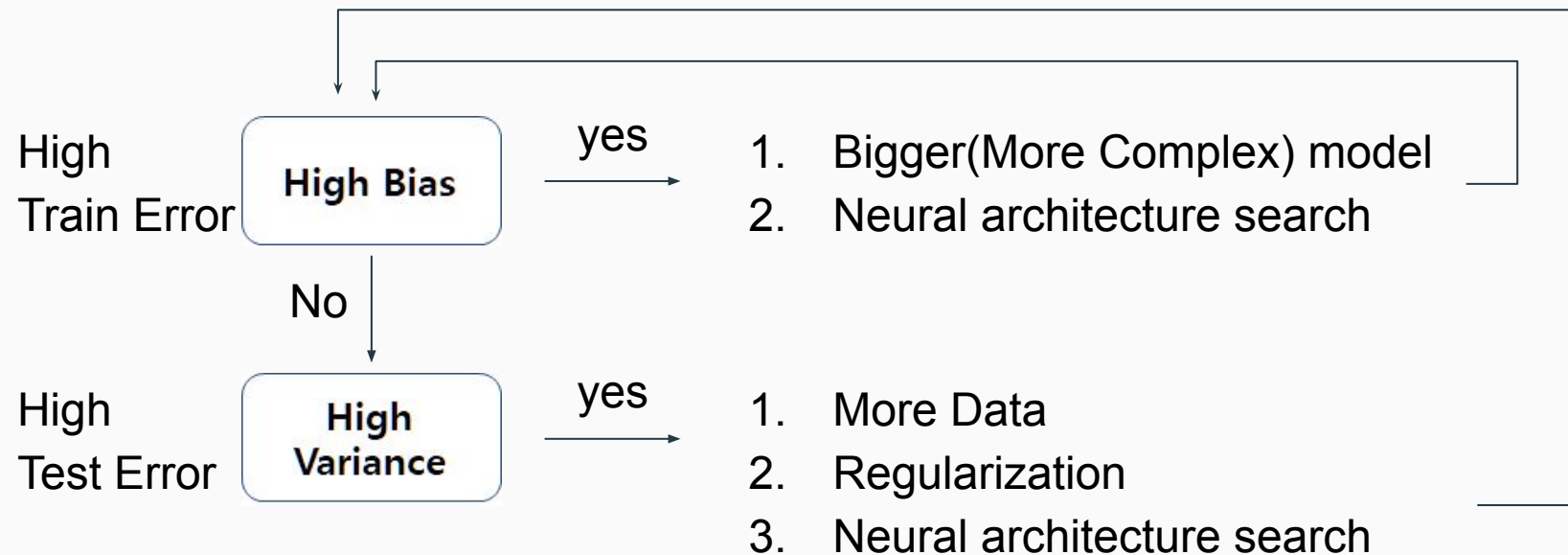


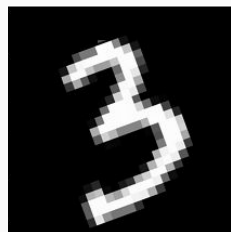
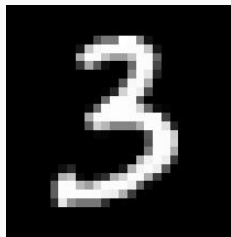
# Deep Learning with Tensorflow Keras

4강 합성곱 신경망

신유주

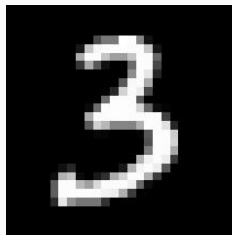


# MNIST 숫자 분류

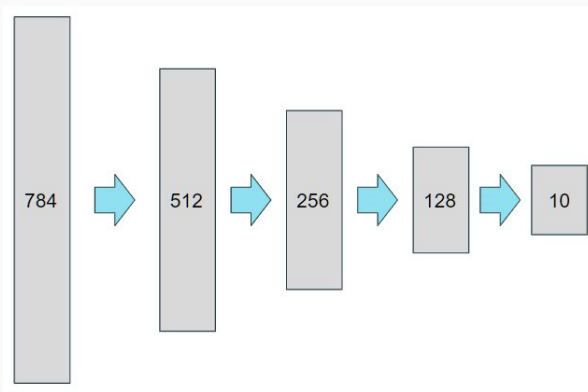
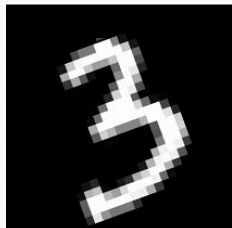


# MNIST 숫자 분류

Train  
Data



Test  
Data

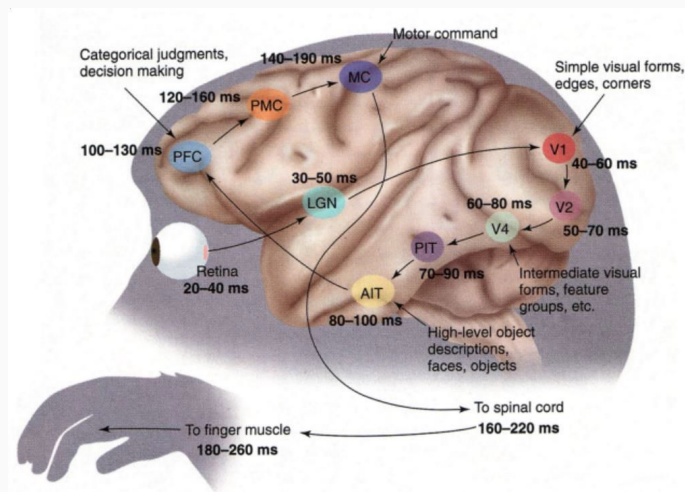
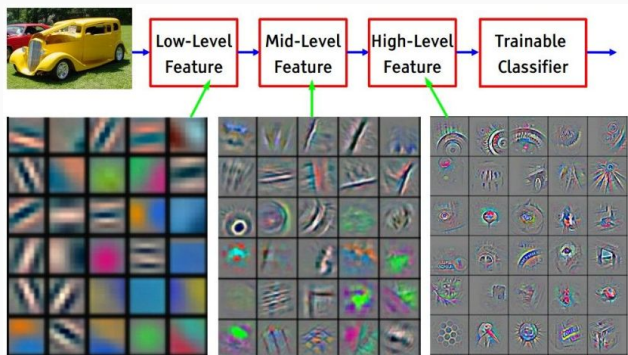


# 이미지 분류시 FCN의 한계

- 각 픽셀에 하나의 가중치 부여
- 임의의 두 픽셀간의 연관성을 모든 경우의 수에 대해 학습함
  - 파라미터 많아지고 깊은 DNN 구성 불가(overfitting)
- 이미지라는 데이터에 대한 선형적 지식이 거의 사용되지 않음
  - 한 픽셀과 그 근처의 다른 픽셀들은 색(값)이 같은 가능성이 높음
  - 한 픽셀의 값이 중요한 것이 아니라 어떤 시각적 패턴이 나타나는지가 중요
    - 사물의 형태와 색 변화가 이미지 분류에 더 중요함

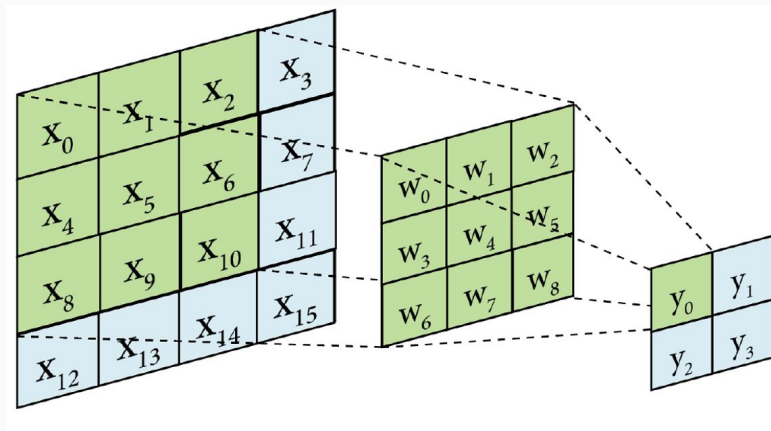
# 인간의 시각피질

- 단계별로 인식하는 패턴이 다름
- 처음엔 기본적인 모서리, 그다음엔 복잡한 소용돌이 무늬, 그다음엔 사물
  - Hierarchical Representation이라고 함



# 합성곱(Convolution)

- 이런 학습을 할 수 있는 **Convolution**이라는 연산을 **DNN**에 추가함
- 각 원소와 필터라는 가중치 형태를 원소별로 곱해 모두 더하는 형식
- 이미지의 형태에 대한 특징을 뽑아낼 수 있게 됨



# 실시간 합성곱 연산

1	0	1
0	1	0
1	0	1

Filter

1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved  
Feature



# 합성곱 연산 예시



원본 데이터  
(그냥 낙서)

10	0	0
0	10	0
0	0	10

픽셀로 표현한  
원본 데이터

10	0
0	10

우하향 대각선 필터

200	0
0	200

우하향 대각선이  
있다!

0	5
3	0

우상향 대각선 필터

0	30
50	0

우상향 대각선  
감지 안됨

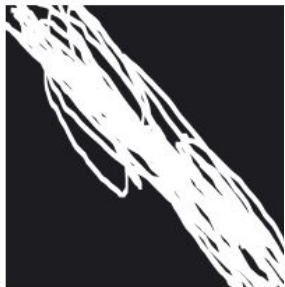
2	6
0	0

가로선 필터

20	0
60	20

가로선  
감지 안됨

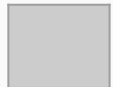
# 합성곱 연산 예시



원본 데이터  
(그냥 낙서)

10	0	0
0	10	0
0	0	10

픽셀로 표현한  
원본 데이터



: Weight

10	0
0	10

우하향 대각선 필터

200	0
0	200

우하향 대각선이  
있다!

0	5
3	0

우상향 대각선 필터

0	30
50	0

우상향 대각선  
감지 안됨

2	6
0	0

가로선 필터

20	0
60	20

가로선  
감지 안됨

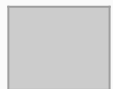
# 합성곱 연산 예시



원본 데이터  
(그냥 낙서)

10	0	0
0	10	0
0	0	10

픽셀로 표현한  
원본 데이터



: Weight



: Bias

10	0
0	10

우하향 대각선 필터

20

200	0
0	200

우하향 대각선이  
있다!

0	5
3	0

우상향 대각선 필터

10

0	30
50	0

우상향 대각선  
감지 안됨

2	6
0	0

가로선 필터

20

20	0
60	20

가로선  
감지 안됨

# 합성곱 연산 예시



원본 데이터  
(그냥 낙서)

10	0	0
0	10	0
0	0	10

픽셀로 표현한  
원본 데이터



: Weight



: Bias

10	0
0	10

우하향 대각선 필터

20

220	20
20	220

우하향 대각선이  
있다!

0	5
3	0

우상향 대각선 필터

10

0	30
50	0

우상향 대각선  
감지 안됨

2	6
0	0

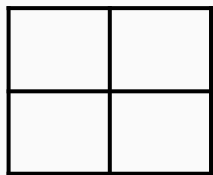
가로선 필터

20

20	0
60	20

가로선  
감지 안됨

# 합성곱 연산 예시



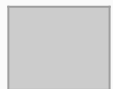
: Feature(Activation)  
Map  
= Input for next layer



원본 데이터  
(그냥 낙서)

10	0	0
0	10	0
0	0	10

픽셀로 표현한  
원본 데이터



: Weight



: Bias

10	0
0	10

우하향 대각선 필터

20

220	20
20	220

우하향 대각선이  
있다!

0	5
3	0

우상향 대각선 필터

10

0	30
50	0

우상향 대각선  
감지 안됨

2	6
0	0

가로선 필터

20

20	0
60	20

가로선  
감지 안됨

# 합성곱 효과

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

근처 픽셀과 색을 섞게  
되어 출력 합성곱  
결과가 흐려지게 됨

Here's a result that I got:



-1	-1	-1
-1	8	-1
-1	-1	-1

근처 픽셀과 비슷한 색이면  
0이 되고 다른 색이면  
차이가 커지게 되어  
모서리를 감지하게 됨

Below result I got with edge detection:



# 패딩(Padding, Zero-padding)

**Convolution(합성곱)** 전 데이터 테두리에 **0**픽셀을 추가해 출력 크기 조절



원본 데이터



10	0	0
0	10	0
0	0	10

픽셀로 표현한  
원본 데이터



0	5
3	0

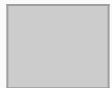
우상향 대각선 필터



10

10	40
60	10

우상향 대각선이  
있...나...?



: Weight



: Bias

# 패딩(Padding, Zero-padding)

**Convolution(합성곱)** 전 데이터 테두리에 픽셀을 추가해 출력 크기 조절



원본 데이터

0	0	0	0	0
0	10	0	0	0
0	0	10	0	0
0	0	0	10	0
0	0	0	0	0

Zero Padding을 붙이고  
픽셀로 표현한  
원본 데이터

0	1
1	0

우상향 대각선 필터

10

10	20	10	10
20	10	20	10
10	20	10	20
10	10	20	10

우상향 대각선 필터로  
우하향 대각선  
모양이 나옴



: Padding

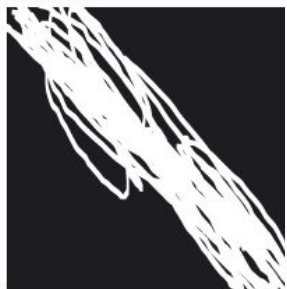


# 패딩을 하는 이유

- 입력값과 출력값의 차원을 맞춰주기 위해 시행함
- 의도하지않은 데이터 차원 감소를 막아줌
  - 이미지 끝 부분에 들어 있는 정보를 제대로 전달할 수 있음

# 스트라이딩(Striding)

필터적용 위치 간격(이전까진 **1**이었음, **3**로 시도)



원본 데이터

0	0	0	0	0
0	10	0	0	0
0	0	10	0	0
0	0	0	10	0
0	0	0	0	0

0	1
1	0

우상향 대각선 필터

0	



: Padding

Zero Padding을 붙이고  
픽셀로 표현한  
원본 데이터

# 스트라이딩(Striding)

필터적용 위치 간격(x\_Stride = 3)



원본 데이터


0	0	0	0	0
0	10	0	0	0
0	0	10	0	0
0	0	0	10	0
0	0	0	0	0

Zero Padding을 붙이고  
픽셀로 표현한  
원본 데이터

0	1
1	0

우상향 대각선 필터

0	0

 : Padding

# 스트라이딩(Striding)

필터적용 위치 간격(y\_Stride = 3)



원본 데이터

	0	0	0	0
0	10	0	0	0
0	0	10	0	0
0	0	0	10	0
0	0	0	0	0

0	1
1	0

우상향 대각선 필터

0	0
0	



: Padding

Zero Padding을 붙이고  
픽셀로 표현한  
원본 데이터

# 스트라이딩(Striding)

필터적용 위치 간격(x\_Stride = 3)



원본 데이터


0	0	0	0	0
0	10	0	0	0
0	0	10	0	0
0	0	0	10	0
0	0	0	0	0

Zero Padding을 붙이고  
픽셀로 표현한  
원본 데이터

0	1
1	0

우상향 대각선 필터

0	0
0	0

 : Padding

# 합성곱 후 이미지 크기

**L** = 입력 이미지의 한 면 길이

**F** = 필터의 한 면 길이

**O** = 출력 크기

**P** = Padding

**S** = Stride

$$O = \frac{L + 2P - F}{S} + 1$$

# Pooling

채널을 유지하되 **Weight**를 요구하지 않으며, 데이터 크기를 줄여줌.



원본 데이터

0	0	0	0	0
0	10	0	0	0
0	0	10	0	0
0	0	0	10	0
0	0	0	0	0

Zero Padding을 붙이고  
픽셀로 표현한  
원본 데이터

MaxPooling  
2 X 2  
stride=3

AveragePooling  
2 X 2

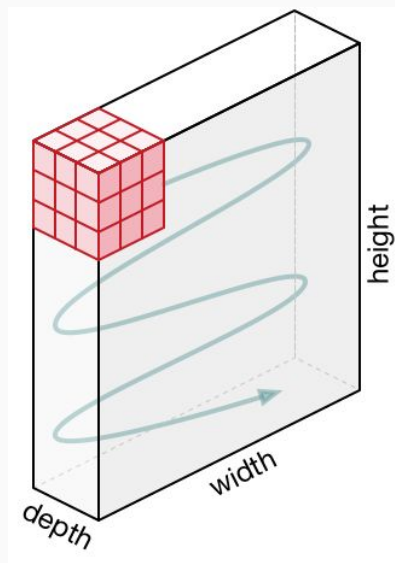
...

10	0
0	10



: Padding

# 필터(=커널)의 채널 수



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

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

Kernel Channel #1

308

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

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

Kernel Channel #2

-498

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

164

+ 1 = -25

Bias = 1

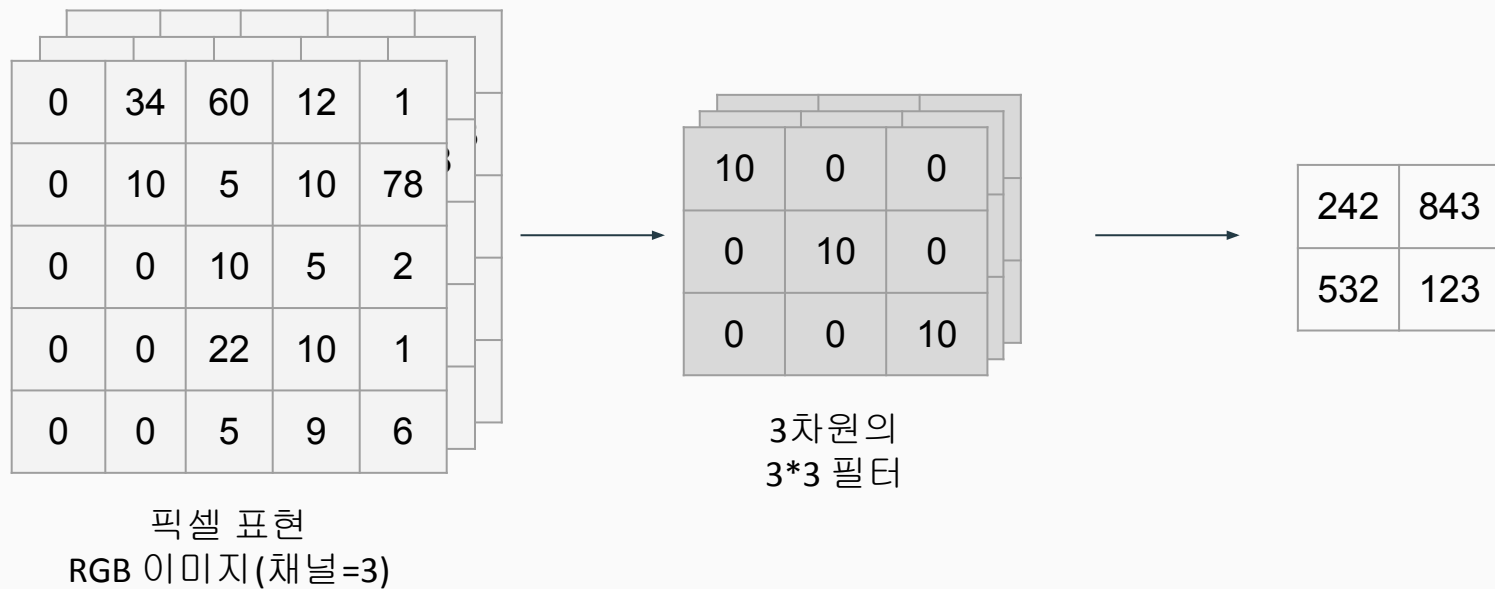
Output

-25			...
			...
			...
			...
			...
...	...	...	...



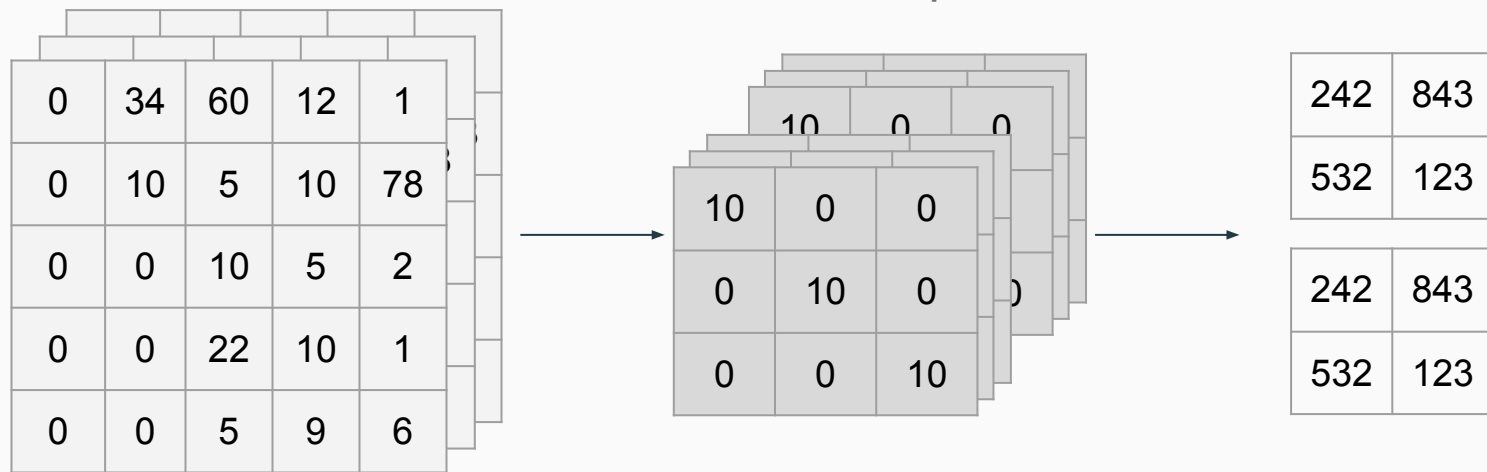
# 필터 수

필터는 이전 이미지의 차원(채널)의 갯수만큼 존재



# 필터 수와 출력 이미지의 채널 수

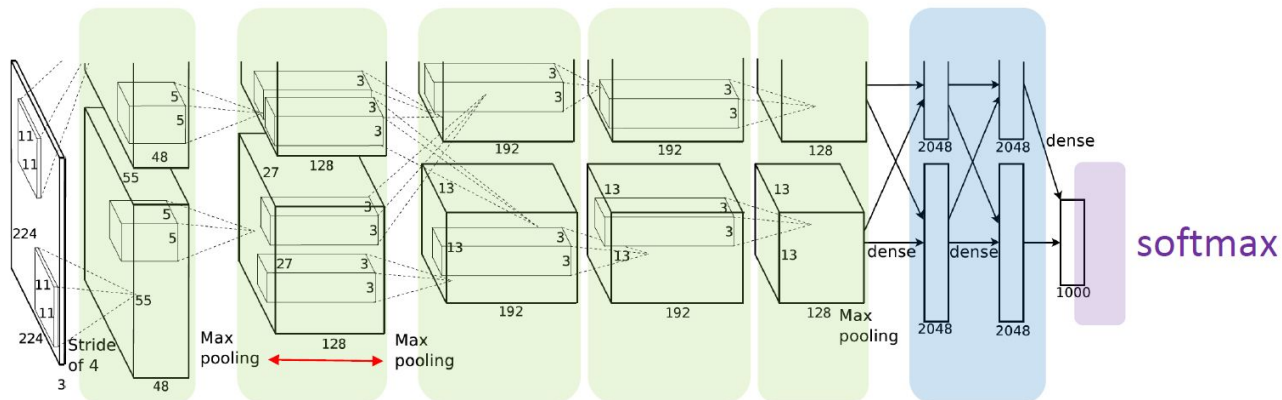
- 이전 CNN 층 필터의 수 = 출력 feature map의 채널 수가 됨



픽셀 표현  
RGB 이미지(채널=3)

3차원의  
3\*3 필터 2개

# 일반적인 CNN 구조



depth = # of Filters

Fully connected layers

깊어질수록 필터의 가로세로 크기는 작아지고 채널(depth) 수는 많아지는 경향  
대부분의 CNN이 이런 경향을 보임

# Keras 구현

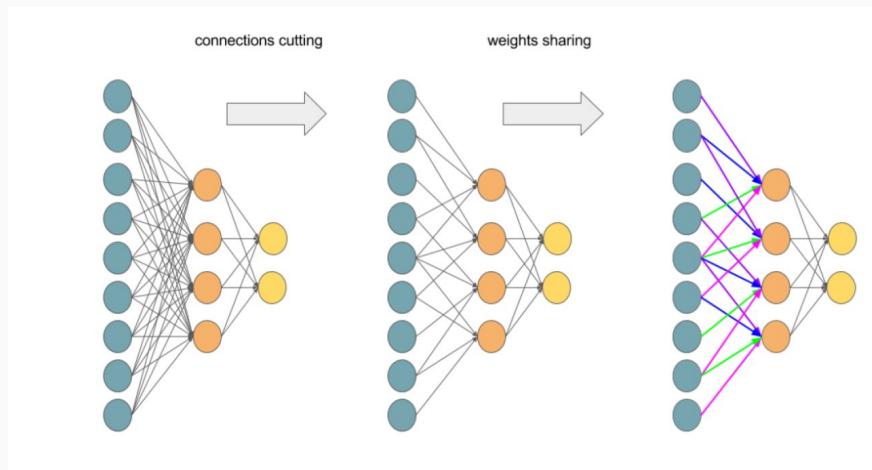
```
from functools import partial

DefaultConv2D = partial(keras.layers.Conv2D,
                        kernel_size=3, activation='relu', padding="SAME")

model = keras.models.Sequential([
    DefaultConv2D(filters=64, kernel_size=7, input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=128),
    DefaultConv2D(filters=128),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=256),
    DefaultConv2D(filters=256),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(units=128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=10, activation='softmax'),
])
```

# 사실은 CNN도 FCN?

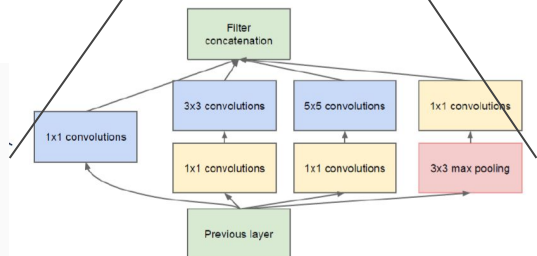
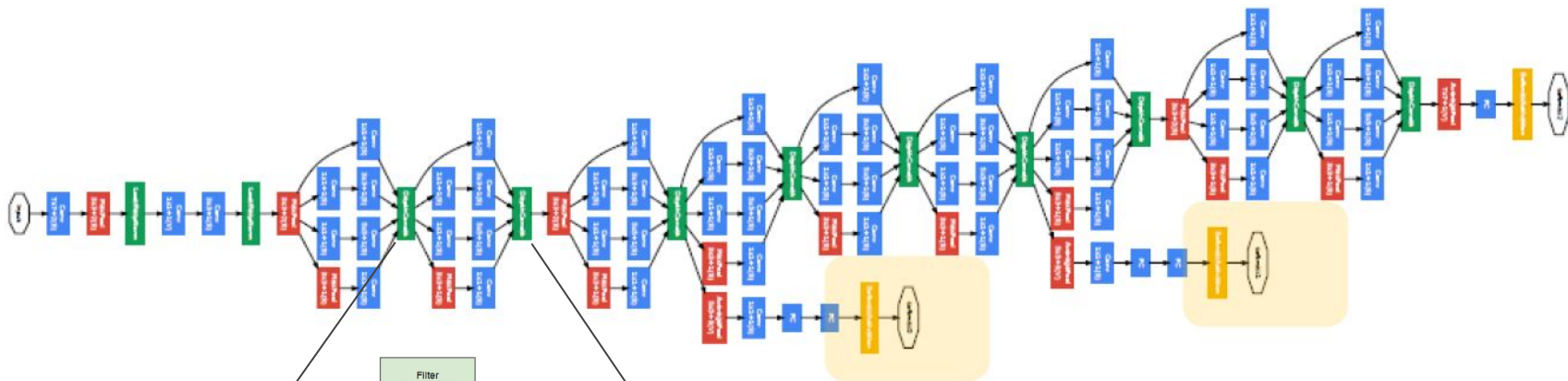
- CNN은 사실 일부 입력 값에 대해 가중치를 공유하도록 만든 FCN
- 그 가중치가 바로 CNN 필터의 가중치와 같음
- 맨 오른쪽 그림의 화살표 색이 같으면 같은 가중치임



# CNN의 발전

- 더 깊은 층을 쌓아 Hierarchical representation learning 최대화
- 그러나 경사 소실 문제 발생
  - Auxiliary output
  - Residual error

# GoogLeNet



(b) Inception module with dimension reductions

- 모델이 훈련할 채널을 고를 수 있도록 설계함
- 벡터를 이어붙여 차원이 늘어나면 **pooling**
  - 데이터 차원 감소시킴
- 1x1 conv: 채널수 감소용
- Auxiliary output에서 Backprop 수행

# 차원이 다른 ResNet의 층 수



AlexNet (8 layers)



VGGNet (19 layers)



GoogLeNet (22 layers)

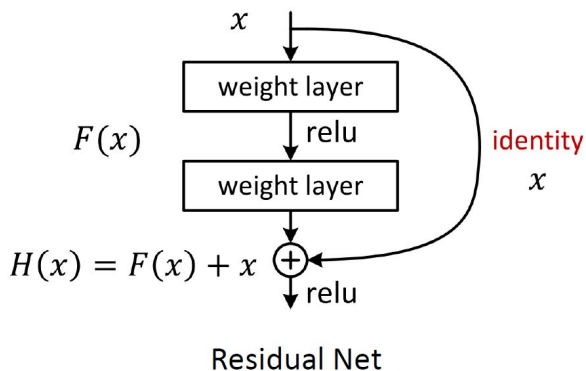
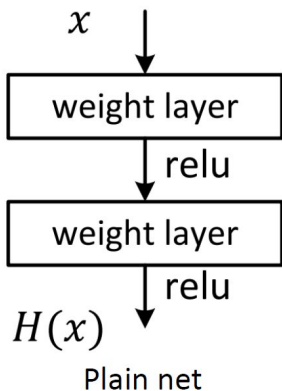


ResNet (152 layers)

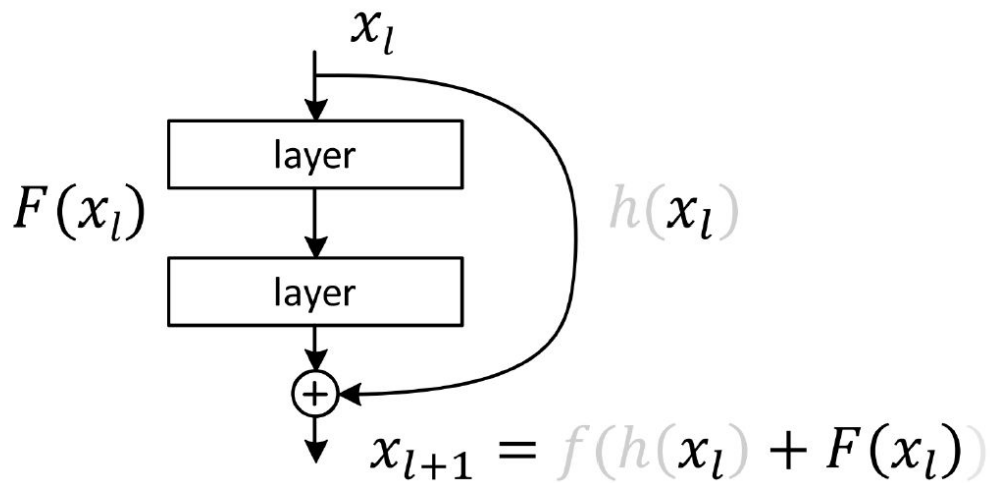


# ResNet

- 미분값을 우회해서 전달하고, 의미있는 층만 학습하도록 함
- 최악의 경우 **convolution**을 하는 모든 필터의 가중치를 0으로 만들어 **input**으로 받은 데이터를 그냥 넘길 수 있도록 함



# ResNet 작동원리



$$x_{l+1} = x_l + F(x_l)$$



$$x_{l+2} = x_{l+1} + F(x_{l+1})$$

$$x_{l+2} = x_l + F(x_l) + F(x_{l+1})$$

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

# ResNet 작동원리

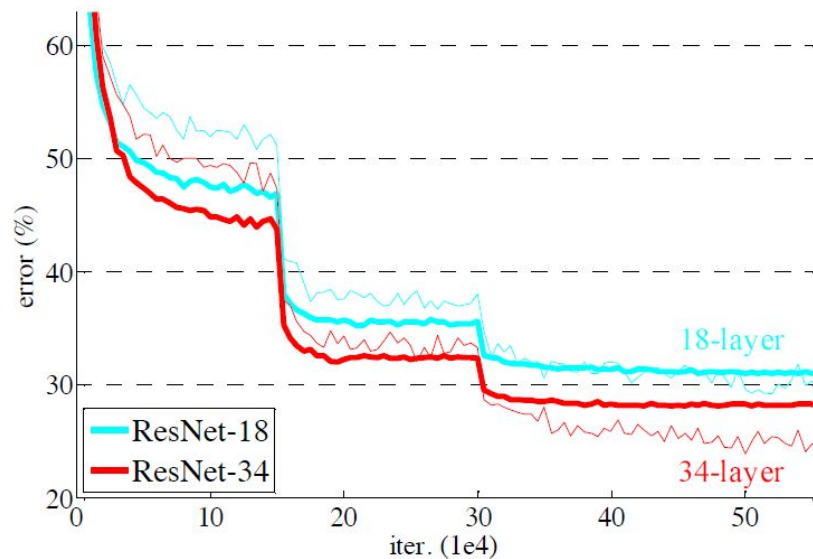
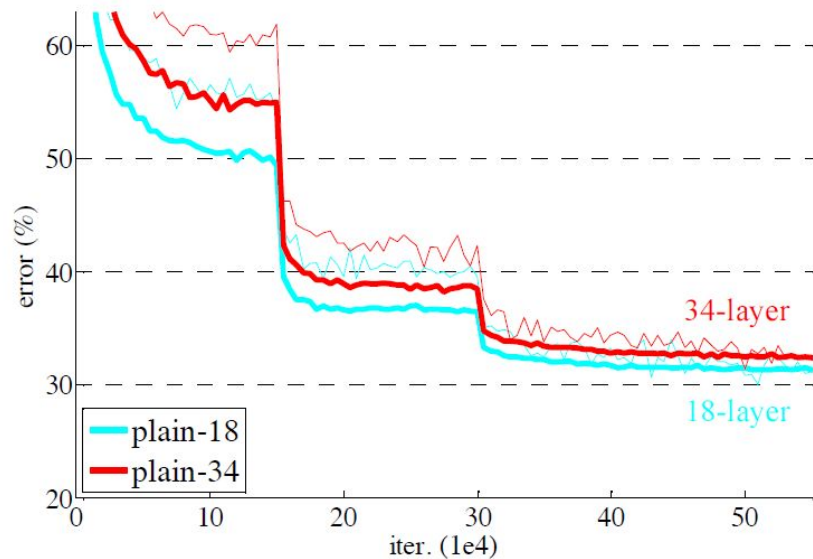
역전파시 미분해도 남아있는  
상수값(1) 때문에 항상 더하는  
미분 값이 생겨 여러 번의  
미분값 곱으로 생기는 경사  
소실이 방지됨

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$



$$\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L} \frac{\partial x_L}{\partial x_l} = \frac{\partial E}{\partial x_L} \left( 1 + \frac{\partial}{\partial x_l} \sum_i^L F(x_i) \right)$$

# ResNet 성능



# ResNet 구현

```
DefaultConv2D = partial(keras.layers.Conv2D, kernel_size=3, strides=1,
                        padding="SAME", use_bias=False)
```

```
class ResidualUnit(keras.layers.Layer):
```

```
    def __init__(self, filters, strides=1, activation="relu", **kwargs):
```

```
        super().__init__(**kwargs)
```

```
        self.activation = keras.activations.get(activation)
```

```
        self.main_layers = [
```

```
            DefaultConv2D(filters, strides=strides),
```

```
            keras.layers.BatchNormalization(),
```

```
            self.activation,
```

```
            DefaultConv2D(filters),
```

```
            keras.layers.BatchNormalization())
```

```
        self.skip_layers = []
```

```
        if strides > 1:
```

```
            self.skip_layers = [
```

```
                DefaultConv2D(filters, kernel_size=1, strides=strides),
```

```
                keras.layers.BatchNormalization())
```

```
    def call(self, inputs):
```

```
        Z = inputs
```

```
        for layer in self.main_layers:
```

```
            Z = layer(Z)
```

```
        skip_Z = inputs
```

```
        for layer in self.skip_layers:
```

```
            skip_Z = layer(skip_Z)
```

```
        return self.activation(Z + skip_Z)
```

```
model = keras.models.Sequential()
```

```
model.add(DefaultConv2D(64, kernel_size=7, strides=2,
```

```
                        input_shape=[224, 224, 3]))
```

```
model.add(keras.layers.BatchNormalization())
```

```
model.add(keras.layers.Activation("relu"))
```

```
model.add(keras.layers.MaxPool2D(pool_size=3, strides=2, padding="SAME"))
```

```
prev_filters = 64
```

```
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
```

```
    strides = 1 if filters == prev_filters else 2
```

```
    model.add(ResidualUnit(filters, strides=strides))
```

```
    prev_filters = filters
```

```
model.add(keras.layers.GlobalAvgPool2D())
```

```
model.add(keras.layers.Flatten())
```

```
model.add(keras.layers.Dense(10, activation="softmax"))
```

# ResNet 가져다 쓰기

```
model = keras.applications.resnet50.ResNet50(weights="imagenet")
```