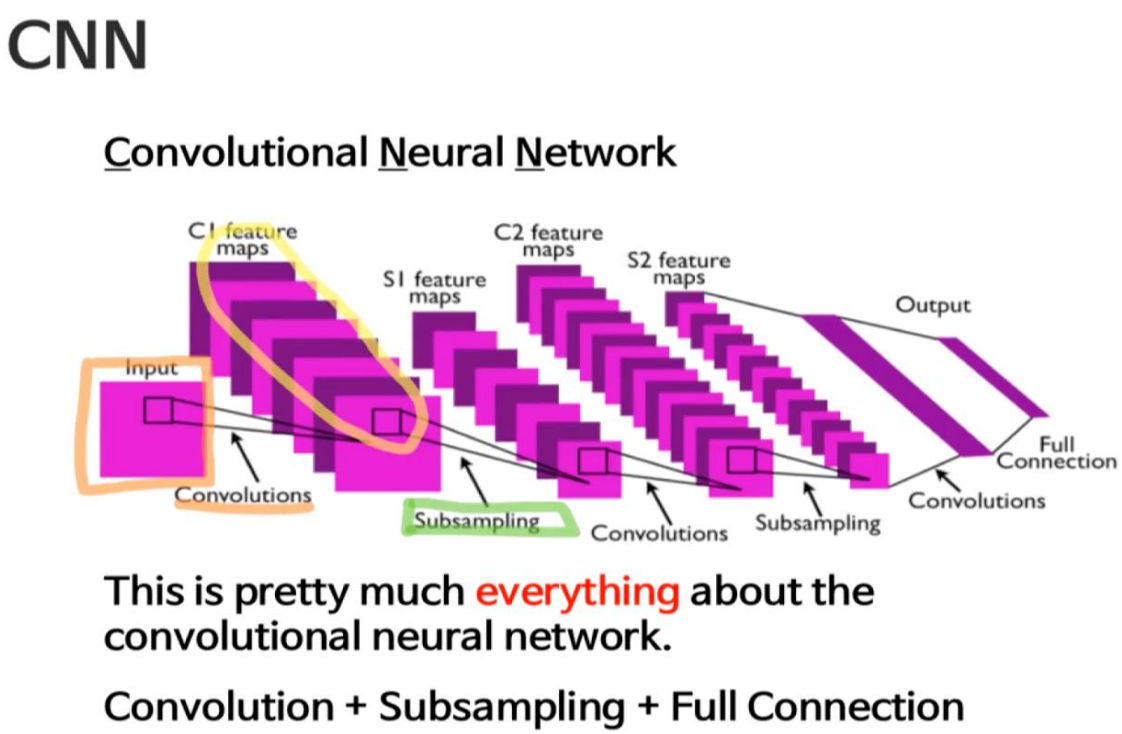
- Convolutional Neural Network
- Convolution 이 뭘까?



이미지에 보이는 것과 같이 convolution subsampling 이 반복되다 fully connected layer 가 적용되는 방식이다.

convolution 과 subsampling 레이어는 "Feature Extraction layers" 로 동작하게 된다.

feature extraction 은 말 그대로 특징 추출인데 이미지에서 중요한 부분을 뽑아내는 것이다.

만약 고양이 사진이 있다고 가정해보자.

고양이라는 이미지에서 중요하다고 생각되는 특징들 뾰족한 귀와 긴 꼬리와 같은 것이 feature 에 해당된다.

feature extraction 을 거치면 특징들이 나오게 될 것이다.

하지만 우리가 최종적으로 얻고 싶은 것은 특징이 아닌 이 특징들을 가진것이 어떤 물체인가를 구분하는 것이다.

특징들을 다 추출한 뒤에는 최종적으로 이미지에 나와있듯 fully connected layer 를 사용하여 이 물체가 어떠한 것인지 판단할 수 있게 된다.

(최근에 엿들은 바로는 요즘 딥러닝 트렌드는 특징 구분을 위한 fully connected layer 를 없애는 방향이라고 하는데... 잘 모르겠다 일단 내가 알고 있는 방법은 fully connected layer 를 사용하여 classificcation 을 하는 것이다.)

feature extraction

convolution

subsampling

- pooling

잘 되는 이유

- local invariance 국소적으로 차이가 없다는 것

Loosely speaking 동일한 convolution filtter 들이 "Sliding" 전체를 돌아다니면서 특징을 추출하게 되기 때문에 해당 추출된 특징의 위치는 문제가 되지 않는다.

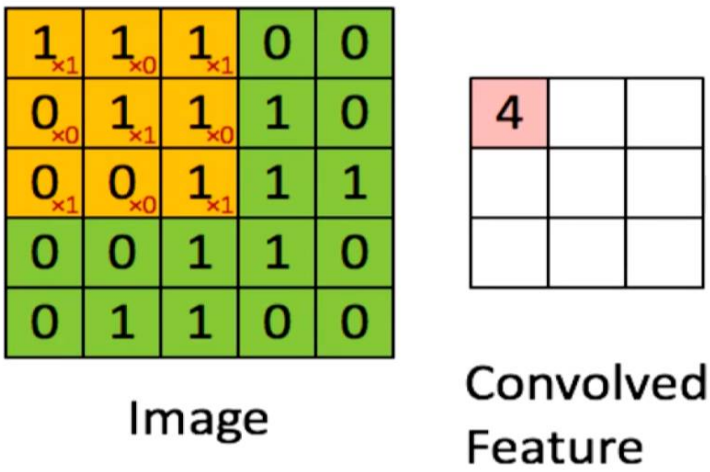
만약에 100x100 이미지가 있다고 가정해보자. train 을 시킨 이미지와는 다르게 고양이가 옆으로 살짝 이동했다고 가정하자. 만약 위치 정보가 물체의 판단에 있어 영향을 미치게 된다면 살짝 옆으로 이동한 이미지는 고양이라고 판단하지 못할 것이다.

- Compositionality

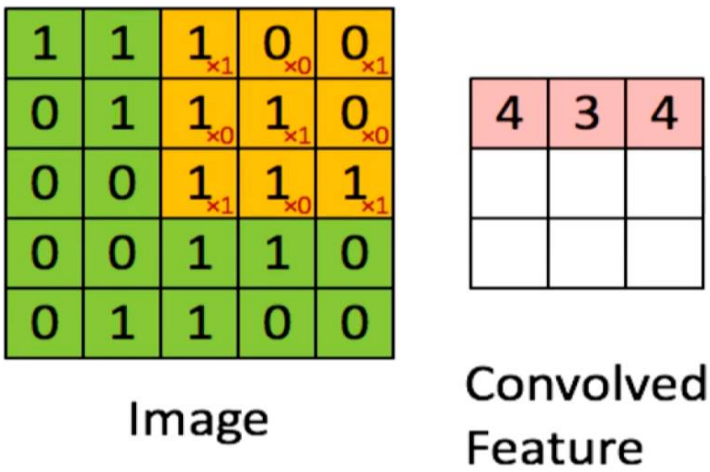
이미지가 주어지면 convolution 그리고 subsampling(pooling) 이 계속 반복되면서 계층 구조를 쌓게 되는데 이런게 일종의 Compositionality 라고 한다.

- Convolution 의 과정

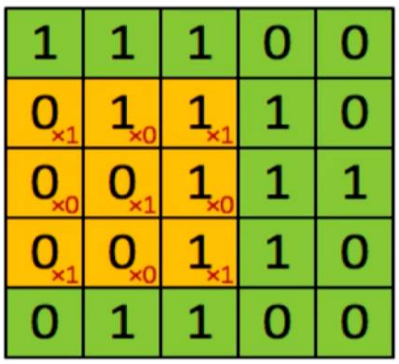
Convolution

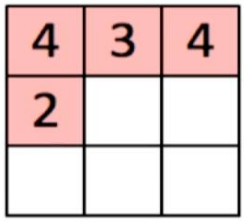


Convolution

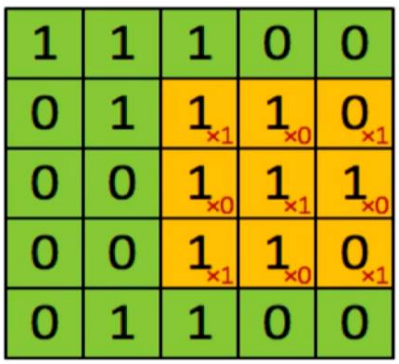


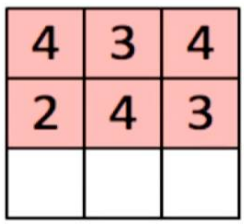
Convolution



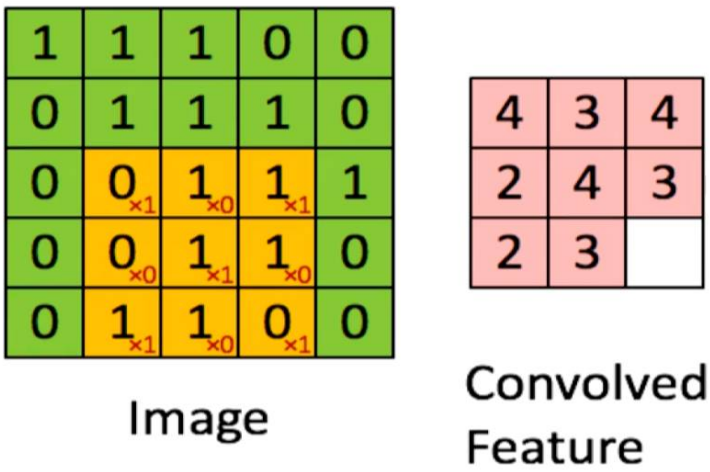


Convolution

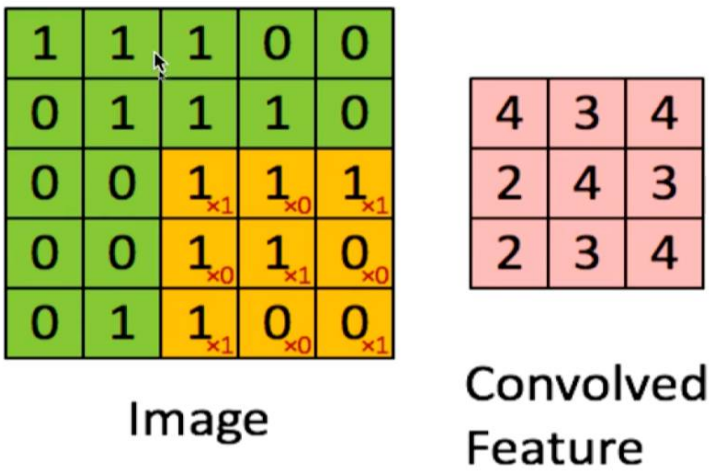




Convolution



Convolution



5x5 이미지가 있고 3x3 필터링을 거치는 것이다.

3x3 의 대각선에 1이 위치하는 것을 확인할 수 있다.

마지막 이미지를 보고 연산을 진행해보자. 1x1 + 1x1 + 0x1 + 1x1 + 1x1 = 4 일치하는 것을 확인할 수 있다.

이 연산을 보면서 우리는 유추할 수 있다.

이 결과는 지금 conv 하고 싶은 이 위치에 픽셀들이 내가 가지고 있는 convolution filter 이미지의 모양과 얼마나 유사한지 그 정도를 계산하여 feature 의 해당 부분에 넣게 된다.

convolution 연산이라는 것이 어떤 이미지 픽셀에 대해 (매직 아이 하고 보는 것처럼) 그냥 대충 본 (대각선의 픽셀 정보만 가진) 결과를 feature의 해당 픽셀에 대입한다. 비슷하면 response 의 값이 굉장히 높게 나온다.

우리가 학습시키는 것은 이 필터의 모양인데 주어진 데이터를 통해서 내가 어떤 convolution fileter 모양을 가지고 있었을 때 가장 좋은 성능이 나오는지 학습을 시키고 새로운 이미지가 주어졌을 때 convolution 을 하게 된다.

동일한 이미지가 전체를 돌아다니면서 convolution 을 한다는 것이 매우 중요하다.

이런 이해를 위해 stride zero-padding channel 에 대한 개념을 확실히 알아둘 필요가 있다.

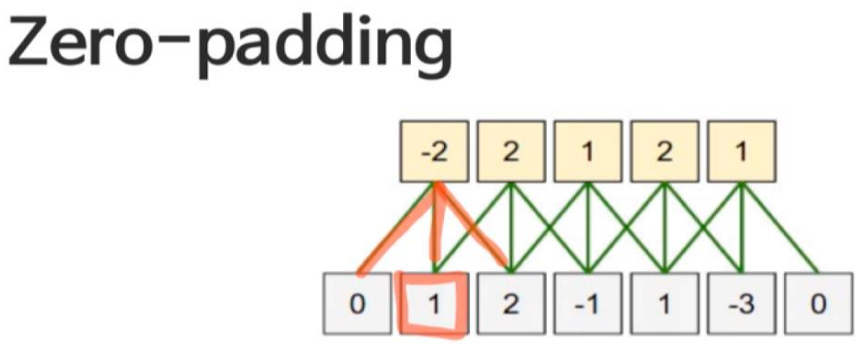
- zero-padding 이란

어떤 이미지가 있을 때 conv 을 가장자리에서도 할 수 있도록 하는 것이다. 그림과 같이 1x5 이미지가 있다고 가정해보자.

3짜리 convolution 을 가지고 convolution 을 진행한다고 생각해보자.

빨간 부분에 집중해 보면 3개로 convolution 을 진행해야 하는데 맨 가장자리의 경우 적용할 수 있는 zero padding 이 힘들다.

그래서 0을 넣는 것이다! 간단하쥬?



정리하면 (질문)

인풋의 size 는 몇일까요? 5

결과의 size는 몇일까요? 5

filter 의 size 는 몇일까요? 3

zero-padding 의 사이즈는 어떻게 될까요? 1

만약에 filter 가 5가 된다면 zero-padding 은 2 가 될 것이다.

이를 공식으로 정리해 보면

"n" out = ("n" in + "n" padding - "n" filter) + 1 이 된다.

• stride 란

영어로는 널리 뛰다 라는 뜻인데

직관적인 설명 5x5 가 있다고 하고 stride 를 2 라고 하자

convolution 을 하게 되면 5/2 = 2.5 가 되는데 2.5 를 가질 수는 없으니까 output 이 3이 되게 된다.

만약에 stride size 가 filter 의 사이즈와 동일하다면 거기에는 overlapping 되는 것이 없을 것이다.

코드로는 어떻게 적용되는지 확인해보자

Conv2D

```
tf.nn.conv2d(input, filter, strides, padding,
use_cudnn_on_gpu=None, name=None)
```

Computes a 2-D convolution given 4-D `input` and `filter` tensors.

Given an input tensor of shape [batch, in_height, in_width, in_chnnel] and a filter / kernel tensor of shape `[filter_height, filter_width, in_channels, out_channels]`, this op performs the following:

conv2d 안에는 input 그리고 filter stride padding use-cudnn-on-gpu=none, name=none 이렇게 되었다.

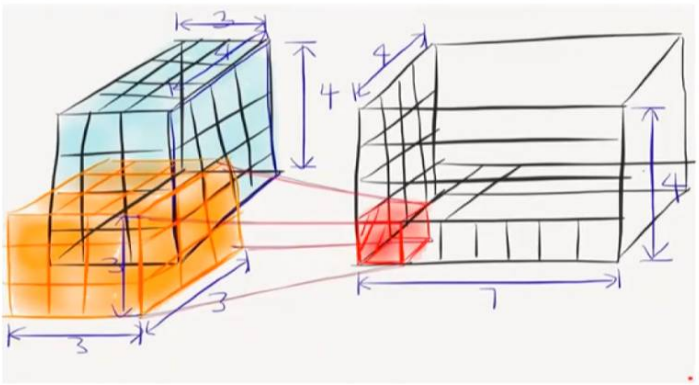
우리가 입력으로 주는 tensor 의 모양은 [batch, in_height, in_width, in_chnnel] 로 구성되어있고

Conv2D

```
tf.nn.conv2d(input, filter, strides, padding,
use_cudnn_on_gpu=None, name=None)
```

Computes a 2-D convolution given 4-D `input` and `filter` tensors.

Given an input tensor of shape [batch, in_height, in_width, in_chnnel] and a filter / kernel tensor of shape [filter_height, filter_width, in_channels, out_channels] :his op performs the following:



[batch, in_height=4, in_width=4, in_chnnel=3]

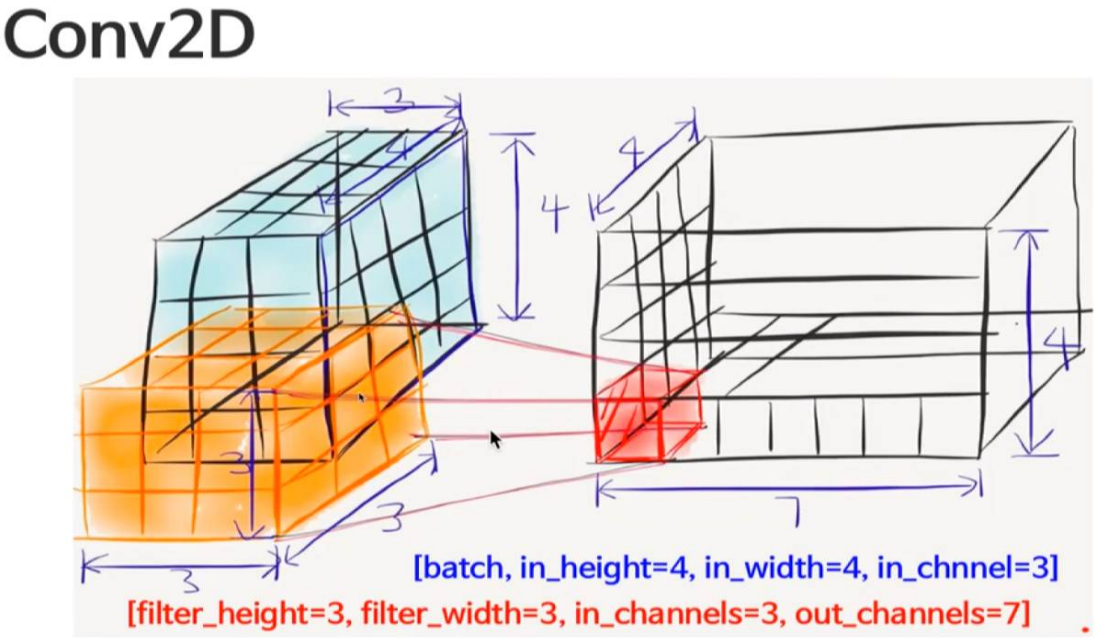
우리가 주의해야할 부분은 저기 input 파라미터에 존재하는 in_chennel 과 filter 에 존재하는 out chennel 이 동일해야 한다는 것이다.

stride 가 1이라면 width 가 4이니까 out put 의 width 도 동일하게 4가 될 것이다.

여기서 보면 filter 가 3x3x3 으로 이루어져 있다 우리는 총 27번의 연산을 수행하게 된다.

4x4를 conv 하였으니 똑같이 4x4 이다. 모든 이미지 픽셀마다 해당 연산을 수행해 주면 4x4가 되고 out_chennels 가 7개이므로 지금 가지고 있는 conv 개수라고 정의할 수 있다.

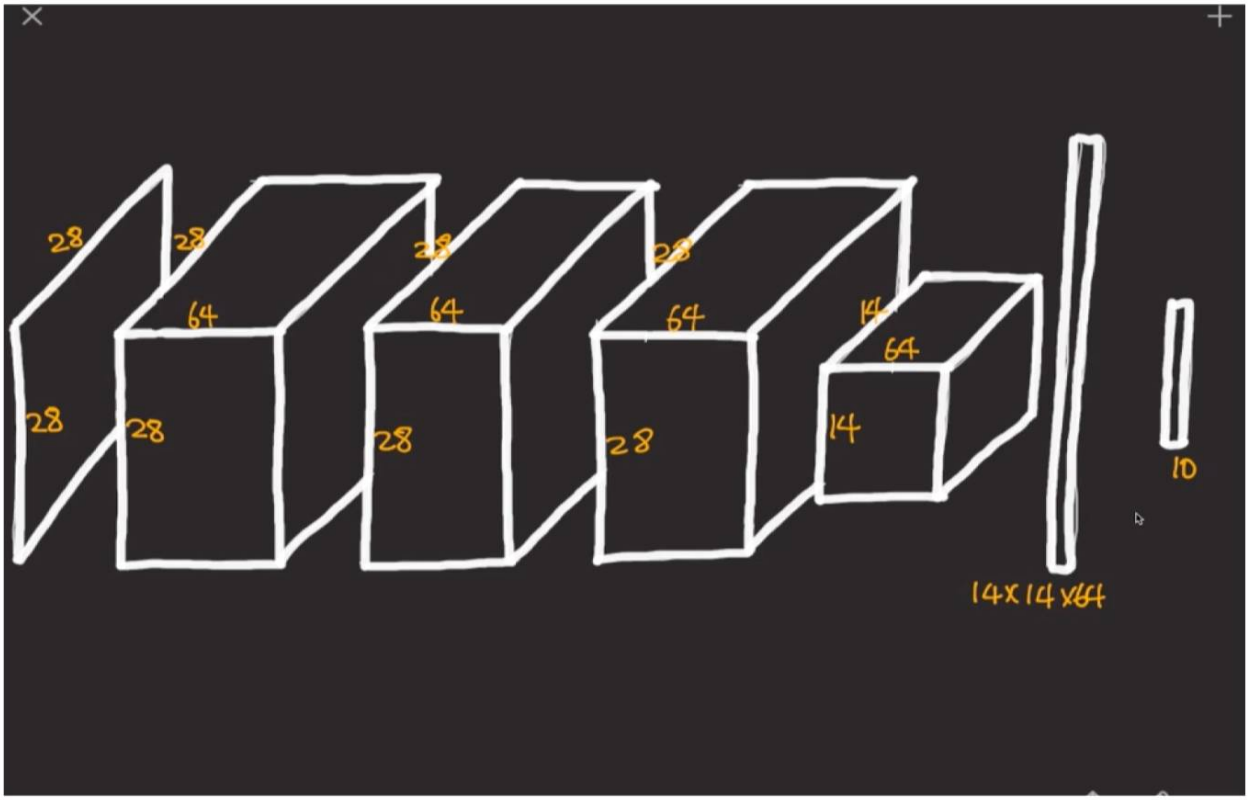
즉 서로 다른 모양의 필터를 7개 가지고 있어서..



number of parameters 의 수는 몇개일까요

3x3x3x7 개의 파라미터가 존재한다. 파라미터의 수는 굉장히 유의미 동일한 수의 데이터가 있다면 파라미터 수가 적으면 적을 수록 좋다.

그리고 겹 레이어를 많이 쌓이게 하는게 목표이다 그래서 우리는 다양한 스킬들을 사용하게 된다.



이 그림을 보면 28x28 을 진행하자

3x3 convolution 을 진행하자 그리고 fully connected 를 진행

여기서 64 는 64개의 필터가 존재한다는 것이다

우리는 여기다가 bias 를 더해준다

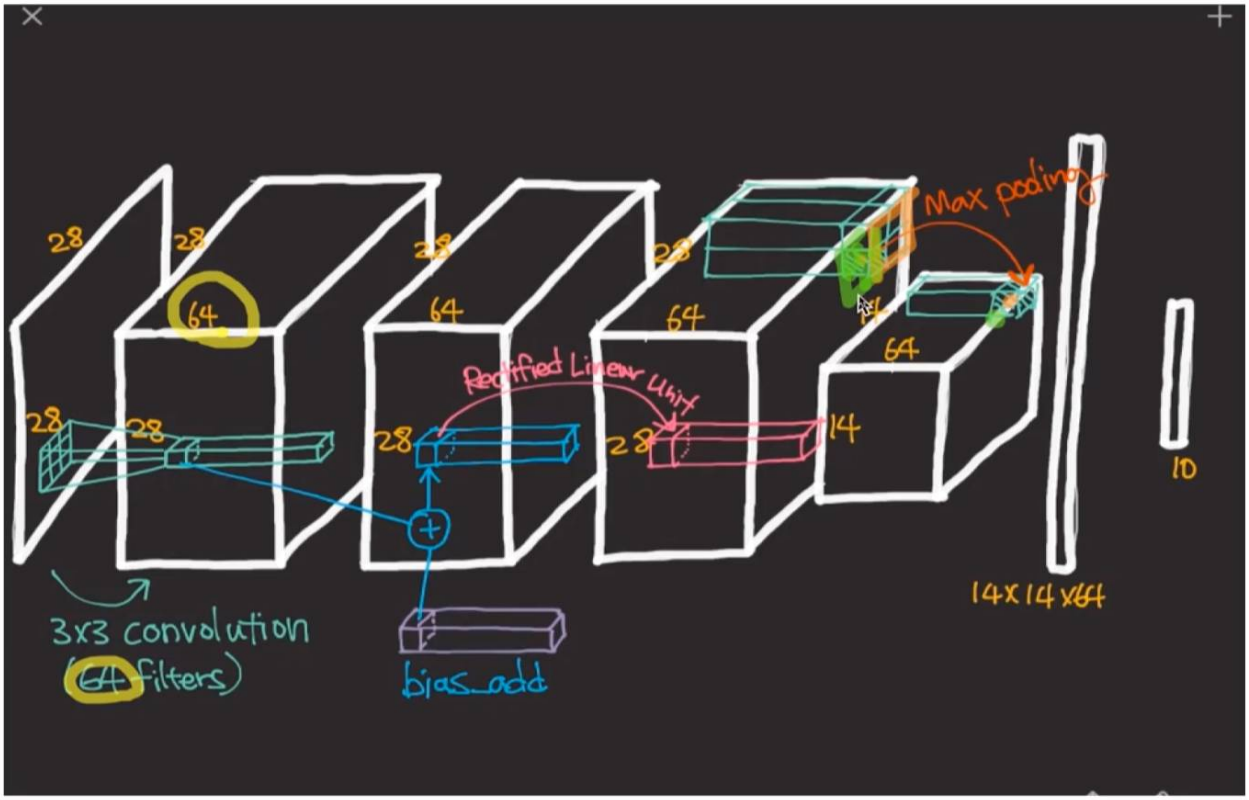
각각의 채널에다가 bias , 동일한 숫자를 한개씩 더해주게 된다. 그리고 activation function 을 진행한다. 후에 maxpooling 을 진행한다.

2x2 max polling 을 한다는 것은 4개의 숫자중에서 가장 큰거 하나를 집어넣고 overlaping 없이 옮겨서 젤 큰ㄴ수를 옆칸에 넣는식으로

전체에 대해서 연산을 취하면 맨 마지막에서 3번째 이 그림이 된다.

max pooling 은 최대값을 넣기 때문에 max pooling 이고

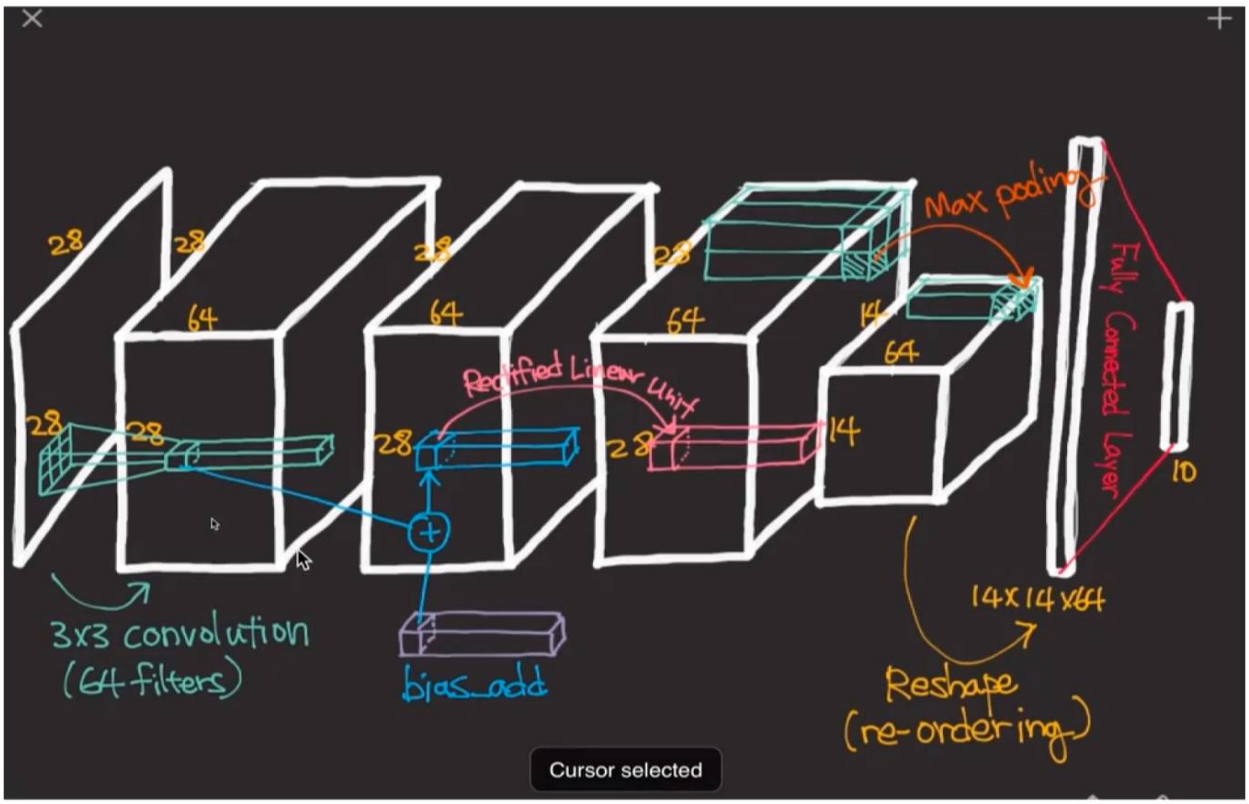
average pooling 도 존재한다.

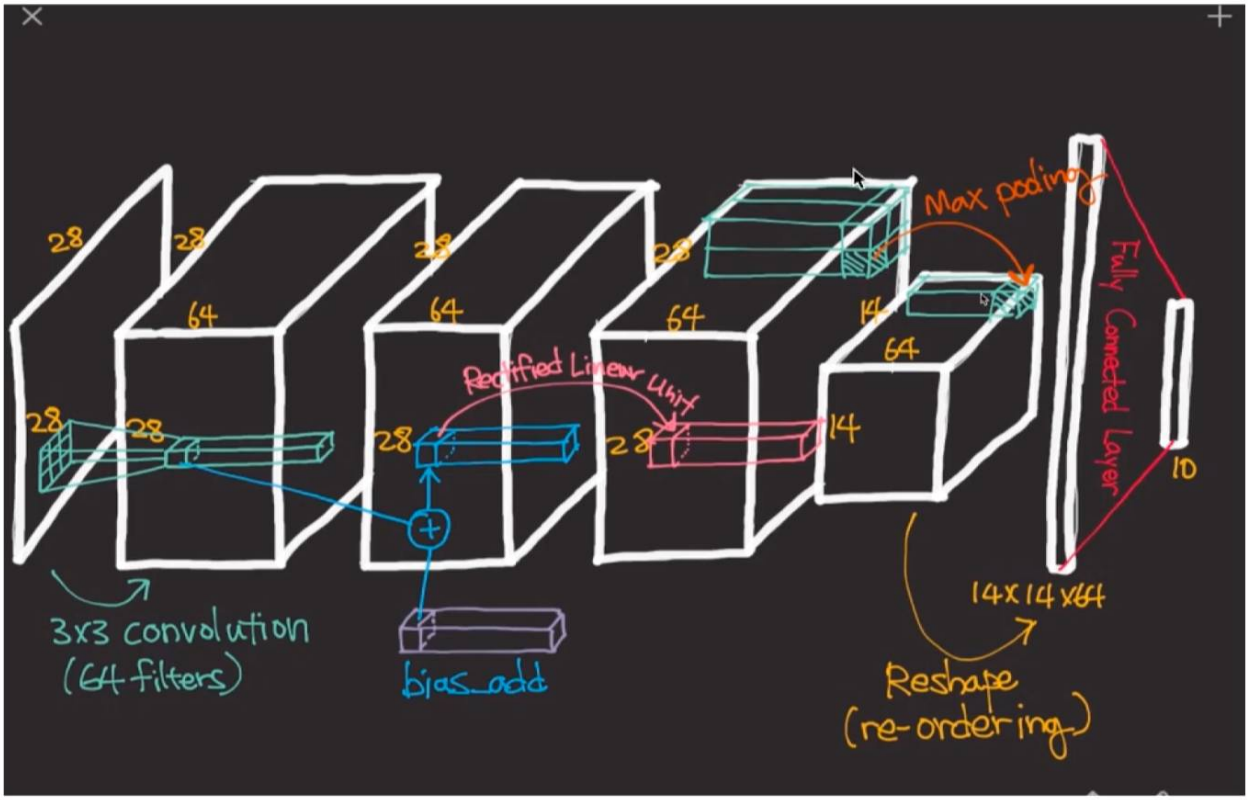


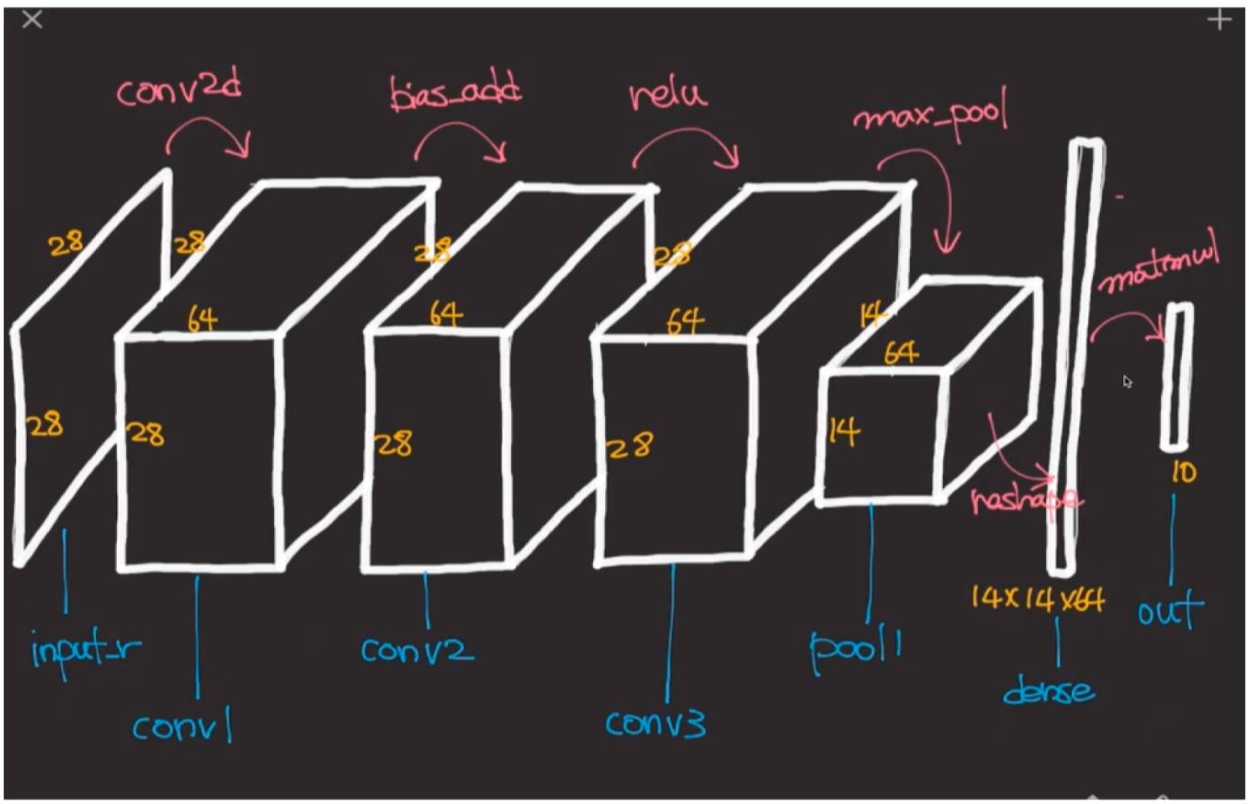
맨 마지막 단계는 14 x 14 x 64 를 한줄로 길게 펴준다.

후에 fully connected layer 를 거쳐 10개의 숫자로 바꾸어주게 된다.

이때는 행렬을 하나 만든다. 14x14x64x10 을 통해 10차원 벡터를 통해 마지막 단계로 바꾸어준다.







convolution 을 정의하기 위한 파라미터 수

3x3 짜리 필터가 64개 있는거에 더하기 64가

convolution layer 를 정의하기 위한 파라미터 수

fully connected 를 정의하기 위한 파라미터 수는 14x14x64x10 +10 이 된다. 바이어스가 존재하기 때문에

컨볼루션 레이어를 정의하는 것 보다 fully connected 가 훨씬 많다

따라서 최근 트렌드는 이러한 fullyconnected 레이어를 없애는 fully convolution network 구조를 사용하거나 뒷단의 fully connected 를 간소화 시키는 방법을 사용한다.

이렇게 파라미터를 계산해보는 것이 좀 습관화 되어야겠다는 생각

그래야 내가 네트워크를 구성할 때도 어떻게 하면 파라미터의 수를 줄이고 효과적으로 구성할건지에 대해 고민해 볼 수 있기 때문.