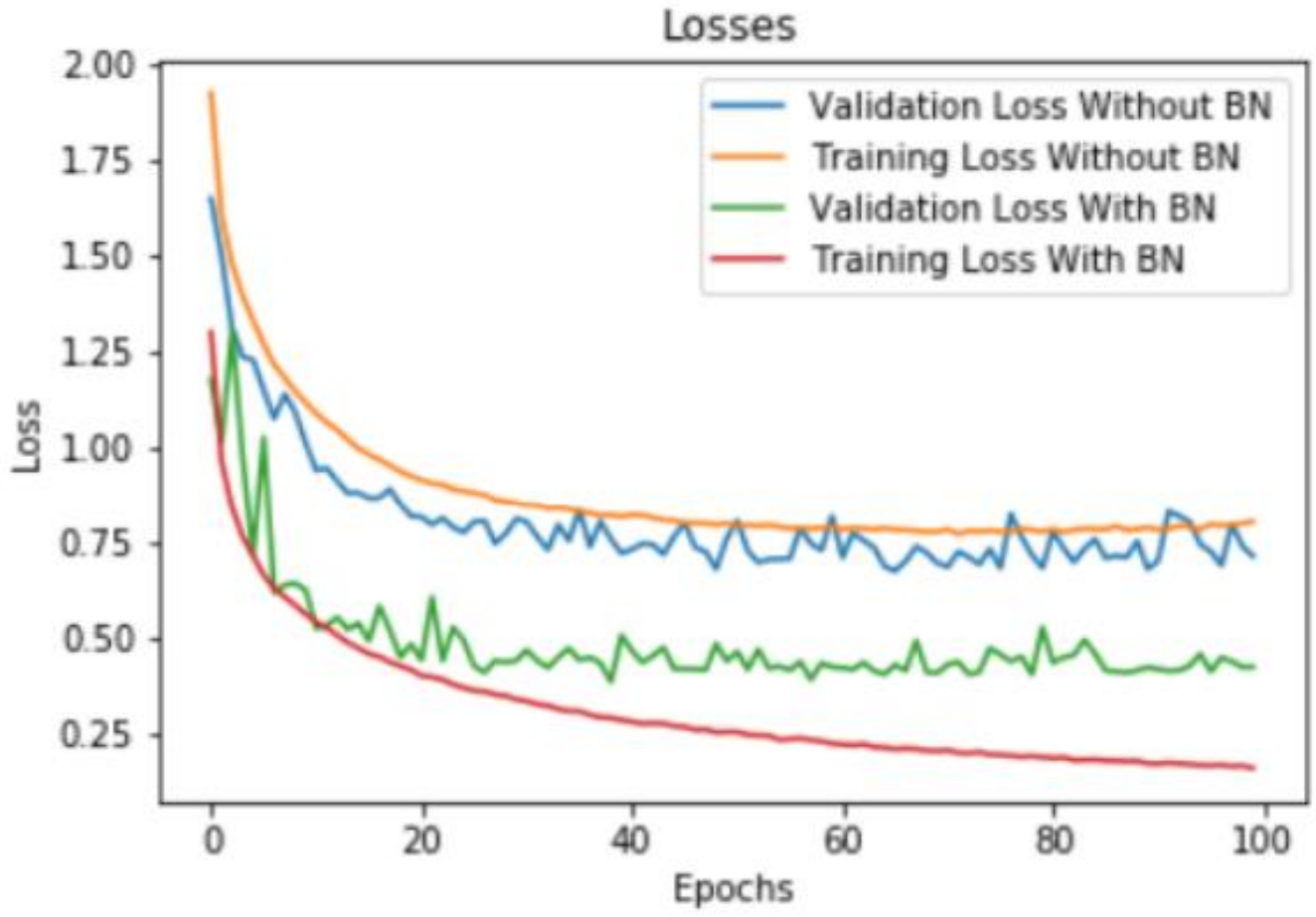
* Batch Normalization 이해와 적용

전반적인 흐름



* Batch Normalization 이해와 적용

BN의 효과



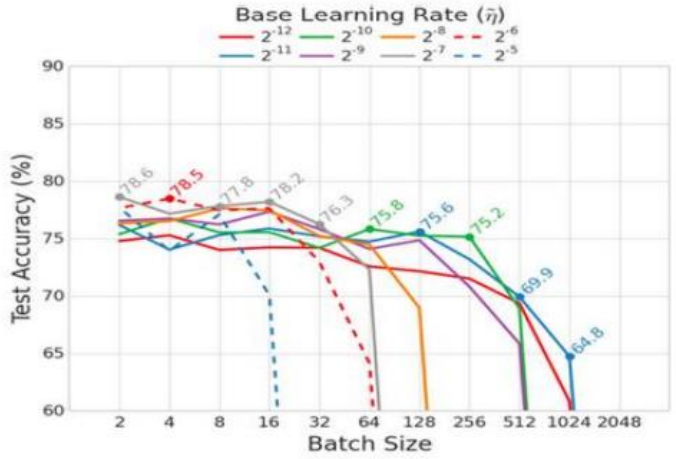
성능향상, 규제의 효과(Noise 효과, 스케일링 파라미터를 인위적으로 조절하면서), 가중치 초기화 설정을 신경 쓸 필요가 없어짐.

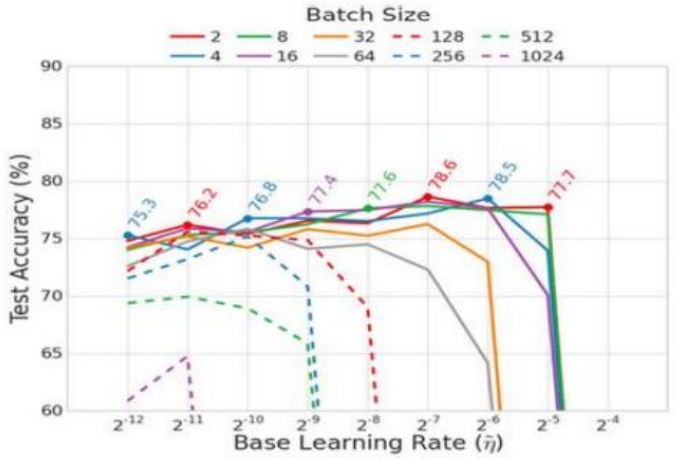
하지만, test는 보통 batch크기보다 작은 데이터이기에 BN을 할 수 가 없다.
→ 지수가중 이동(가장 최근 값에 영향을 많이 두개 끔 한 값)으로 구해 둔 값과 감마, 베타는 학습 시 최적화된 값으로 진행한다.

* Batch Normalization 이해와 적용

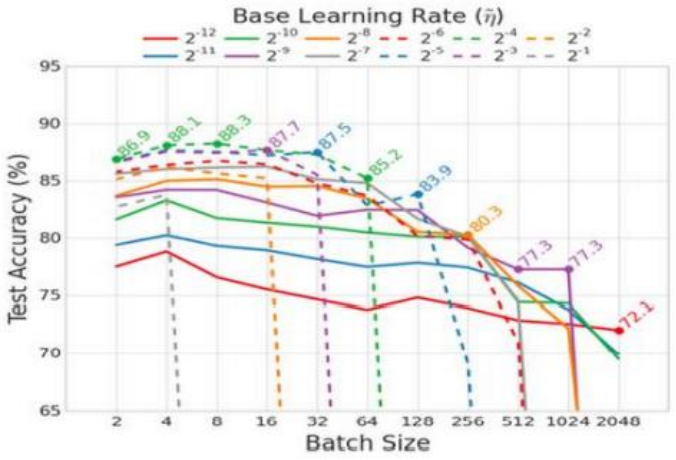
Batch Size 변화에 따른 모델 성능

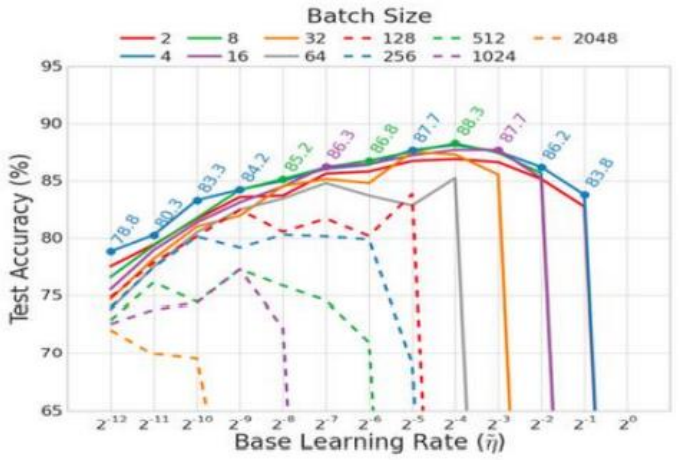
BN을 적용하지 않은 경우





BN을 적용한 경우





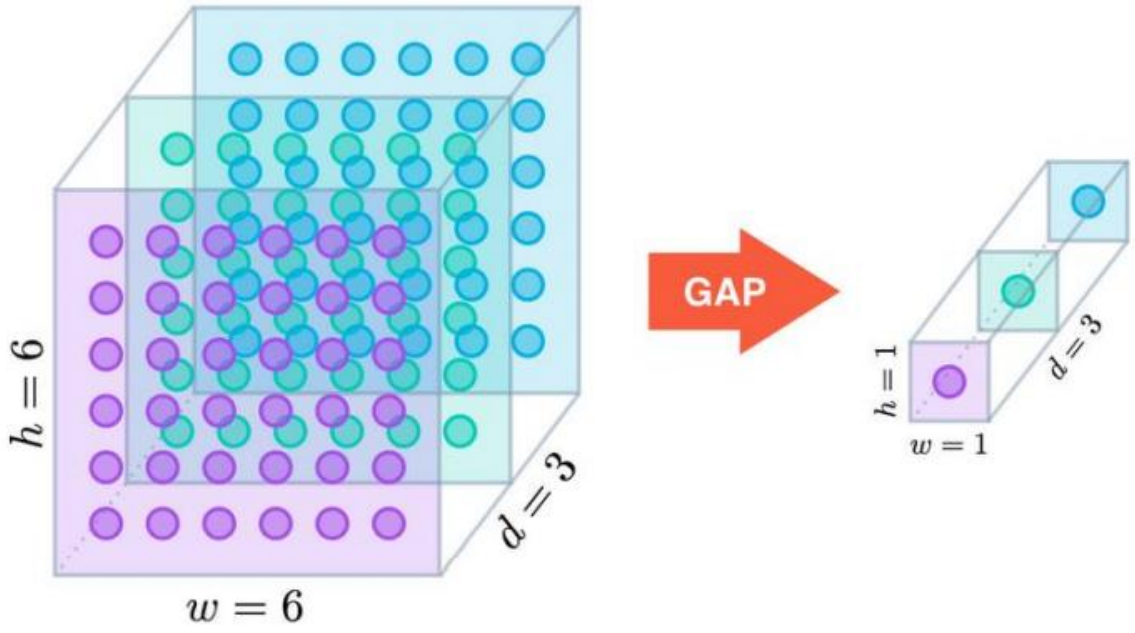
batch size가 증가할수록 test 정확도가 급격하게 하락.

BN을 적용하면, 그나마 조금 덜 떨어진다.

→ 학습 속도가 느리다고, Batch size를 키우면 성능이 떨어지는 현상이 일어날 것이다.

* Global Average Pooling

Conv 층 이후, Dense 층을 연결하는 과정에서 flatten 후 진행하면 파라미터가 굉장히 많이 증가한다. (kernel size * kernel size * channel 수) 그런데 GAP를 진행하면 channel 수만큼만 파라미터가 생긴다.(일반적으로 channel 수가 512개 이상일때, 사용한다.)
→ 성능이 무조건 더 좋고 할 수는 없지만 과적합을 막는 효과와 Epoch를 조금 더 진행하여 조금 더 성능에 나은 효과를 준다고는 해석해 볼 수 있다.



→채널별로 평균 값을 추출

* Weight Regularization

모델이 Loss를 줄이는 데만 집착하다 보면 학습 데이터에만 지나치게 정교화되게 되는 문제가 발생하기 쉽다. 따라서 Loss 값에 어떠한 값을 더하여 Loss 값을 무뎌 지게 만든다.

$$\text{손실 함수\_목표} = \text{Min}\left(\text{Loss}(W) + \text{alpha} * ||W||_2^2 \right)$$

→ l1, l2, l1_l2 등의 규제가 있는데 수식을 보면 Loss를 줄이면서 커진 가중치들의 값이 더해지면서 Loss를 지나치게 학습데이터에 맞추어 줄이면 가중치는 커지게 되어 총합 수식의 값을 커지게 만드는 효과를 가지도록 설계 되어있다.
일반적으로 Dense Layer에 적용한다.

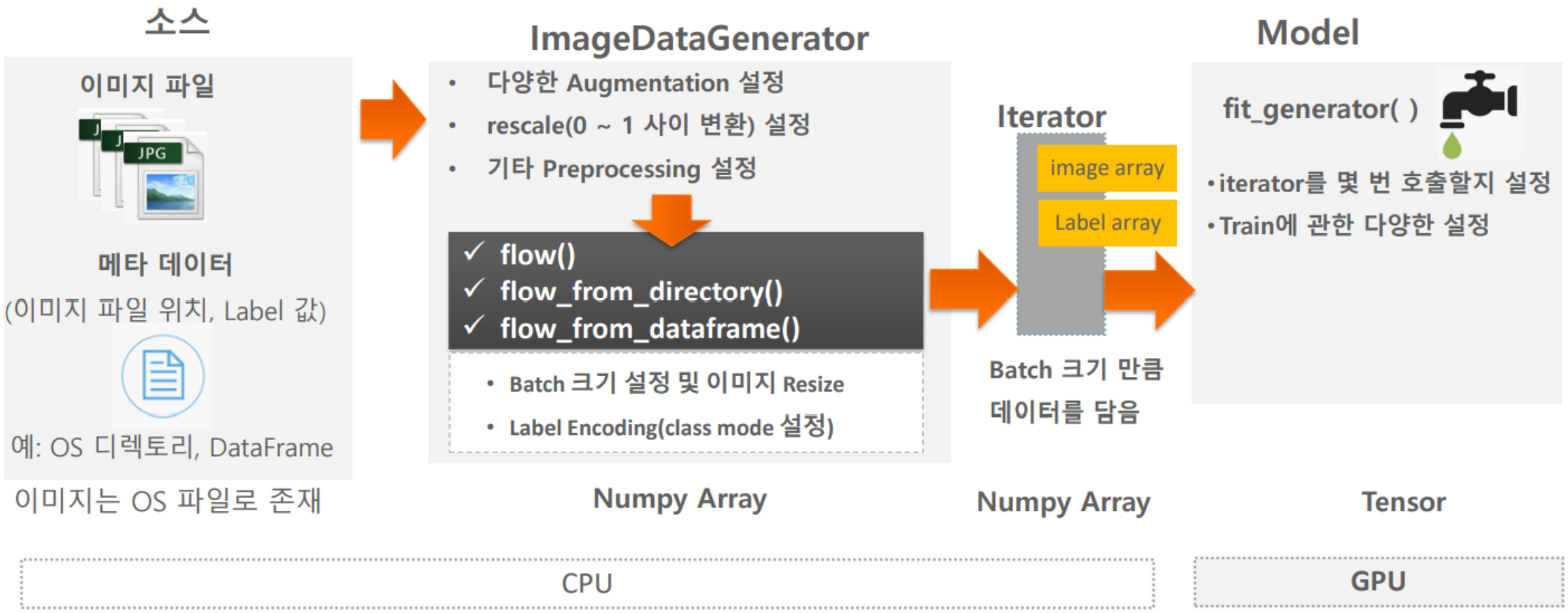
* Data Augmentation

머신 러닝의 기본은 데이터가 많아야 한다는 것이다. 그러나 각 분야에 필요한 정형화된 이미지를 늘리는 것은 쉽지가 않다. 따라서 개별 원본 이미지를 변형해서 학습을 수행하는 방법이 있다.
(rotation, scale, noise등)

공간(Spatial) 레벨 변형	Flip: Vertical(up/down), Horizontal(left/right) Crop: Center, Random Affine: Rotate, Translate, Shear, Scale(Zoom)
픽셀(Pixel) 레벨 변형	Bright, Saturation, Hue, GrayScale, ColorJitter Contrast Blur, Gaussian Blur, Median Blur Noise, Cutout Histogram Gamma RGBShift Sharpen

* Data Augmentation

Preprocessing과 Data Loading 메커니즘



Model에서 **fit() 또는 fit_generator()를 호출하기 전까지는** preprocessing과 Data Loading 실제 수행되지 않음.

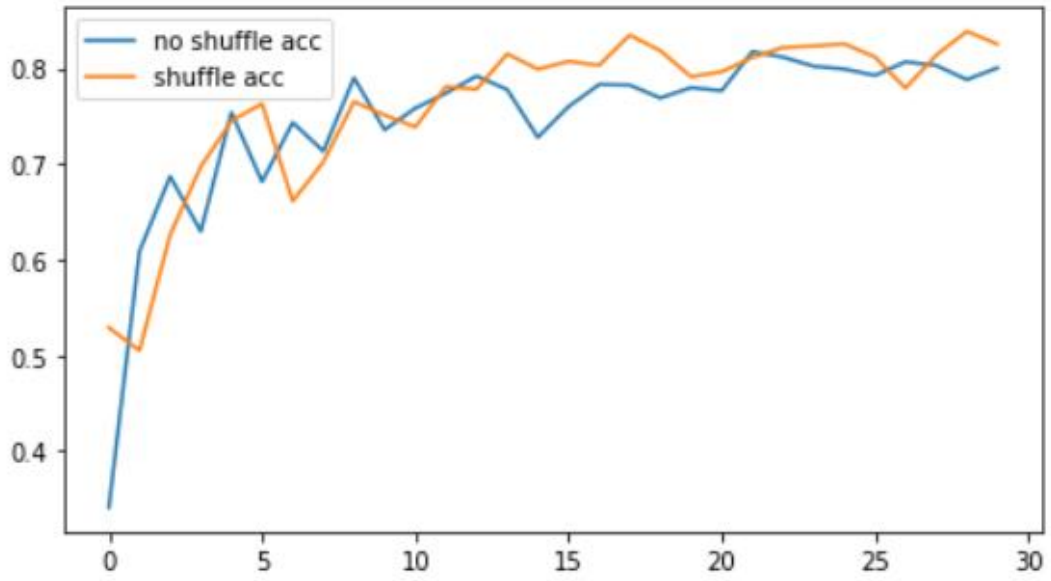
* Data Augmentation

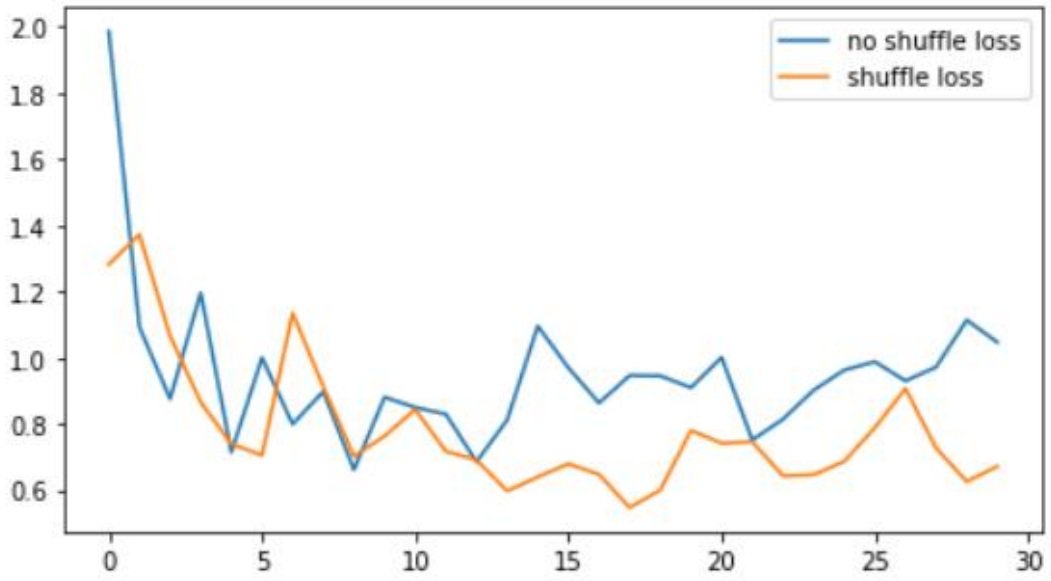
Preprocessing과 Data Loading 메커니즘

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_generator = ImageDataGenerator(
    #rotation_range=20,
    #zoom_range=(0.7, 0.9),
    horizontal_flip=True, # 좌우반전
    #vertical_flip=True,
    rescale=1/255.0
)
valid_generator = ImageDataGenerator(rescale=1/255.0)

flow_tr_gen = train_generator.flow(tr_images, tr_oh_labels, batch_size=BATCH_SIZE, shuffle=True)
# 원래 model.fit에 넣었다 (batch, shuffle 작업)
flow_val_gen = valid_generator.flow(val_images, val_oh_labels, batch_size=BATCH_SIZE, shuffle=False)
```

* Shuffle에 따른 성능 테스트

fit에 True가 디폴트 : 상대적으로 성능이 조금 좋아지는 효과가 있다.





감사합니다.