CNN(Convolution Neural Network)

1조 다음은 2조

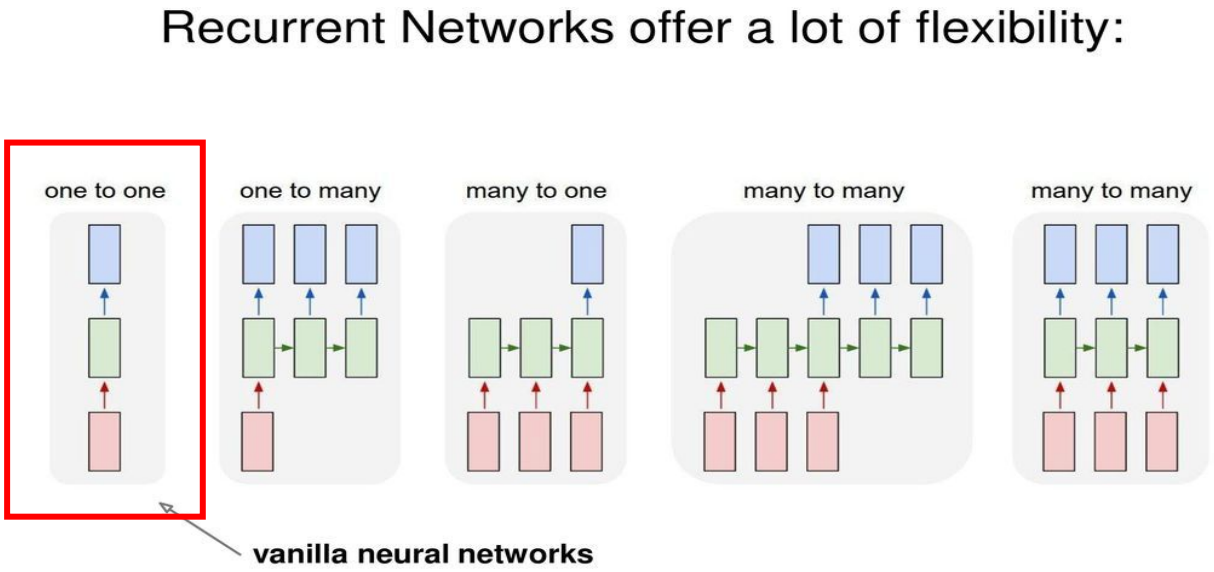
목차

- Vanilla Neural Network의 약점과 이를 보완하려는 시도
- Convolutional Neural Network 에서의 Filter 의 의미
- Convolutional Neural Network의 의미, 아이디어 및 구조
- Filter 및 image에서의 Convolution에 대한 간단한 소개
- Convolutional Neural Network에서의 역전파
- 여러가지 Convolutional Neural Network 의 종류

Vanilla?

먹는건가?

Vanilla Neural Network (RNN)

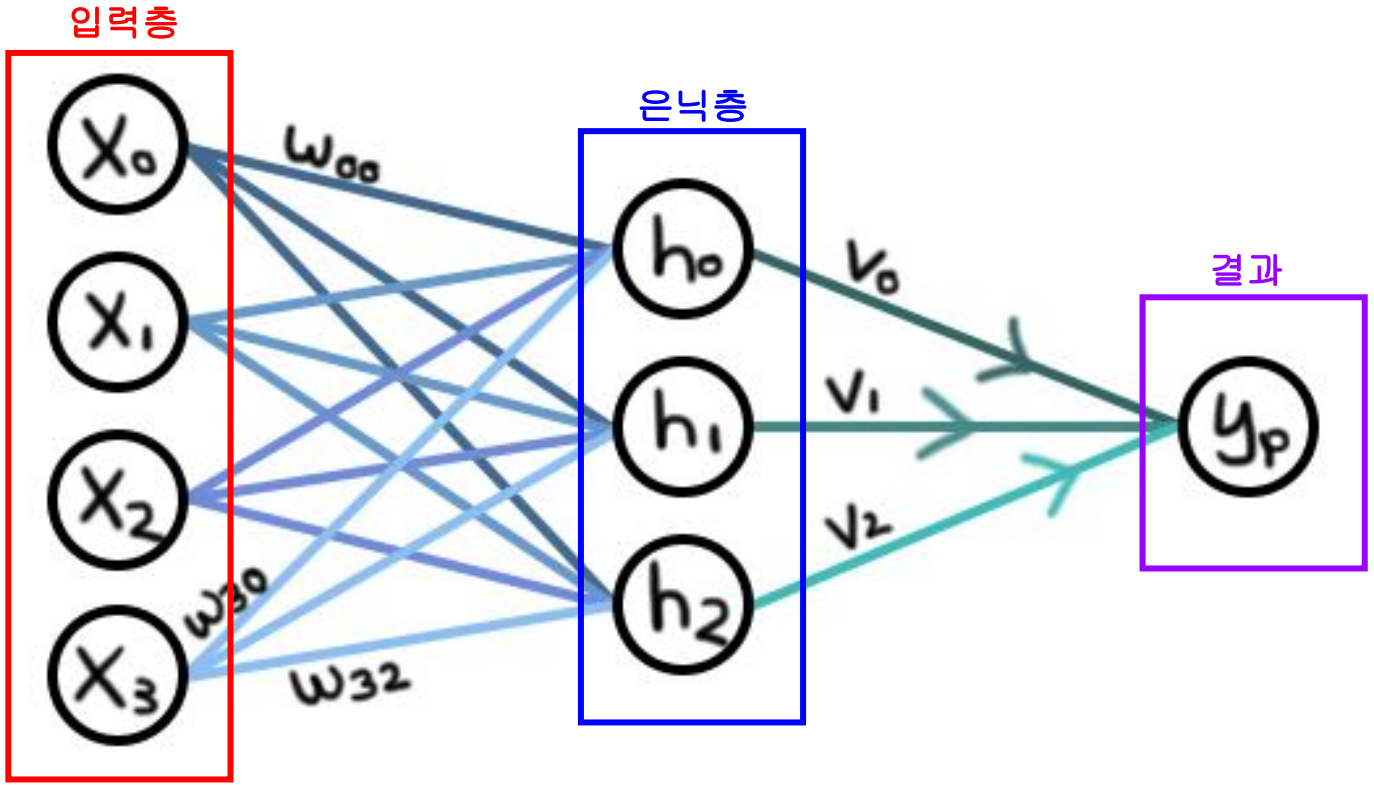


- **Vanilla Neural Network**

  - 하나의 은닉층을 가지고 있는 MLP

Andrej Karpathy

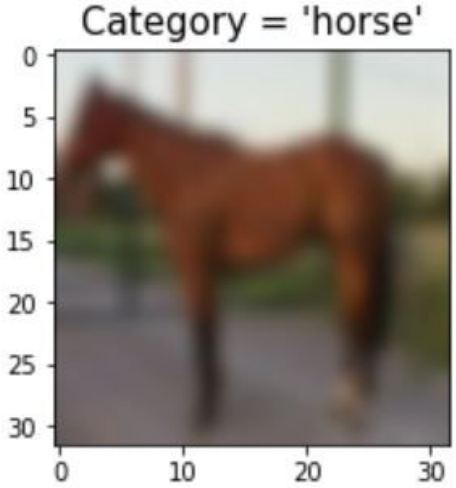
Vanilla Neural Network - Fully Connected



완전 연결층의 한계

이미지 데이터를 다루는 데 부적합

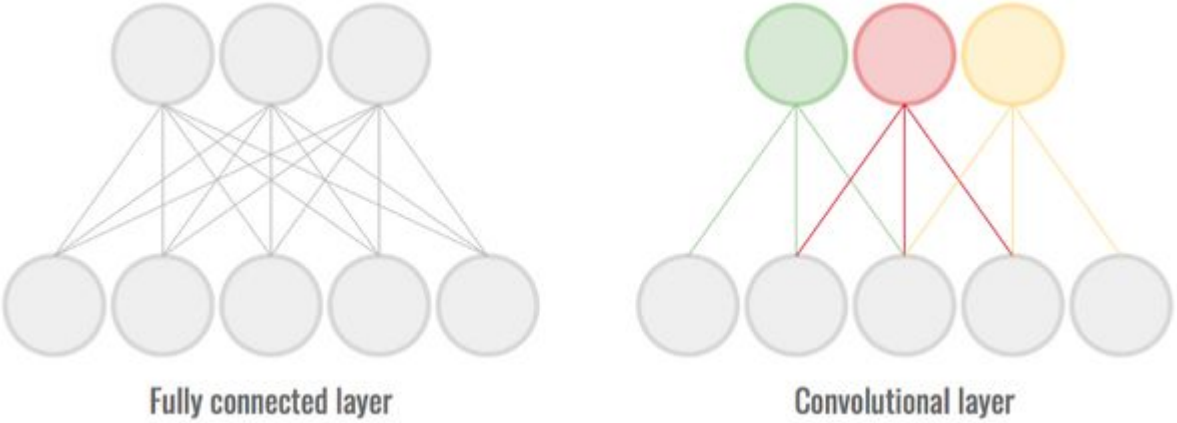
- 이유 1. 오버피팅 발생 가능
- 이유 2. 값의 중복
- 이유 3. 입력 형태가 1차원으로 한정



32 × 32 × 3 = 3072

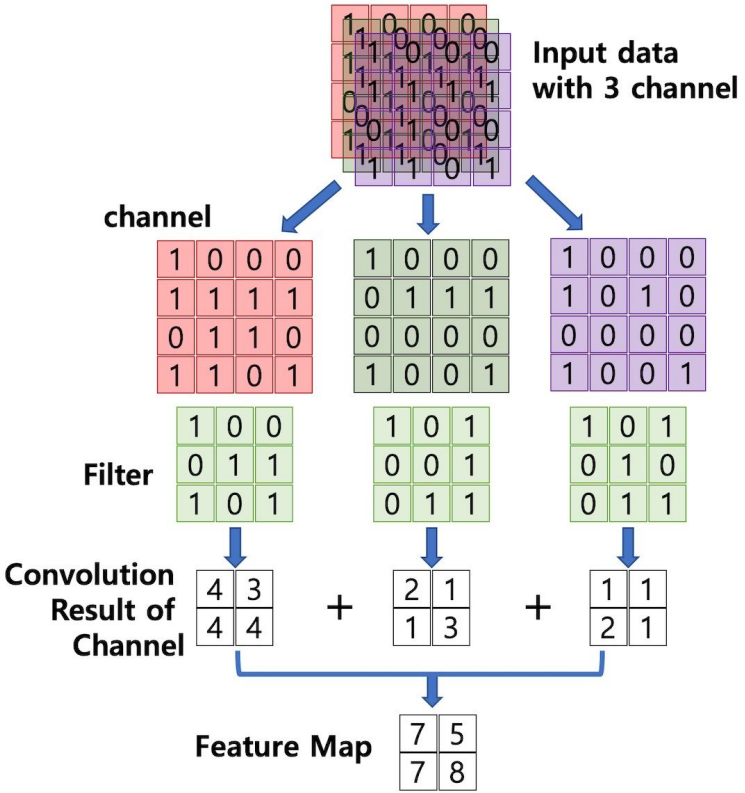
200 × 200 × 3 = 120000
???/?????????????

CNN과 FC의 차이



- **합성곱 층** - 입력 데이터 형상 유지
- **Max pooling** - 정보를 유지하면서 변수 수 줄임

Convolution - Filter, Image



- 필터 수 = 채널 수
- 필터는 각 채널을 순회하며 합성곱을 계산
- 각 채널의 피처 맵을 합산하여 최종 피처 맵으로 반환

Convolutional Neural Network(CNN)

의미, 아이디어 및 구조

CNN(Convolutional Neural Network)란 무엇인가 ?

- CNN : “Convolution” 이라는 작업이 들어간 NN(Neural Network)

  우리가 알고 싶은 것
  Convolution = “합성곱” 이라곤 하는데,,,,그래서 이게 뭘 하는 건데 ?

그전에 알고싶다 (1)

- 그전에, 초창기 CNN 개발 시, 아이디어

  고양이가 보는 것마다 자극 받는 뇌의 위치가 다른 것을 보고, 아이디어를 얻었다고 한다.
  = 즉, 한 Image 전체를 보는 것이 아니라, 부분을 보는 것이 핵심 !
  여기서, “부분” 이, 곧 우리가 알고있는, “filter” 라고 한다.

그전에 알고싶다 (2)

- 앞선, CNN의 추상적인, 정의말고, 하나 더

  CNN은 DNN(Deep Neural Network)에서 이미지나 영상과 같은 데이터를 처리할 때 발생하는 문제점들을 보완한 방법

그전에 알고싶다 (2) (Cont.)

- DNN이 무슨 문제가 있는데 ?

DNN은 (기본적으로) 1차원 데이터를 사용. 가장 대표적인 예는 iris데이터 (붓꽃 종 분류)

이런 데이터가 많이 있고, 여기서 꽃받침과 꽃잎의 정보를 Input데이터(=X)라 하고, 꽃의 종류를 output데이터(=Y)로 설정하고 학습하는 방식

Index	꽃받침길이	꽃받침너비	꽃잎길이	꽃잎너비	꽃의종류(Target)
1	5.1	3.5	1.4	0.2	setosa

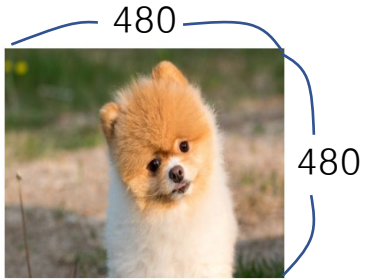
- 위에서 설명한 토대로 생각했을 때, DNN이 1차원 데이터를 사용한다는 것은

  즉, 데이터가 하나의 row(열)로 표현 되기 때문

그전에 알고싶다 (2) (Cont.)

- 근데, 중요한 것은, 이미지(특히 컬러이미지)는 보통 어떤가 ?

  하나의 이미지는, 예를 들어 다음과 같은 강아지 사진이 있다치자.



이와 같은, 이미지 데이터는 하나의 row로
표현하고
DNN으로 표현 할수 있을까 ?
=> 할 순 있겠지만, 데이터의 큰 손실을 갖게 될 것

만약, 1차원 데이터로 실제로 표현할려면, 위의 그림과 같이, 왼쪽위에서부터, 1px 값씩 잘라서 나열하는 형태 일 것임 (실제로는 더 작은 숫자들의 집합으로 이뤄진 한 줄 데이터)

이렇게 되면, 결국 이미지와 같은 데이터를 DNN으로 학습시킬 경우 문제를 알 수 있다.
= 이미지데이터는 한 점(특정 픽셀)들이 모여 하나의 객체(형상 ?)을 만들 게 되는데
한 줄로 된 row데이터에서는, 이런 연관 관계가 제거 된다는 문제가 있다 !

CNN 은..

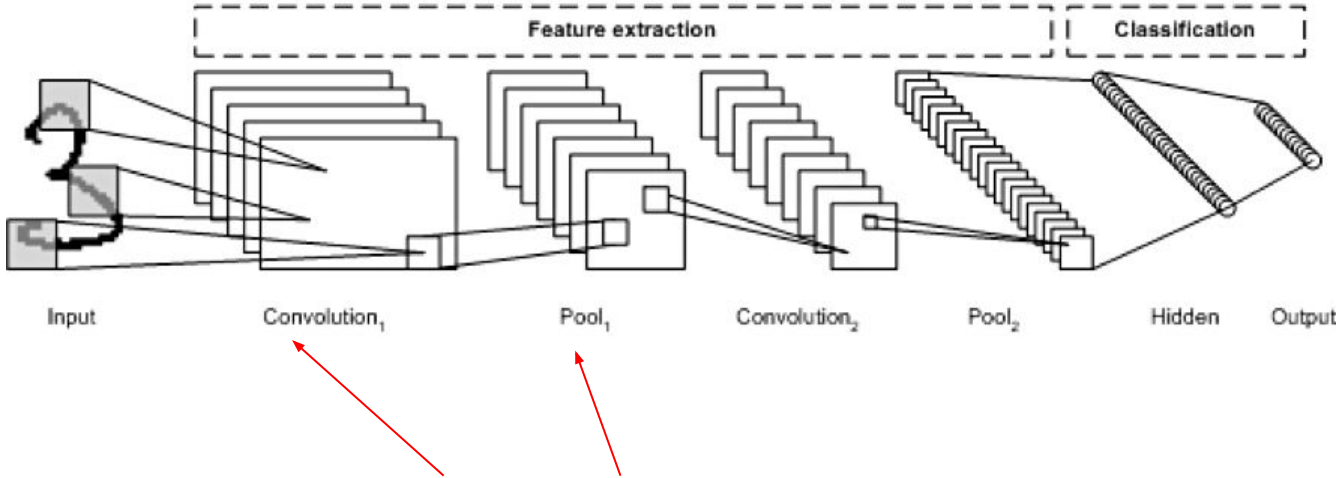
이전, 강아지 사진을 예로, 앞으로 설명할 자세한 중간과정은 생략하고, 대략적으론

맨 오른쪽, 처럼 특정 크기만큼의 영역을 나누고, 우리는 앞으로 설명할 일련의 과정을 할 것이다.

(일단 DNN과의 차이는 확실한건, 데이터를 1차원으로 표현을 해야할 필요가 없고, 그만큼 데이터의 손실이 없다는 점 !!)

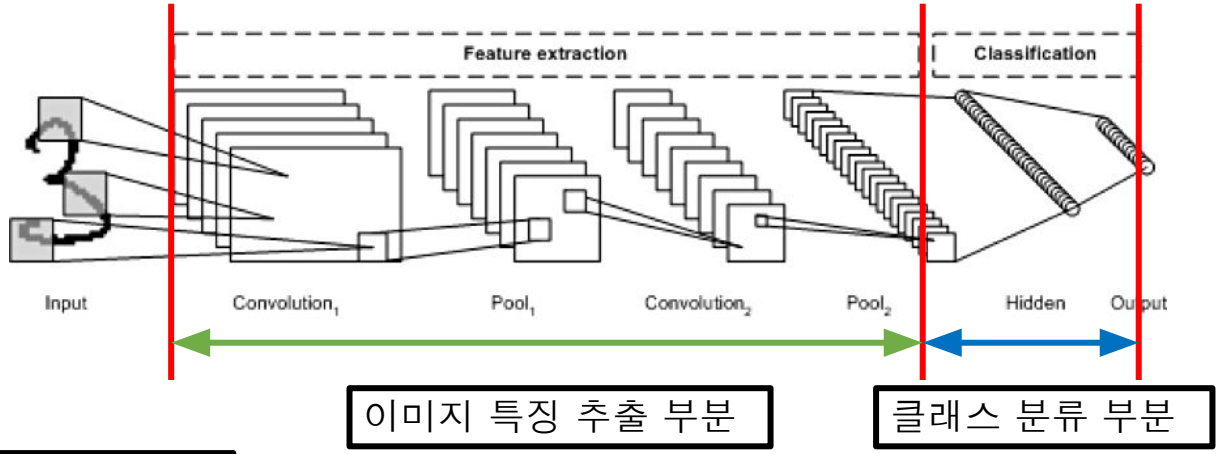
Convolutional Neural Network(CNN)

- CNN(또는 ConvNet)은 모델이 직접 이미지, 비디오, 텍스트 또는 사운드를 분류하는 머신 러닝의 한 유형인 딥러닝에 가장 많이 사용되는 알고리즘
  - 데이터에서 직접 학습하며,패턴을 사용하여 이미지를 분류하고 특징을 수동으로 추출할 필요 X
  - 자율주행자동차, 얼굴인식과 같은 객체인식, Computer Vision 분야 사용



이 과정들은, 각각 뭘하는 걸까 ?

Convolutional Neural Network(CNN)



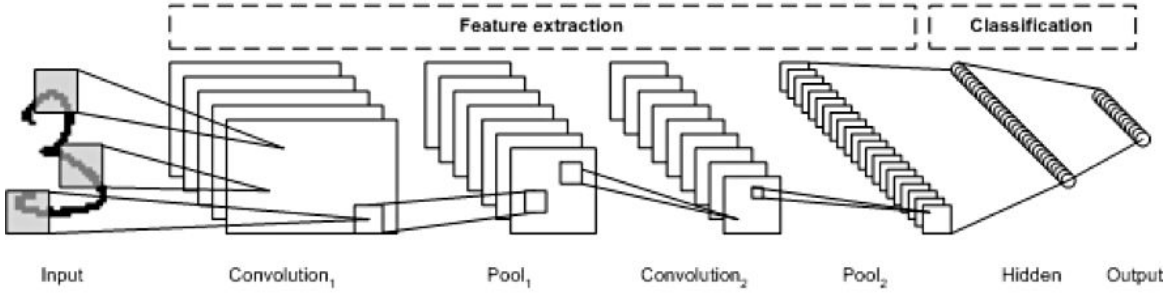
이미지 특징 추출 부분

- Convolution Layer 와 Pooling Layer를 여러 겹 쌓는 형태로 구성
  - Convolution Layer는 입력 데이터에 필터(filter)를 적용 후 활성화 함수를 반영하는 필수적 레이어
  - 반면, Pooling Layer는 선택적인 레이어

클래스 분류 부분

- CNN의 마지막 부분이다. Fully Connected Layer(완전 연결 층)가 추가된다.
  - 이미지 특징 추출 부분과, 클래스 분류 부분의 경계에 Flatten Layer 가 위치 (데이터 1D로 변환)

Convolutional Neural Network(CNN)

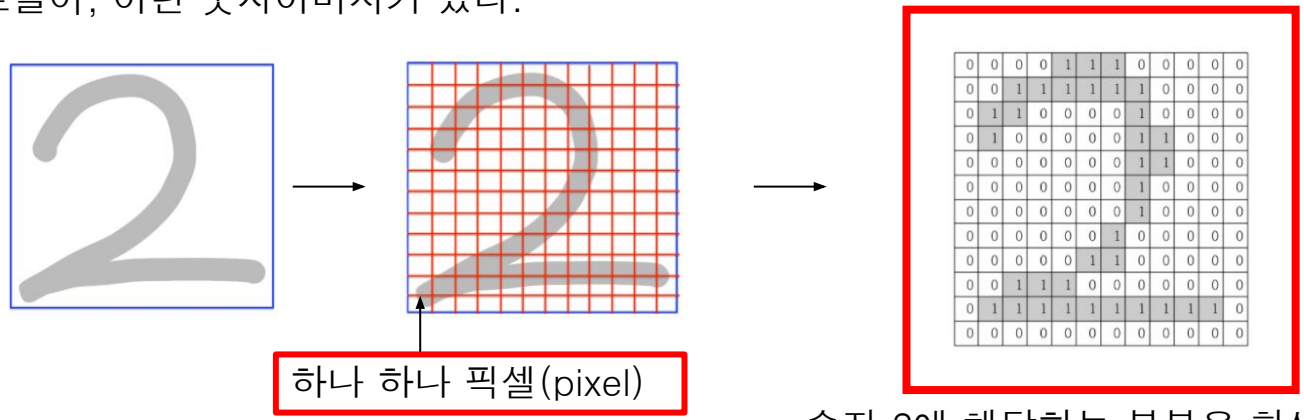


CNN 구성요소

- Convolution (합성곱)
- Channel (채널)
- Filiter (필터)
- Kernel (커널)
- Stride (스트라이드)
- Padding (패딩)
- Feature Map (피처 맵)
- Acivation Map (액티베이션 맵)
- Pooling Layer (풀링 레이어)

Convolution (합성곱)

예로들어, 이런 숫자이미지가 있다.

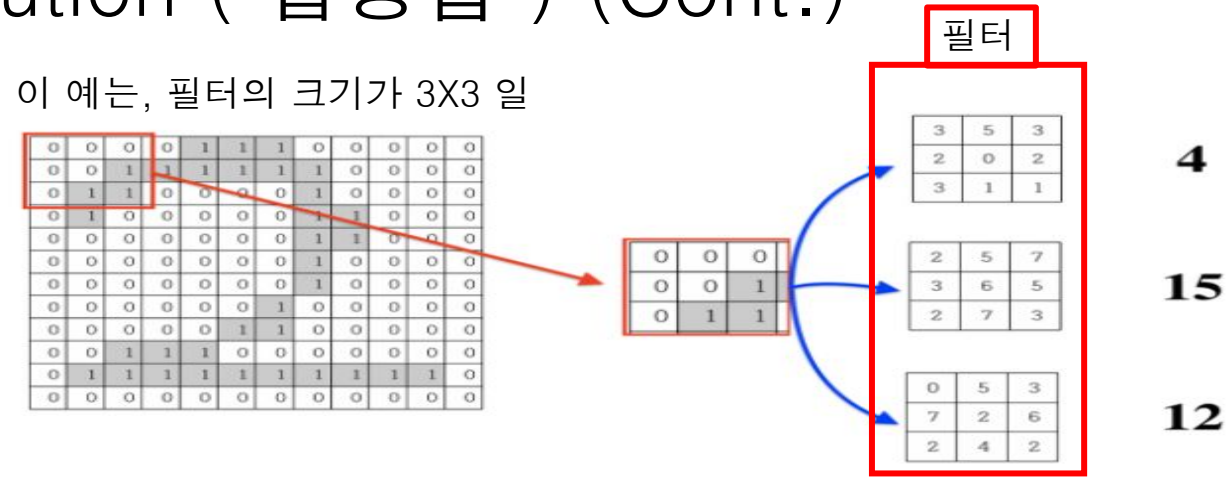


숫자 2에 해당하는 부분은 회색 (1 에 해당)
그외 흰 부분은 (0 에 해당)

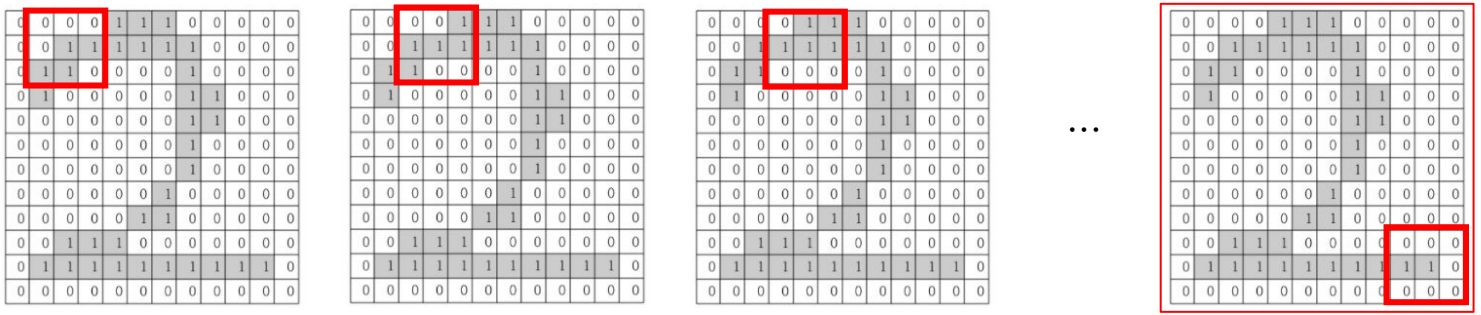
- CNN에서는 이런 이미지의 한 픽셀과 주변 픽셀들의 연관관계를 유지시키며 학습하는 것이 목표
- 즉, 하나의 이미지로부터 픽셀 간이 연관성을 살린 여러 개의 이미지를 생성하는 것부터 시작

Convolution (합성곱) (Cont.)

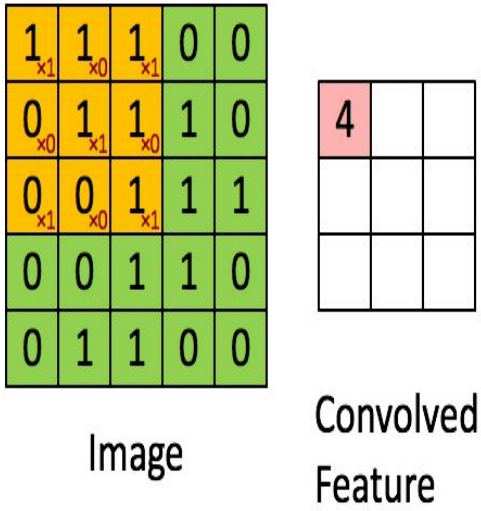
[참고] 이 예는, 필터의 크기가 3X3 일 경우



위와 같은 합성곱 연산을



Convolution (합성곱) (Cont.)



- 왼쪽의 경우, 각 스텝당 1개의 value얻게 되고,
  이 과정을 반복하면, 3x3의 output을 얻게 됨
  (value를 구하는 과정이, 하나씩 규칙적으로 보임)
- 이 때,전체 이미지 크기(NxN)보다 필터가 작으니깐
  필터를 “옮겨가며 " 반복하는 과정이 필요한데,
  이 때 “ 얼마나 “ 옮겨가며 진행해야하는가 ? 가 정해져야 하고
  우리는 이 옮기는 양을 “ stride(스트라이트) “ 라고 한다.
  (왼쪽의 예는, stride = 1 이겠죠 ?)
- 실제 [illegible] 는 다음과 같은 공식이 있다.

Total size ; N (i.e. N x N)

Filter size ; F (i.e. F x F)

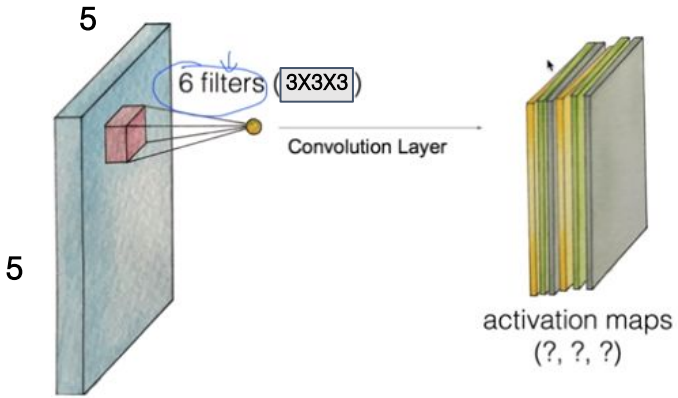
Output size ; k:=(N-F)/stride + 1 where k is an integer.

위의 예에서는, k = (5−3)/1(stride) + 1 = 3 이고, 오른쪽에 이 값처럼, 3x3의 output이 나옴

위처럼 나온 output을 Convolution layer 라고 한다.

필터가 N 개면, output Convolution Layer도 N개가 나올 것이다.

Convolution (합성곱) (Cont.)



- 예를들어, 6개의 filter로 부터, 우리가 6개의 Convolution Layer를 얻었다 치자.
- 현재 왼쪽의 예제에서 그럼, N = 5, F = 3, Stride = 1 이기 때문에, 3X3 짜리 Convolution Layer를 6개 얻은 셈이다.
- 이를 하나로 합치면, 3X3X6 짜리 Convolution Layers (또는 **Activation Maps**) 를 얻게 되는 것이다.
- 이렇게, 일련의 Activation Maps(Convolution Layers)까지 얻는 과정을 우리는 Convolution 이라고 하는 것이다.
- 이런 Convolution은 한 번만 할 수도 있지만, 상황에 따라서는, 여러 번의 Convolution을 해도 되는 것이다.

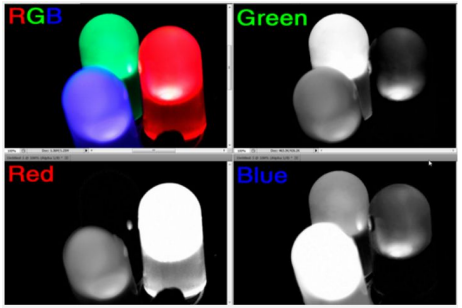
필터(Filter)

뒷 발표자와 겹치므로 생략

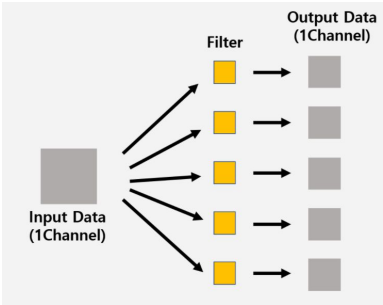
(중요하지 않아서 생략하는거 절대 아님 !!)

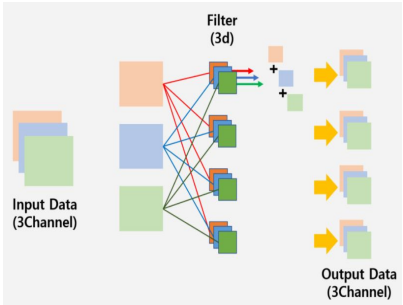
채널(Channel)

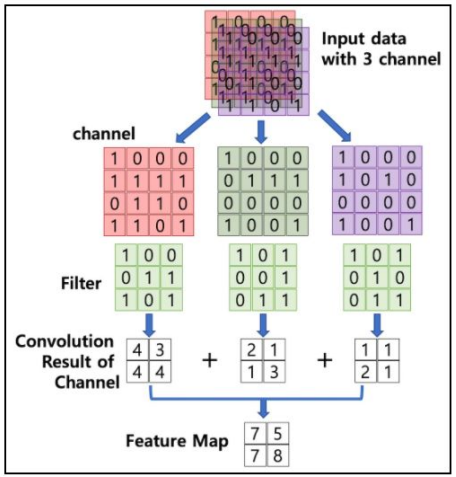
우리가 보통 생각하는 컬러 이미지는 RGB 채널이 합쳐진 이미지, 즉 하나의 컬러이미지는 3개의 채널로 구성



보통은, 연산량을 줄이기 위해 (오차를 줄이기 위해) 전처리에서,
이미지를 흑백(Channel = 1)으로 만들어 처리한다.
(* 컬러 이미지 (Channel = 3))



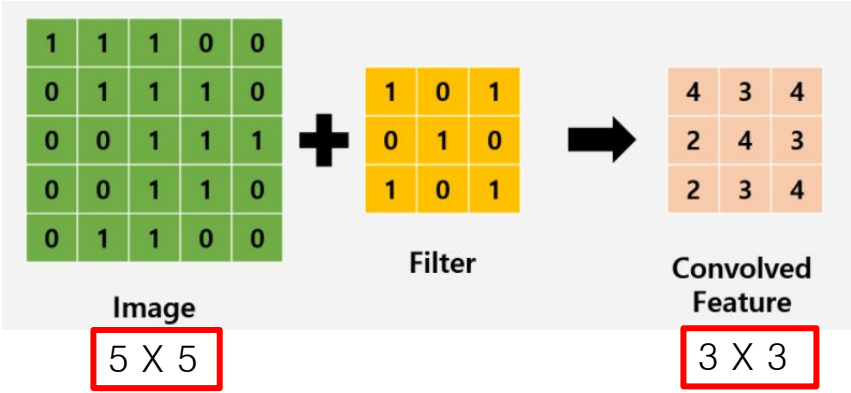




컬러 이미지같이, Multi Channel CNN에서 확인해야 하는 것은

- Input data의 channel수 == filter의 channel수
- Input data의 channe수와 관계없이 filter의 개수만큼 output이 나옴

패딩(Padding)



- Convolution Layer에서는 filter와 Stride의 작용으로, 결국 Feature Map(output) 크기는, Input data보다 작다.

- 근데, 이렇게 입력데이터보다, 출력데이터가 작아지는것을 방지하는 방법이 Padding이다.
- 즉, Convolution 과정에서의 데이터의 손실을 방지하겠다는 것

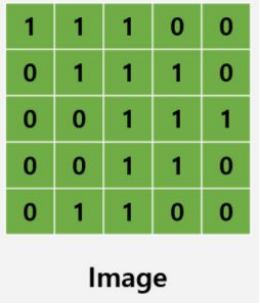
패딩(Padding)

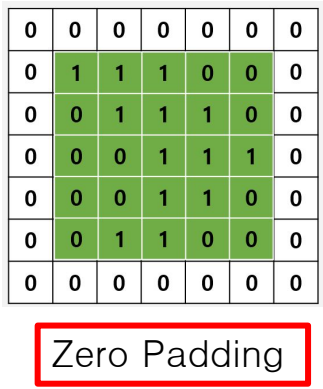
[참고

Total size ; N (i.e. N x N)

Filter size ; F (i.e. F x F)

Output size ; k:=(N-F)/stride + 1 where k is an integer.





Zero Padding

전 슬라이드에서, 소개한 합성곱 연산에 의한 Output크기를 도출해보자.

N size = 7(5+2), F size= 3 , stride = 1 일 때
(전과 동일한 조건의, filter와 stride 크기)

결과 : k = ((7−3)/1) + 1 = 5 << " 기존 입력 데이터 크기 유지 !! "

보통, 가장자리를 0 값으로 양 옆을 열과 행을 추가한다.

Padding에는 두가지 옵션이 있다.

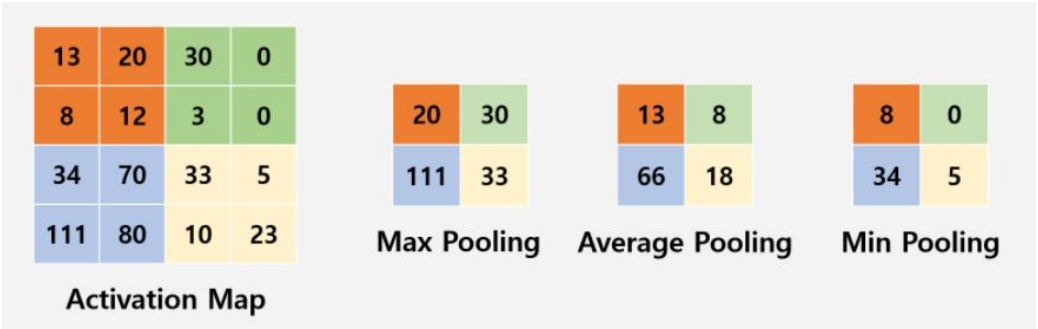
- Valid = Padding 0을 뜻한다. 즉, 입력보다 출력의 크기가 작아진다.
- Same = Padding이 존재, 즉, 입력과 출력 크기가 같아진다.

Pooling(풀링)

모두의 딥러닝에선,, **Pooling이란**
: Convolution을 거쳐서 나온 Activation Maps가 있을 때, 이를 이루는 Convolution Layer를 resizing하여 새로운 Layer를 얻는 것

- Max pooling (★ CNN에서 주로 사용)
- Mean pooling
- Min pooling

Max pooling시, 노이즈가 감소하고, 속도가 빨라지며
영상의 분별력이 좋아진다고 한다.



일반적으로, pooling의 크기와
stride 크기를 같게 설정하여
모든 원소가 한번씩은 처리가 되도록
설정한다고 한다.

Pooling(풀링)

- 근데, 왜 Pooling을 해야할까 ?

  일단, 결론은 **Overfitting을 방지하기 위함이라 한다** .

  feature := elements of input data

  parameter := elements of weight matrix

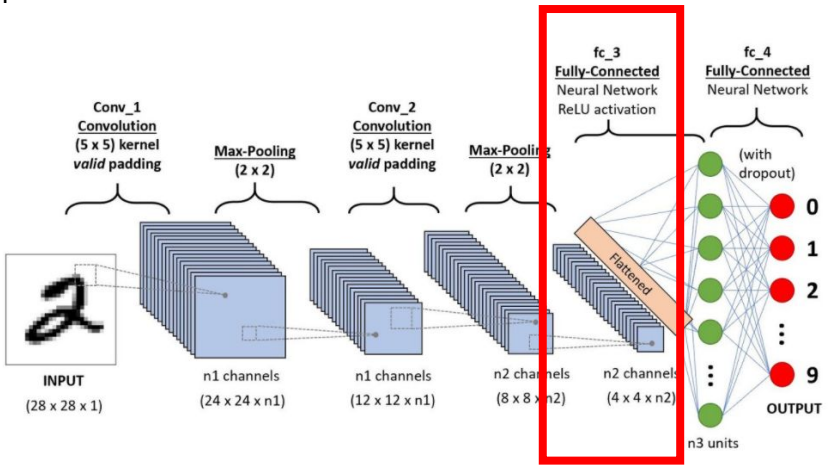
예로 들어, 우리가 size가 96 X 96 인 이미지가 주어졌고, (즉, featur의 수는 96 X 96 개 (단 padding 적용))
이를 400개의 필터로 Convolution한 size 8 X 8 X 400의 Convolution Layers가 있다치자.

(Stride = 1 이라 가정) 그럼, Convolution Layer에는 (96-8+1) X (96-8+1) = 89 X 89 = 7921개의 Feature가 있고, 이런 Layer가 400개니깐, 결과적으로 89 X 89 X 400 = 3,168,400개가 된다.
이 정도로, feature가 많아지면, overfitting이 분명히 있을 것이다.

결과적으로, 중간중간 Padding과 함께, "선택적"으로 Pooling과정을 도입하면
이러한, 문제들이 해소 될 것이이다.

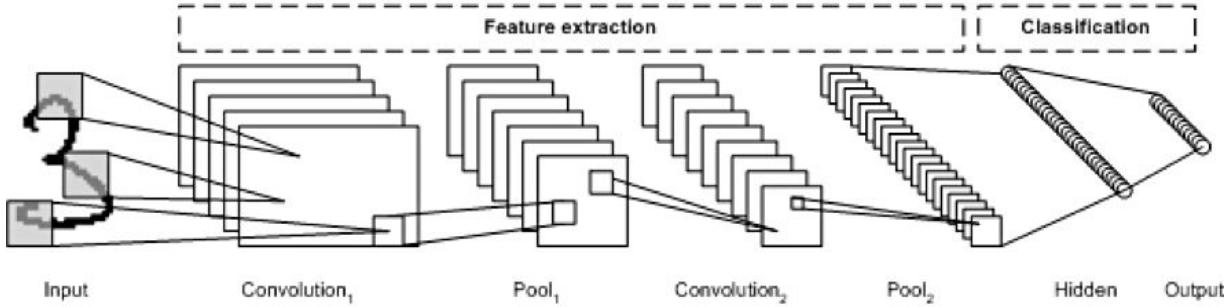
Flatten (저번주, 소개한 내용이긴 함)

- Flatten Layer는 CNN의 데이터 타입을 Fully Connected Neural Network(완전 연결 네트워크 층)
  형태로 변경하는 레이어



- Convolution과 Pooling을 반복해주면, 이미지의 숫자는 많아지면, 크기는 점점 줄어들게 됨
- 즉, NxN이미지는 이미지라기보단, 이미지에서 얻어온, 하나의 특이점 데이터 형태가 됨
- = 1차원 Row데이터로 취급해도 무관한 상태

전체 구조 정리



- 특징 추출 단계(Feature Extraction)
  - Convolution Layer : 필터를 통해 이미지 특징을 추출
  - Pooling Layer : 특징을 강화시키고 이미지의 크기를 줄임

- 이미지 분류 단계 (Classification)
  - Flatten Layer : 데이터 타입을 FC(완전 연결 층)네트워크 형태로 변경. 입력데이터의 shape 변경만 수행
  - Softmax Layer : Classification 수행

CNN은 Convolution과 Pooling을 반복적으로 사용하면서 불변하는 특징을 찾고, 그 특징을 입력데이터로
Fully-Connected 신경망에 보내 Classification을 수행한다.

2.1 CNN 에서의 필터의 의미

- 필터(Kernel)(2D matrix)

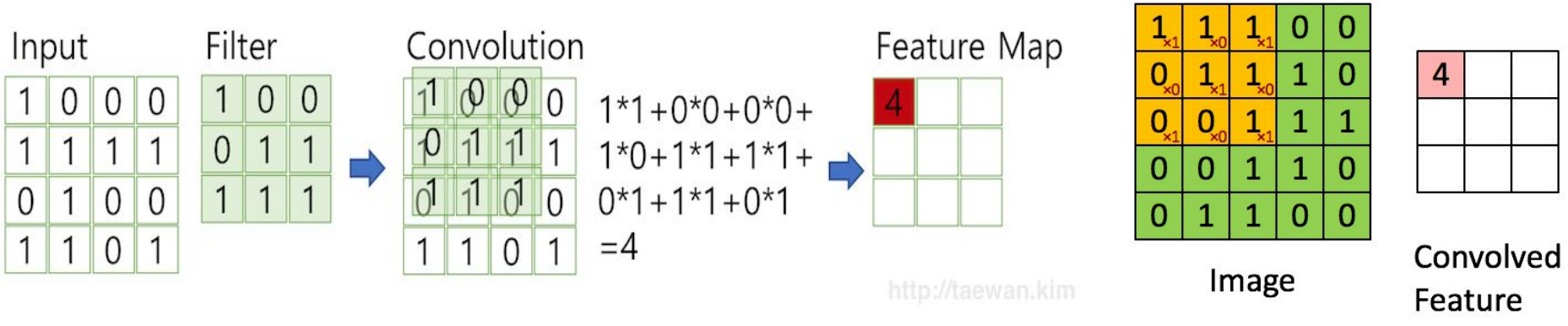
: 입력층과의 합성곱을 위한 가중치 포함한 행렬

- 보통 (3x3) , (5x5) 크기 사용
- 합성곱을 통해 이미지의 feature를 뽑는다

2.2 필터의 계산 과정

- 과정

  : 입령층 (합성곱)X 필터 => 출력층

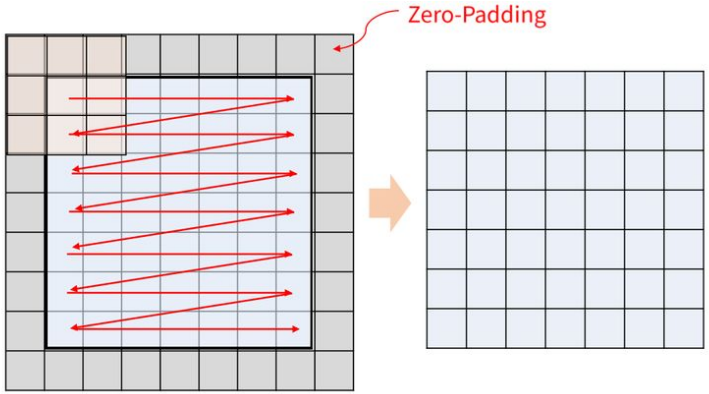


2.3 필터의 데이터량 조절

- 패딩

  : 입력층 가장자리에 행과 열을 추가

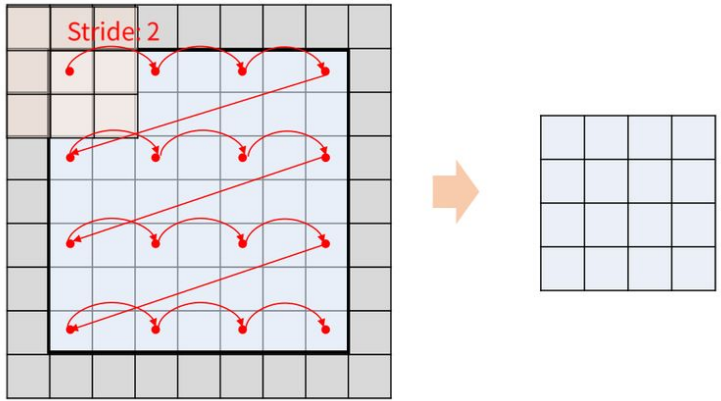
  - 데이터의 누락 방지



- 스트라이드

  : 필터의 이동 칸 수 조절

  - 학습량 조절 -> 과대적합 방지



2.4 필터의 종류

- Sobel Filter

  : 이미지의 가로세로 feature를 뽑아낼 수 있는 필터

- **Gaussian Filter**

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

  : 가우시안 함수이용
  잡음 제거 **=>** 정중앙이 가장 크고 각 방면으로 작아지는 가우시안 분포형

Sobel filter

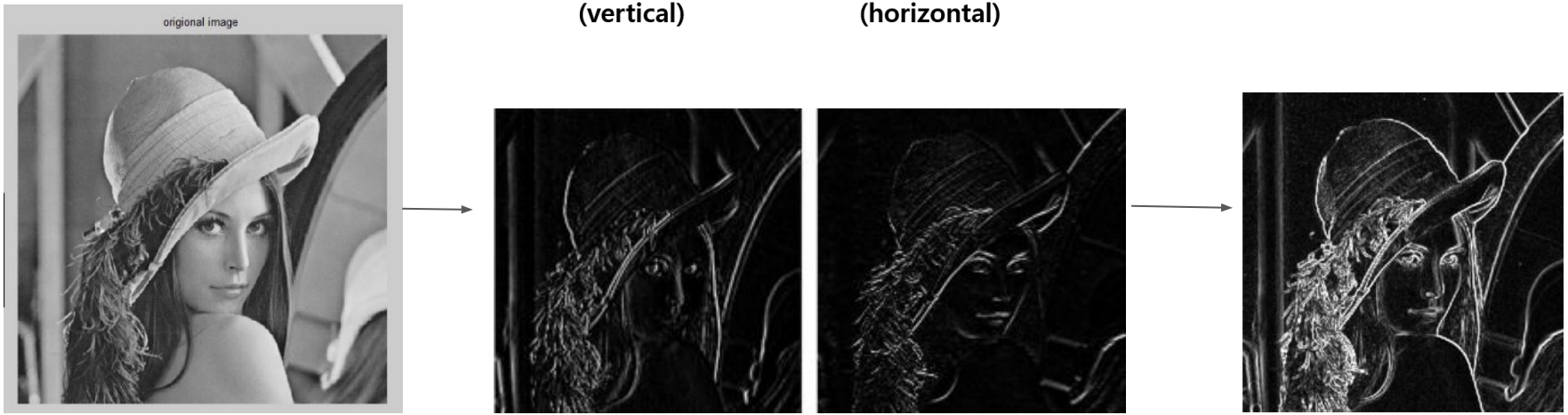
: 이미지의 가로세로 feature를 뽑아낼 수 있는 필터

-1	0	+1
-2	0	+2
-1	0	+1

Sobel-X (vertical)

+1	+2	+1
0	0	0
-1	-2	-1

Sobel-Y (horizontal)



Gaussian Filter

: 잡음 제거 => 정중앙이 가장 크고 각 방면으로 작아지는 가우시안 분포형

$$* \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} =$$

Gaussian Filter 실습

```
from google.colab import files
uploaded = files.upload()
```

```
import cv2
import numpy as np
from PIL import Image

im = Image.open('tae.jpg') # Image load
im_array = np.asarray(im) # Image to np.array

kernel1d = cv2.getGaussianKernel(5, 3) # 5크기의 1차원벡터, 시그마3,
kernel2d = np.outer(kernel1d, kernel1d.transpose()) # 전치시켜서 2차원벡터로

low_im_array = cv2.filter2D(im_array, -1, kernel2d) # convolve

low_im = Image.fromarray(low_im_array) # np.array to Image
low_im.save('tae.bmp','BMP')
```

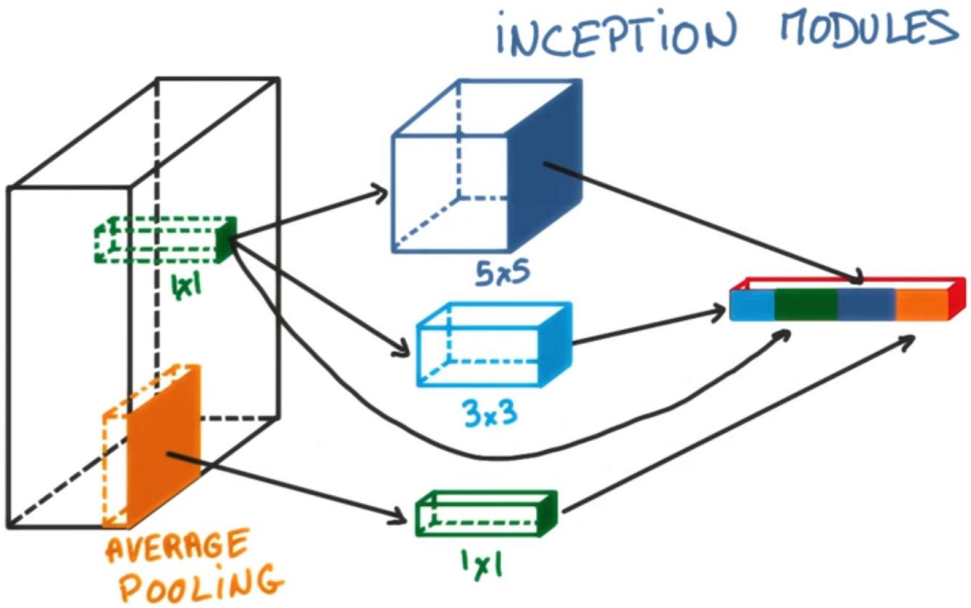
Gaussian Filter 실습

2.5 다양한 필터의 사용

보통 하나의 layer에 한종류의 필터사용 하지만

- **GoogleNet 모델**

  : 여러 종류의 필터사용



3.1 NN(Neural Network)에서의 역전파 알고리즘

- CNN의 역전파 알고리즘을 이해하기 위해서는 NN의 역전파 알고리즘을 이해해야함
- 표기법

1. 인덱스 : 모든 인덱스는 0에서 시작합니다
2. 행렬에서 행과 열의 크기는 해당 인덱스의 대문자를 사용합니다.
3. C : 코스트 함수를 나타냅니다.
4. L, l : L, l 번째 층, 특히 대문자 L 은 출력층을 나타냅니다.
5. w_{ij}^{l} : $l-1$번째 층과 l 번째 층을 연결하는 가중치weight의 i, j 번째 요소입니다. 완전 연결층Fully connected layer인 경우 j 가 $l-1$ 번째 층의 인덱스, i 가 l 번째 층의 인덱스임을 유의해야 합니다.
6. z_{ij}^{l} : l번째 층의 가중합weighted sum의 i, j번째 요소입니다.
7. a_{ij}^{l} : l번째 층의 활성값activation value의 i, j번째 요소입니다.
8. $\sigma(x)$: 활성화 함수를 나타냅니다.
9. 가중치, 바이어스, 가중합, 활성값 벡터 또는 행렬은 모두 볼드 문자로 표시합니다.
10. 모든 식은 특별히 언급이 없는 한 인덱스 표기법을 기본으로 사용합니다.

3.2 NN의 역전파 알고리즘 수식

- CNN 역전파 알고리즘의 수식 == NN의 역전파 알고리즘 수식
- 첫번째와 두번째 수식을 이용하여 모든 층의 δ를 구함
- δ를 이용하여 세번째 네번째 수식을 통해 코스트함수의 편미분 값을 구함

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

(BP1)

$$\delta_j^l = \sum_k w_{kj}^{l+1}\delta_k^{l+1}\sigma'(z_j^l)$$

(BP2)

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

(BP3)

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$$

(BP4)

3.3 CNN의 역전파 알고리즘 수식 -Convolution과 Correlation

- 첫번째 식은 코릴레이션 두번째 식은 컨벌루션이라 함.
- 차이점:
  - 코릴레이션은 필터를 입력 배열에 그대로 겹쳐 놓고 같은 위치에 있는 요소끼리 곱함
  - 컨벌루션은 필터를 180도 회전시켜서 같은 연산을 수행 I : 2차원 입력 배열, F : 필터, k는 필터의 크기

$$C(x,y) = \sum_{a=0}^{k-1}\sum_{b=0}^{k-1} I(x-a, y-b)F(a,b)$$

식(1)

$$C(x,y) = \sum_{a=0}^{k-1}\sum_{b=0}^{k-1} I(x+a, y+b)F(a,b)$$

식(2)

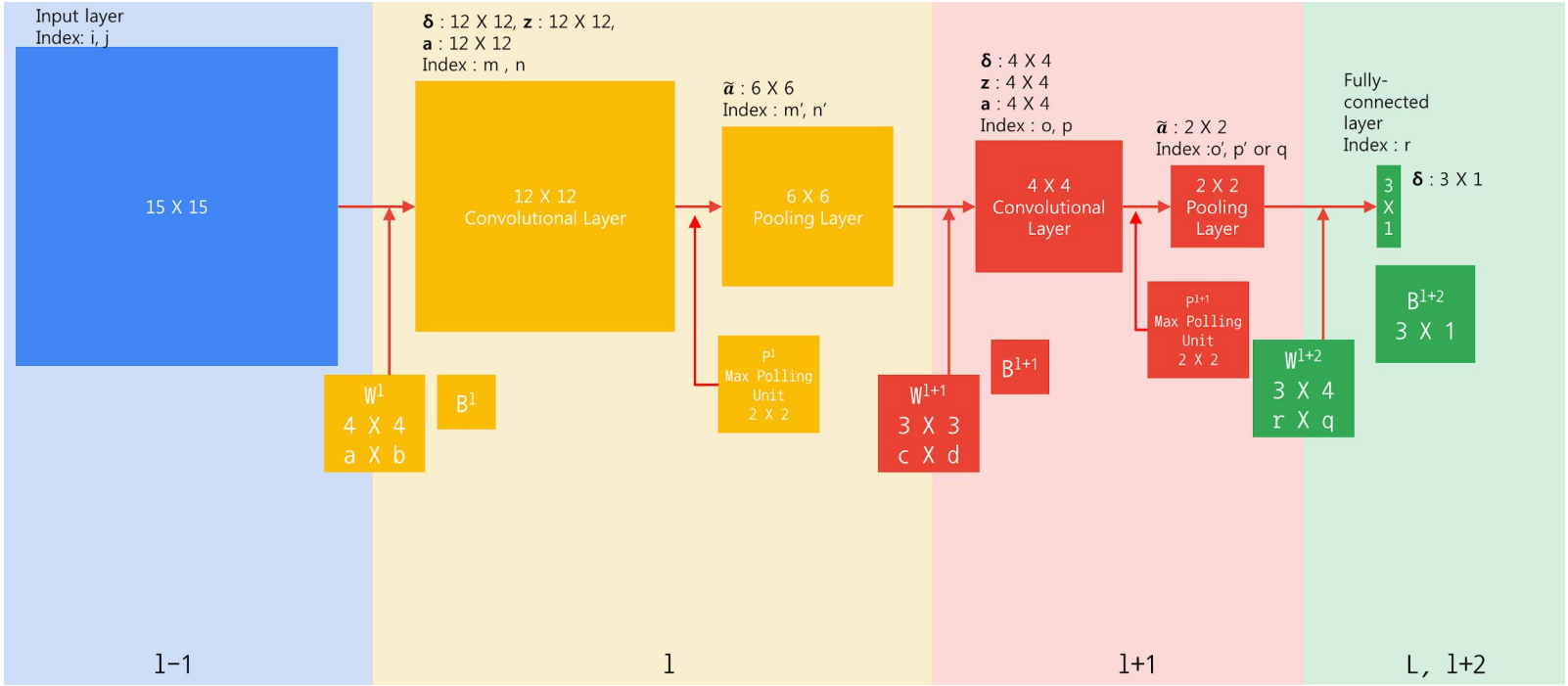
  - 컨벌루션의 수학적 정의가 식(3)과 같기 때문에 곱하고자 하는 함수를 반전시켜야 합니다.
  - (g함수의 τ에 –가 곱해짐) 이와 동일한 행위가 2차원 배열에서는 180도 회전하는 것

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(\tau)g(-\tau + t)d\tau$$

식(3)

- convolution과 Correlation을 통틀어 컨벌루션이라고 하는 경우도 많음

3.4 CNN의 역전파 예제



3.5 단계별 역전파

- 식(4)는 예제 CNN의 I번째 CONV 층에서의 가중 합을 코릴레이션을 사용하여 나타낸 것

식(4) $$z_{mn}^{l} = \sum_{a=0}^{A-1} \sum_{b=0}^{B-1} w_{ab}^{l} a_{(m+a)(n+b)}^{l-1} + b^{l}$$

- 위 식에서 필터 w의 크기는 A x B(전술한 바와 같이 인덱스의 대문자로 크기를 나타냄)
- 식(5)는 마지막 FC층의 가중합을 나타냅니다. 일반적인 NN에서 익숙하던 모습

식(5) $$z_{r}^{L} = \sum_{q} w_{rq}^{l+2} a_{q}^{l+1} + b_{r}^{l+2}$$

- 가중합을 활성화 함수를 통해 활성값으로 만듦

식(6) $$a_{mn}^{l} = \sigma(z_{mn}^{l})$$

풀링을 사용한다면 정해진 풀링 유닛 크기만큼 활성값을 평균 내어 하나의 값을 구하거나(mean pooling), 풀링 유닛의 크기에 해당하는 활성값들 중 최대값을 선택(max pooling)하는 것으로 풀링층을 만들어 낼 수 있음

3.5 단계별 역전파

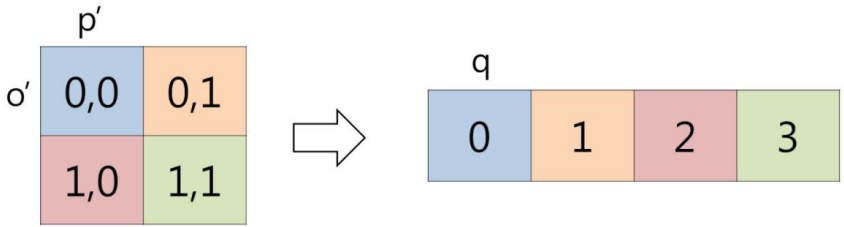
- 0 단계:식(4)를 사용하여 네트워크 전체를 포워드패스,마지막 완전연결 되어있는 층은 식(5)를 사용
- 1 단계: (BP1)을 사용하여 출력층의 δ^L을 구합니다. 출력층은 l+1번째 층과 완전연결 되어있는 층이므로 기존 역전파식과 달라질 것이 없습니다. 따라서 다음 식(CBP1)을 이용하여 출력층의 δ^L을 구합니다.

식(CBP1) $\delta_r^L = \frac{\partial C}{\partial a_r^L}\sigma'(z_r^L)$

- 2단계:원래 역전파 알고리즘은 모든 층의 δ를 구하고 다시 출력층으로 돌아와 w와b에 대한 편미분을 구하는 순서로 진행, 편의상 델타를 구한 후 바로 w와 b의 편미분 진행

$\frac{\partial C}{\partial b_r^{l+2}} = \delta_r^{l+2}$ 단계 1을 통해서 구함

$\frac{\partial C}{\partial w_{rq}^{l+2}} = \tilde{a}_q^{l+1}\delta_r^{l+2}$ 포워드패스 과정에서 구해진 값



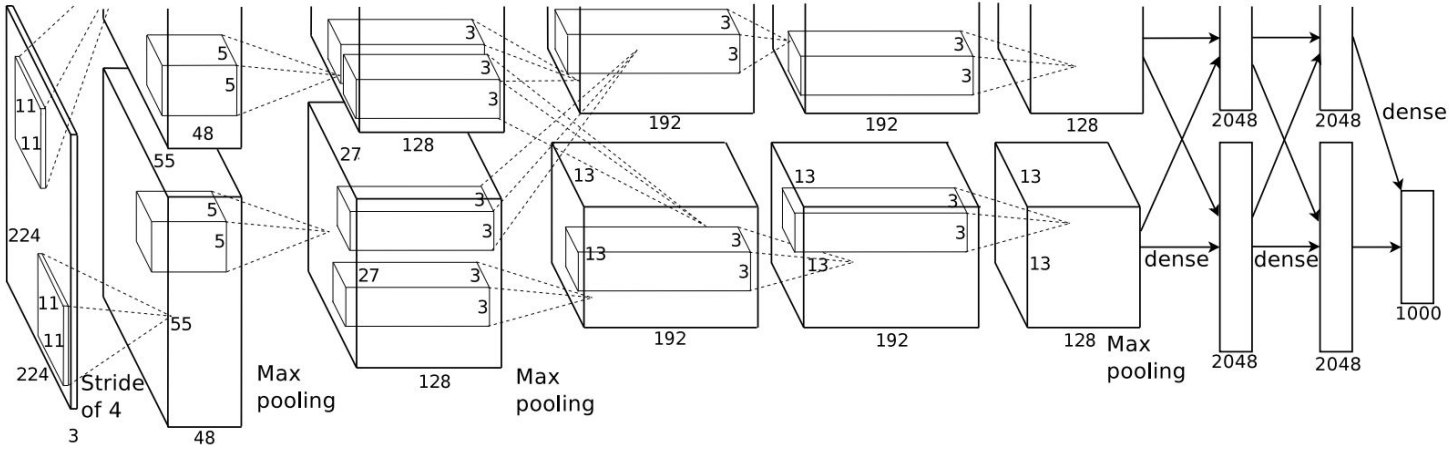
- l+1번째 층의 출력 a~는 o', p′의 인덱스를 가지는 2차원 배열 형태이지만 그림처럼 1차원 배열로 간주하고 인덱스 q를 사용 식(7)참고 $q = o'P + p'$ 식(7)

3.6 결론

- CNN에서의 역전파 수식을 인덱스형으로 모두 알 수 있다.
- 합성함수의 편미분은 미분하고자 하는 함수를 블록다이어그램으로 표시하고 그 다이어그램을 역으로 거슬러 올라가는 과정으로 모델링될 수 있다.
- 역전파 중 나타나는 컨벌루션은 인위적인 과정이 아니라 편미분하는 과정에서 자연스럽게 나타난다.
- 역전파 중 나타나는 up-sampling 역시 인위적으로 행렬의 모양을 맞추기 위한 과정이 아니라 max 또는 mean 함수를 미분하는 과정에서 자연스럽게 일어난다.
- https://metamath1.github.io/cnn/index.html

4.CNN의 종류

1. Alexnet


224
224
3
11
11
11
11
Stride
of 4
48
55
5
5
48
55
5
5
Max
pooling
27
128
128
3
3
3
3
3
3
Max
pooling
192
192
128
13
13
13
13
13
13
192
192
128
3
3
3
3
3
3
Max
pooling
2048
2048
dense
dense
dense
2048
2048
1000

1.Alexnet

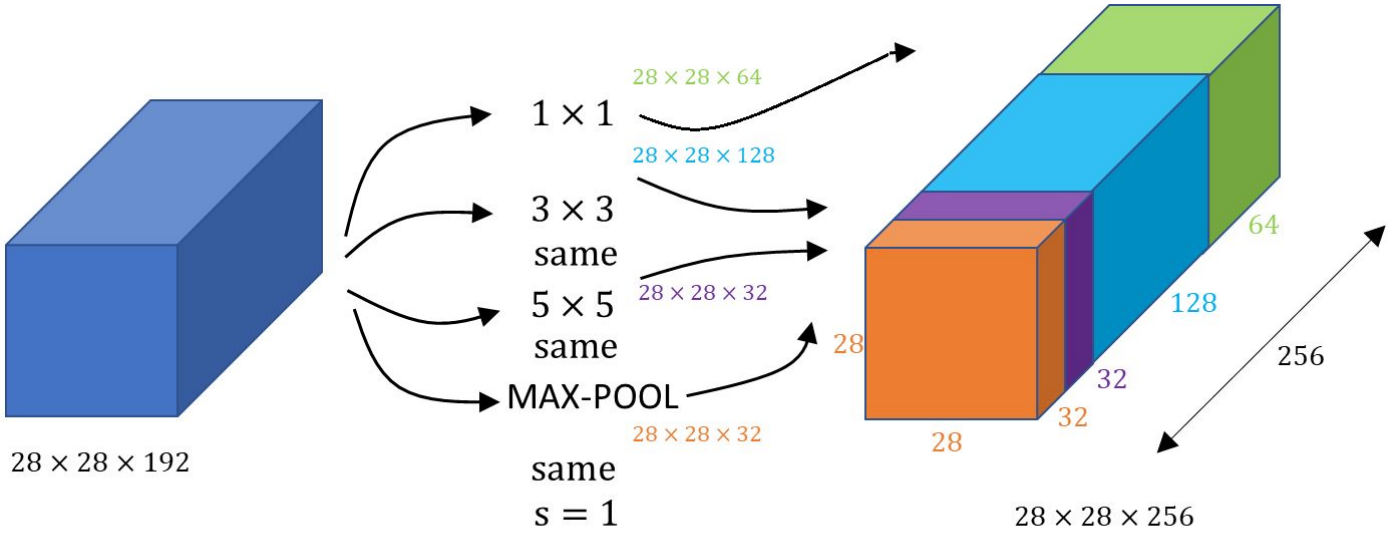
Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

1.Alexnet

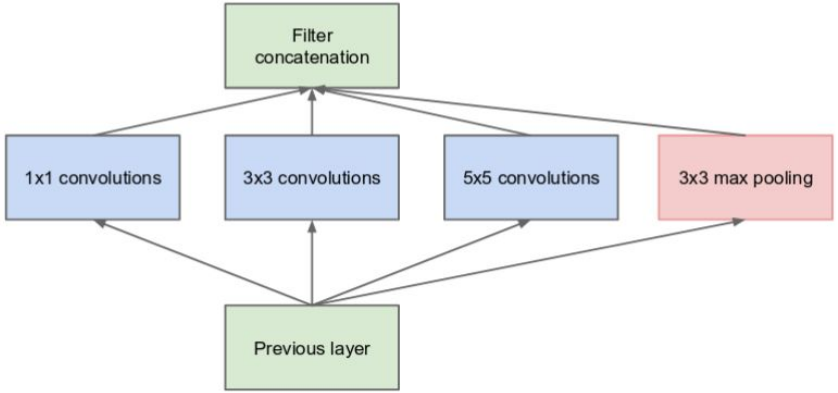
```
model = Sequential()
# 1번째 합성곱 계층
model.add(Conv2D(filters=96, input_shape=(224,224,3), activation='relu',kernel_size=(11,11), strides=(4,4)))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

# 2번째 합성곱 층
model.add(Conv2D(256, activation='relu',kernel_size=(11,11), strides=(1,1)))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

# 3번째 합성곱 층
model.add(Conv2D(384, activation='relu',kernel_size=(11,11), strides=(1,1)))

# 4번째 합성곱 층
model.add(Conv2D(384, activation='relu',kernel_size=(11,11), strides=(1,1)))

# 5번째 합성곱 층
model.add(Conv2D(256, activation='relu',kernel_size=(11,11), strides=(1,1)))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
```

```
# 1번째 완전 연결층
model.add(Flatten())
model.add(Dense(4096, input_shape=(224*224*3,),activation='relu'))
model.add(Dropout(0.4))

# 2번째 완전 연결층
model.add(Dense(4096))
model.add(Activation='relu')
model.add(Dropout(0.4))

# 3번째 완전 연결층
model.add(Dense(1000))
model.add(Activation='relu')
model.add(Dropout(0.4))

model.add(Dense(17))
model.add(Activation='relu')
model.add(Activation='softmax')
```

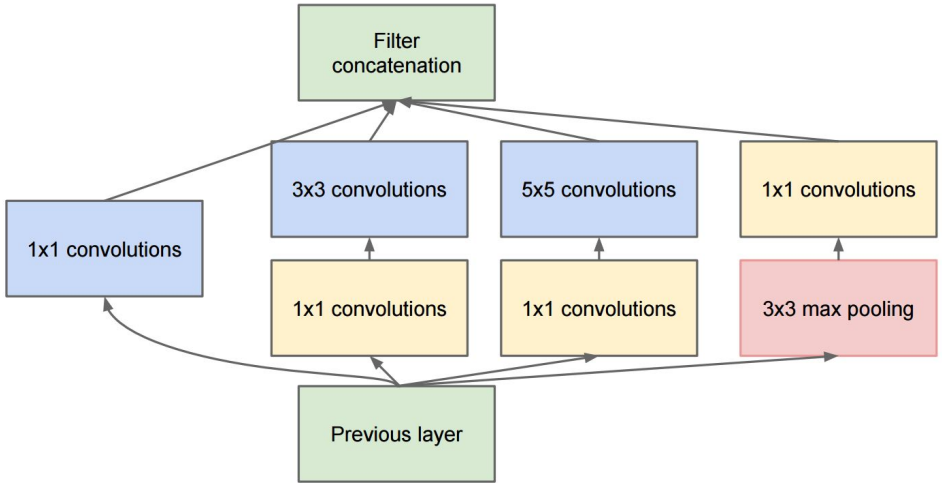
2. googlenet


28 × 28 × 192
1 × 1
28 × 28 × 64
3 × 3
same
28 × 28 × 128
5 × 5
same
28 × 28 × 32
MAX-POOL
same
s = 1
28 × 28 × 32
64
128
32
32
28
28
256
28 × 28 × 256

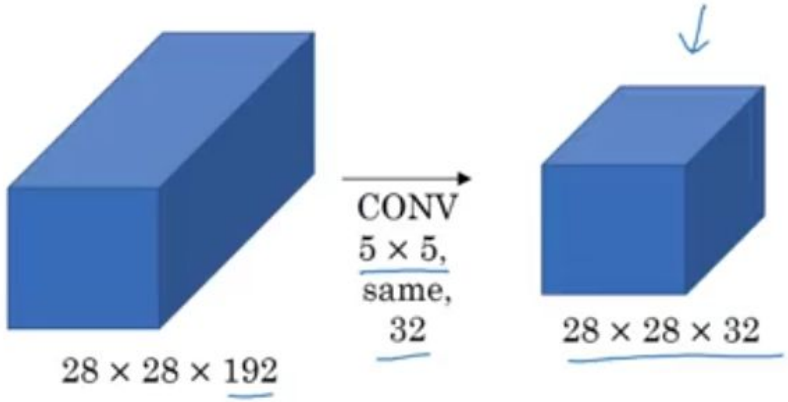
2. googlenet



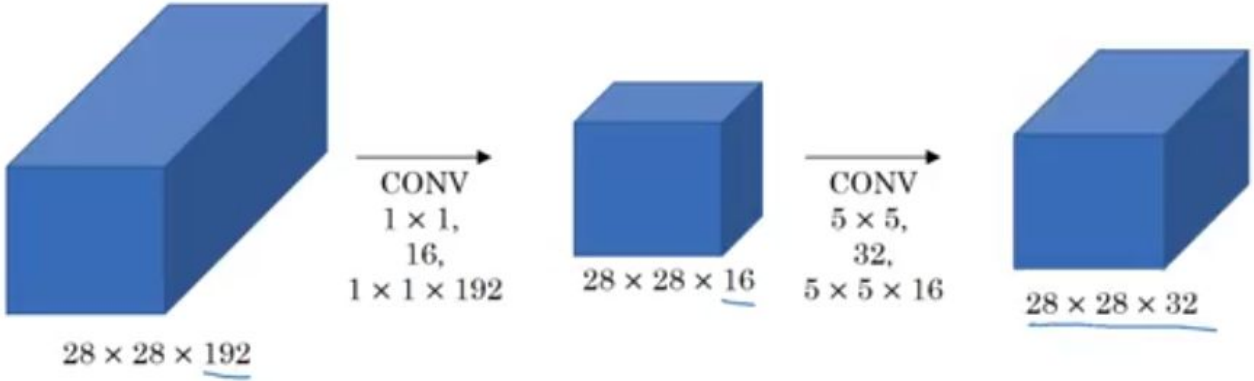
(a) Inception module, naïve version



2. googlenet

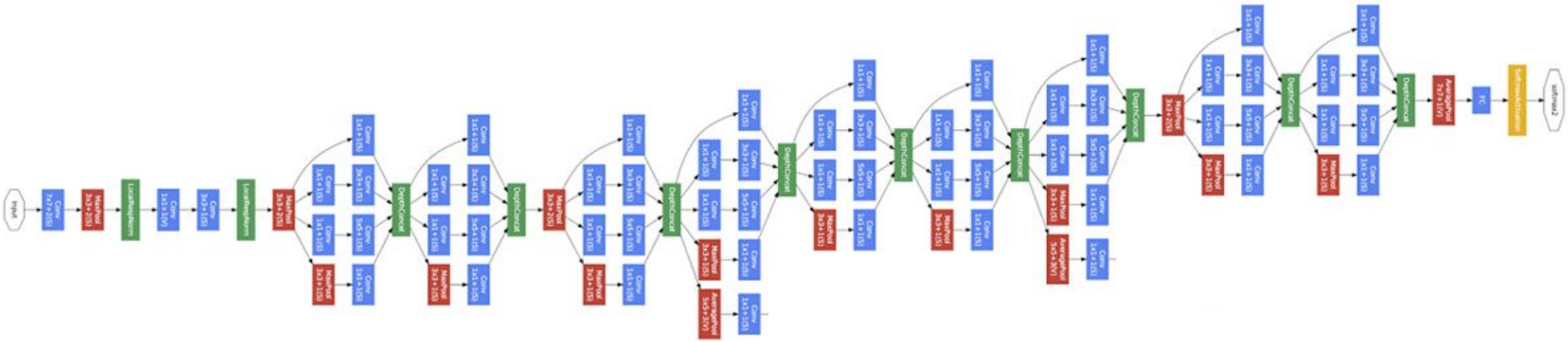


계산수 = (5 * 5 * 192) * (28 * 28* 32)=약 1억 2천만번

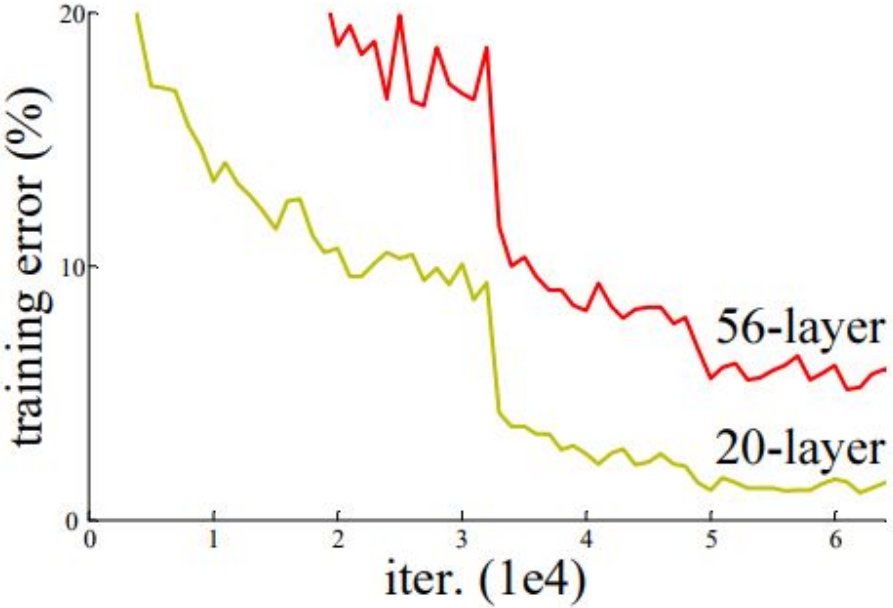


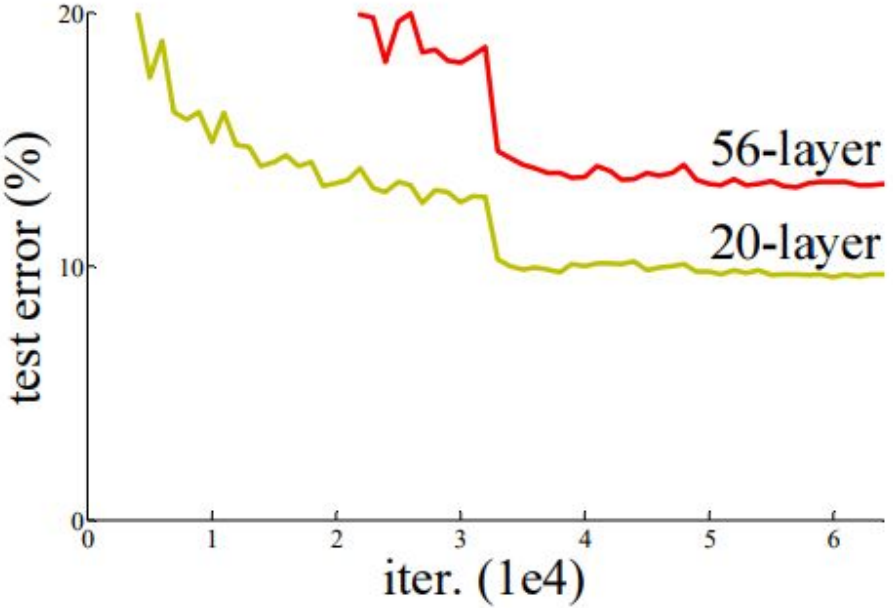
계산수 = (1*1*192) * (28*28*16) + (5*5*16) * (28*28*32) = 약 1200만번
계산수를 비교하면 1X1 필터를 적용시 약 0.1배로 줄어듬

2. googlenet



3. Resnet


training error (%)
20
10
0
0
1
2
3
4
5
6
iter. (1e4)
56-layer
20-layer


test error (%)
20
10
0
0
1
2
3
4
5
6
iter. (1e4)
56-layer
20-layer

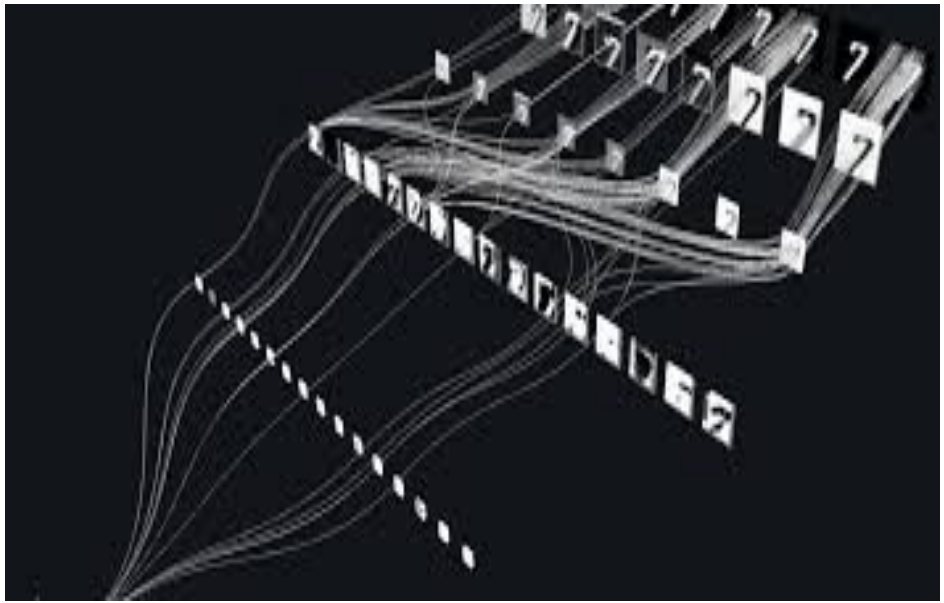
Why?

3.Resnet

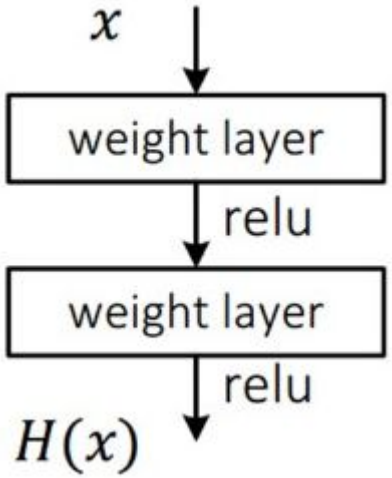
layer가 깊어지면 미분을 많이 시행

backpropagation을 해도 앞의 **layer**일 수록 미분값이 작아짐

그만큼 **output**에 미치는 가중치가 작아져서 **train data**를 제대로 학습하지 못함.



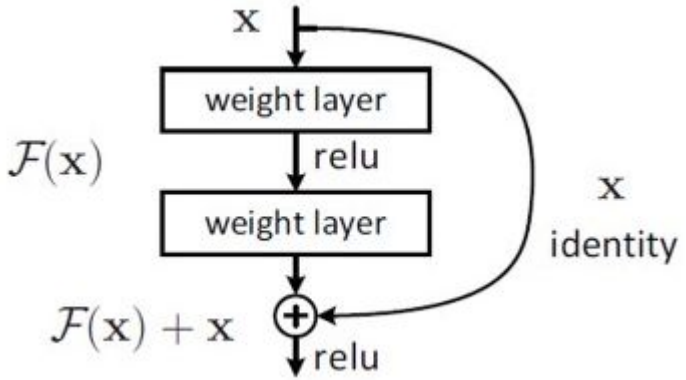
3. Resnet



네트워크의 output이 x 가 되도록? X

마지막에 x 를 더해서 네트워크의 output이 0이 되도록!

(이래야 gradient vanishing 현상이 발생하지 않음.)



3. Resnet

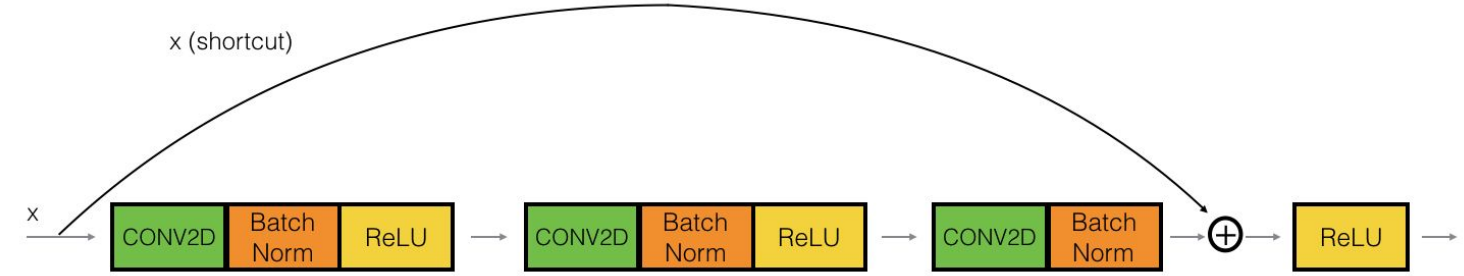
1. 이미지에서는 **H(x) = x**가 되도록 학습시킨다.

2. 네트워크의 **output F(x)**는 **0**이 되도록 학습시킨다.

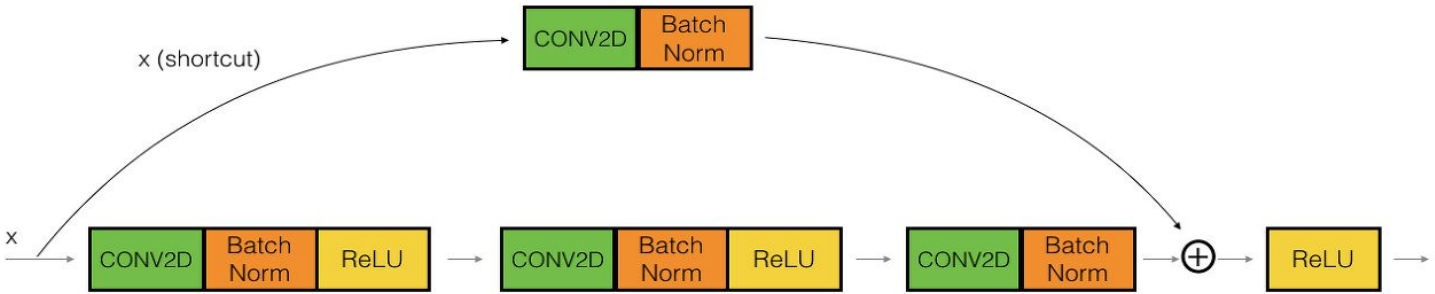
3. F(x)+x=H(x)=x가 되도록 학습시키면 미분해도 **F(x)+x**의 미분값은 **F'(x) + 1**로 최소 **1**이상이다.

4. 모든 **layer**에서의 **gradient**가 **1+F'(x)**이므로 **gradient vanishing**현상을 해결했다.

3. Resnet

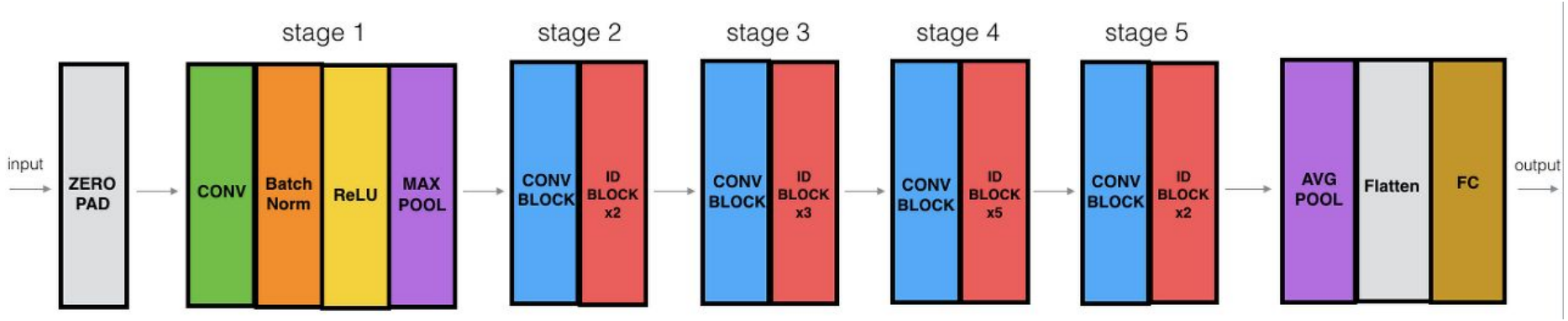


identify block



convolution block

3. Resnet


stage 1
stage 2
stage 3
stage 4
stage 5
input
ZERO PAD
CONV
Batch Norm
ReLU
MAX POOL
CONV BLOCK
ID BLOCK x2
CONV BLOCK
ID BLOCK x3
CONV BLOCK
ID BLOCK x5
CONV BLOCK
ID BLOCK x2
AVG POOL
Flatten
FC
output

4.Resnet 초간단실습

Dataset : Cifar 10

32*32 크기의 image data

60000개의 data,

10개의 class

50000개의 train data,

10000개의 test data

4.1 Resnet 초간단실습(1) layer 잔뜩 쌓기

```
from keras import layers
from keras import models

model = models.Sequential()
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(128,(3,3), activation='relu', input_shape=(32,32,3)))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test,  y_test, verbose=2)
```

```
Train on 50000 samples
Epoch 1/5
50000/50000 [==============================] - 16s 320us/sample - loss: 1.5299 - accuracy: 0.4353
Epoch 2/5
50000/50000 [==============================] - 12s 237us/sample - loss: 1.0676 - accuracy: 0.6203
Epoch 3/5
50000/50000 [==============================] - 12s 238us/sample - loss: 0.8887 - accuracy: 0.6868
Epoch 4/5
50000/50000 [==============================] - 12s 238us/sample - loss: 0.7801 - accuracy: 0.7255
Epoch 5/5
50000/50000 [==============================] - 12s 238us/sample - loss: 0.6998 - accuracy: 0.7539
10000/1 - 1s - loss: 0.7311 - accuracy: 0.7165
[0.8267595977783203, 0.7165]
```

4.2 Resnet 초간단실습 (residual block 적용)

```
inputs = keras.Input(shape=(32, 32, 3))
x = inputs
#x = layers.MaxPooling2D(2)(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(_x)
x = _x
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(_x)
x = x + _x
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(_x)
x = x + _x
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(_x)
x = x + _x
x = layers.MaxPooling2D(2)(x)

x = layers.Flatten()(x)
x = layers.Dense(128)(x)
x = layers.Dense(128)(x)
x = layers.Dense(10, activation='softmax')(x)
outputs = x

model = keras.Model(inputs, outputs)
model.summary()
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test,  y_test, verbose=2)
```

```
Train on 50000 samples
Epoch 1/5
50000/50000 [==============================] - 34s 680us/sample - loss: 1.6324 - accuracy: 0.4205
Epoch 2/5
50000/50000 [==============================] - 33s 667us/sample - loss: 1.1720 - accuracy: 0.5896
Epoch 3/5
50000/50000 [==============================] - 33s 665us/sample - loss: 1.0026 - accuracy: 0.6489
Epoch 4/5
50000/50000 [==============================] - 33s 663us/sample - loss: 0.8923 - accuracy: 0.6879
Epoch 5/5
50000/50000 [==============================] - 33s 666us/sample - loss: 0.8083 - accuracy: 0.7175
10000/1 - 3s - loss: 0.9547 - accuracy: 0.6630
[0.9831310911178589, 0.663]
```

4.3 Resnet 초간단실습 (residual block + pulling)

```
inputs = keras.Input(shape=(32, 32, 3))
x = inputs
#x = layers.MaxPooling2D(2)(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(_x)
x = _x
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(_x)
x = x + _x
x = layers.MaxPooling2D(2)(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(_x)
x = x + _x
x = layers.MaxPooling2D(2)(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(x)
_x = layers.Conv2D(128, 3, activation='relu', padding="same")(_x)
x = x + _x
x = layers.MaxPooling2D(2)(x)

x = layers.Flatten()(x)
x = layers.Dense(128)(x)
x = layers.Dense(128)(x)
x = layers.Dense(10, activation='softmax')(x)
outputs = x

model = keras.Model(inputs, outputs)
model.summary()
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=100)
model.evaluate(x_test,  y_test, verbose=2)
```

```
Train on 50000 samples
Epoch 1/5
50000/50000 [==============================] - 22s 438us/sample - loss: 1.3663 - accuracy: 0.5045
Epoch 2/5
50000/50000 [==============================] - 21s 420us/sample - loss: 0.8867 - accuracy: 0.6915
Epoch 3/5
50000/50000 [==============================] - 21s 420us/sample - loss: 0.7260 - accuracy: 0.7456
Epoch 4/5
50000/50000 [==============================] - 21s 422us/sample - loss: 0.6218 - accuracy: 0.7824
Epoch 5/5
50000/50000 [==============================] - 21s 423us/sample - loss: 0.5350 - accuracy: 0.8134
10000/1 - 2s - loss: 0.6408 - accuracy: 0.7595
[0.7477127169132233, 0.7595]
```

끝!

코드 출처 :
https://colab.research.google.com/drive/1pxmIrnCbG7J3vFm2RWtrv2K-UfTT45IR#scrollTo=xdqyf3fa2XFz