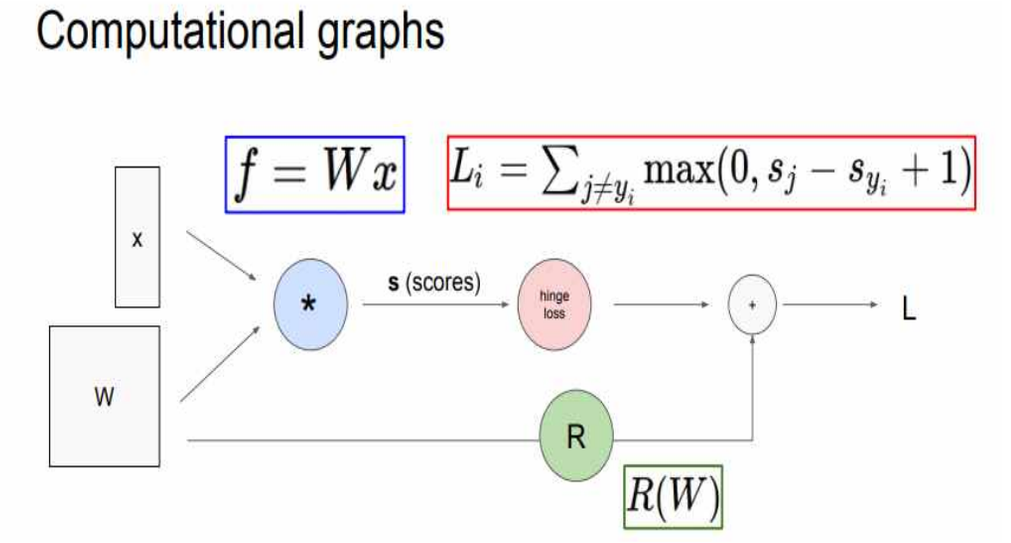
Lecture 4: Backpropagation and Neural Networks

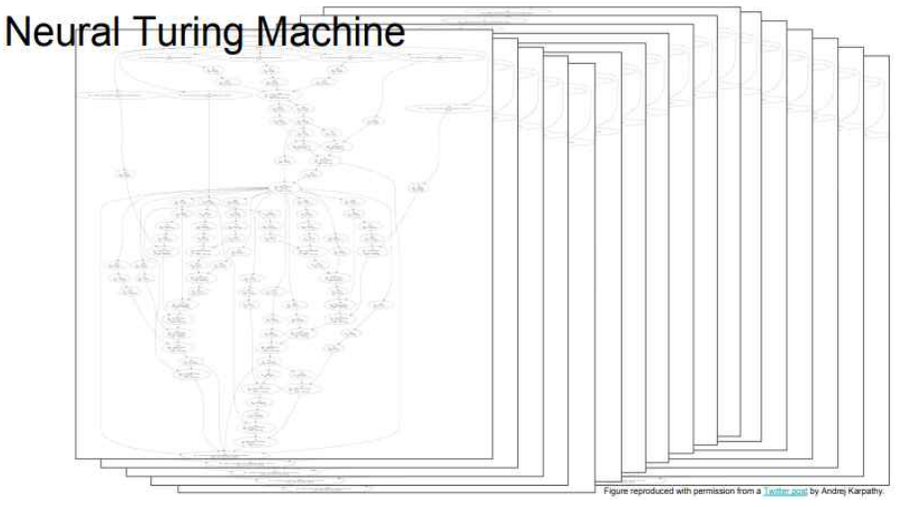


● Computational graphs
: 계산과정을 그래프로 나타낸 것. 노드와 엣지로 표현.
노드= 연산, 엣지= 데이터가 흘러가는 방향

● computational graph를 사용해서 함수를 표현 => backpropagation 가능

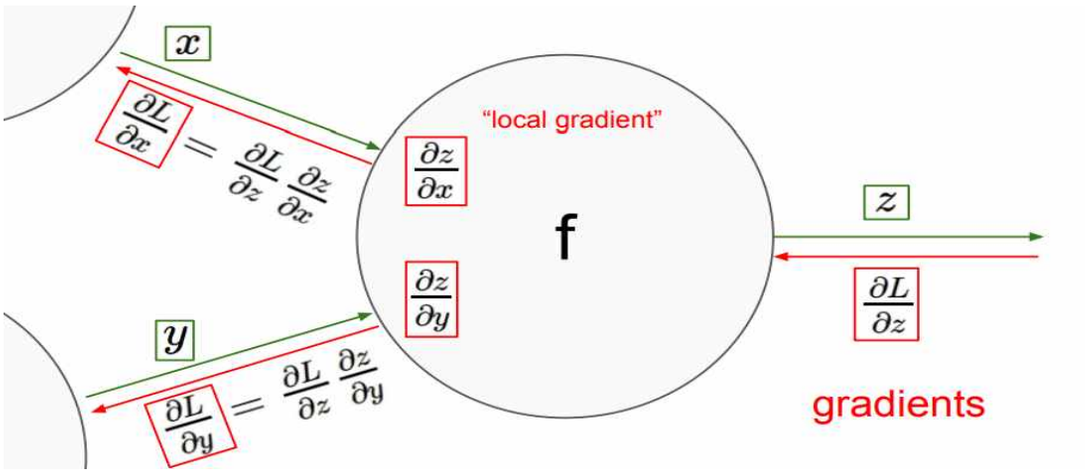
● 위 식은 단순해서 괜찮지만 하나하나 다 계산하는 것은 추천하지 X

ex)



=> 딥러닝의 다른 종류인 neural turing machine의 computational graph. 몹시 복잡하다.

● backpropagation: gradient 계산 위해 computational graph 내부의 모든 변수에 대해 chain rule을 재귀적으로 사용

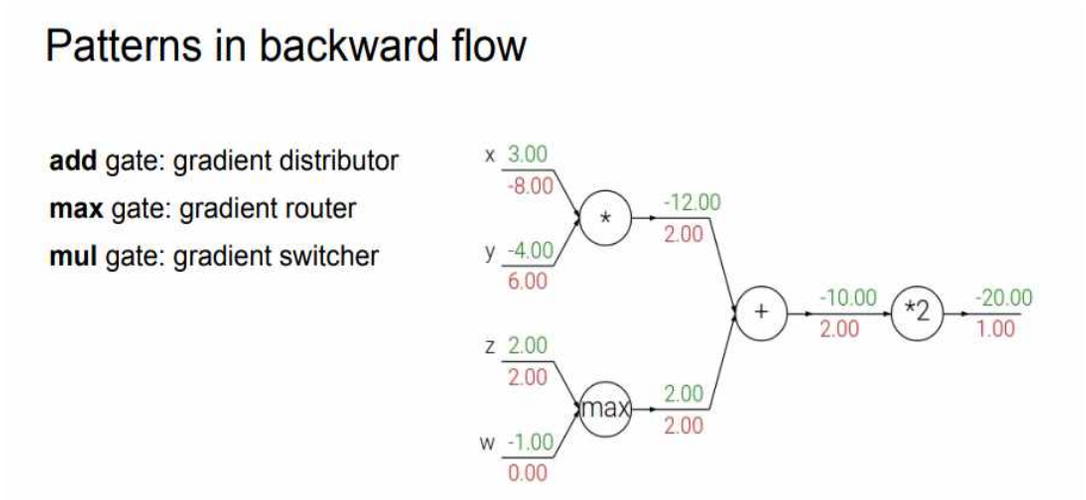


● 각 노드의 local input, output => local gradient 계산 가능
(= 들어오는 입력에 대한 출력의 기울기)

● z에 대한 L (최종 loss)의 gradient를 이미 구해놓음 (=gradients)

● gradients와 local gradient를 곱해 직전노드에 대한 gradient 계산 가능

● Backpropagation 중 gate에 대한 gradient의 특성

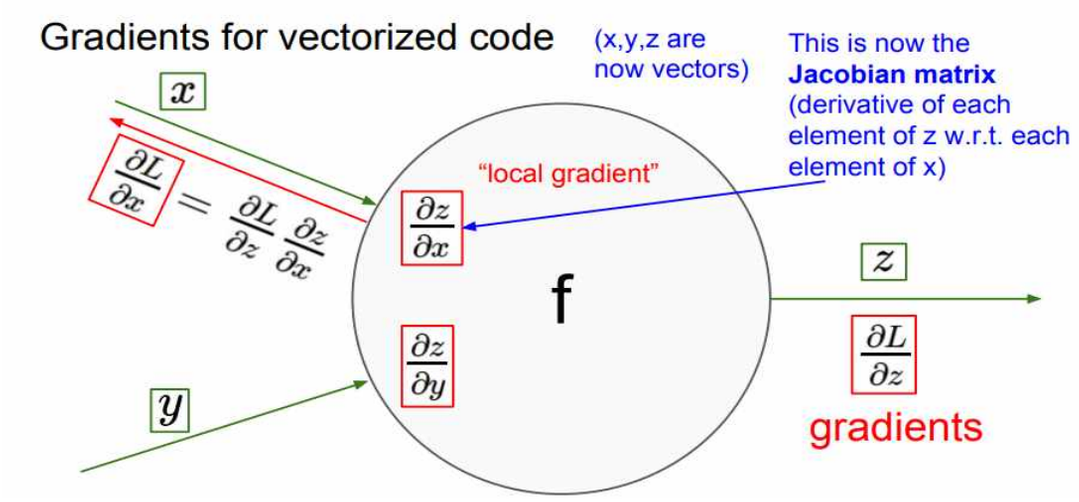


1) 덧셈게이트 - gradient 나눠줌. 앞 노드의 gradient를 똑같이 여러 개에 전달

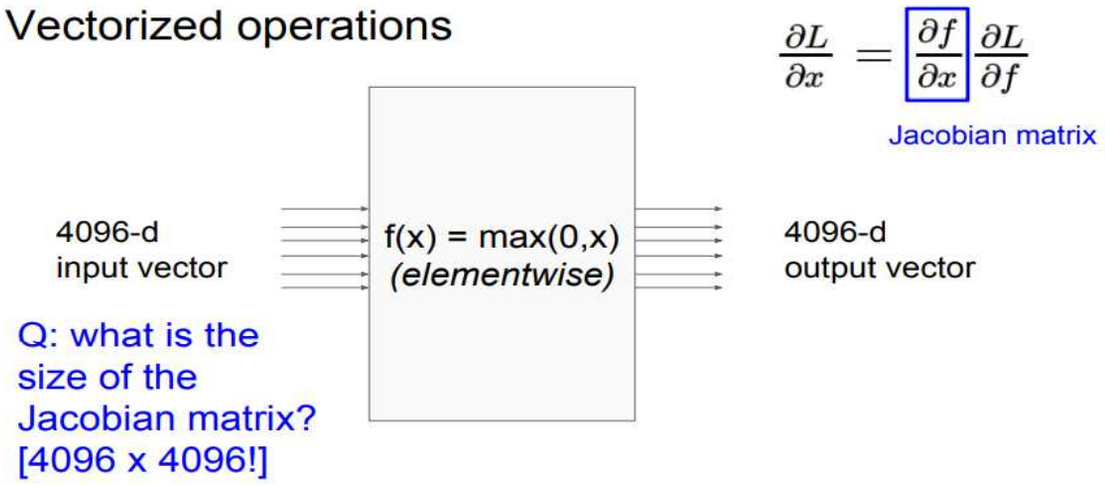
2) 맥스게이트 - 더 큰 입력값을 가진 노드에게 gradient 그대로 전달,
더 작은 입력값 가진 노드에게는 0 전달

3) 곱셈게이트 - ex) X*Y의 경우 X에 대한 gradient는 Y*upstream gradient
Y에 대한 gradient는 X*upstream gradient가 되기 때문에
곰셈게이트를 gradient switcher 라고 함

● 지금까지는 X,Y (input)가 스칼라일 경우의 gradient 살펴보았고,
이제부터는 input이 벡터인 경우 (다변수인 경우)



=> gradient가 자코비안 행렬이라는 차이점
= 다변수 벡터 함수의 도함수행렬



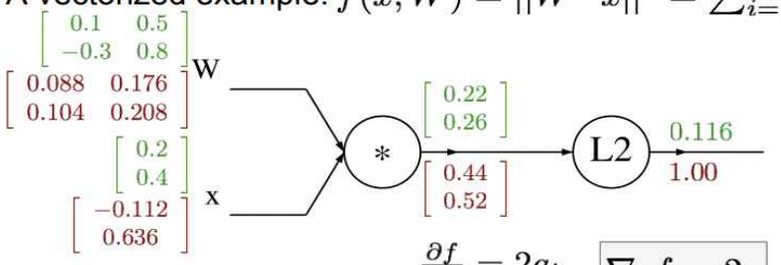
ex) 4096차원의 벡터 입력 => 요소별로 0과 비교해 최대값을 취하는 노드 => 4096차원의 출력
=> 이때 자코비안 행렬의 사이즈 = 4096 * 4096
(자코비안 행렬의 각 행 = 입력에 대한 출력의 편미분)

=> 만약 100개의 입력을 동시에 받는 배치 사용 => 기존 사이즈*100 => size = 409600 * 409600
=> 그러나 실제로는 이렇게 큰 자코비안 행렬을 계산하지 X

=> 여기선 어떤 구조를 볼 수 있는가?
: 요소별로 보기 때문에 입력의 각 요소(첫 번째 차원)는 오직 출력의 해당요소에만 영향
=> 자코비안 행렬은 대각행렬이 됨
=> 전체 자코비안 행렬을 작성하고 공식화할 필요 X

=> 출력에 대한 X의 영향, 이 값을 사용하는 것에 대해서만 알면 된다.

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$\frac{\partial f}{\partial q_i} = 2q_i \qquad \boxed{\nabla_q f = 2q}$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} = \sum_k (2q_k)(\mathbf{1}_{k=i} x_j) = 2q_i x_j$$

$$\boxed{\nabla_W f = 2q \cdot x^T}$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

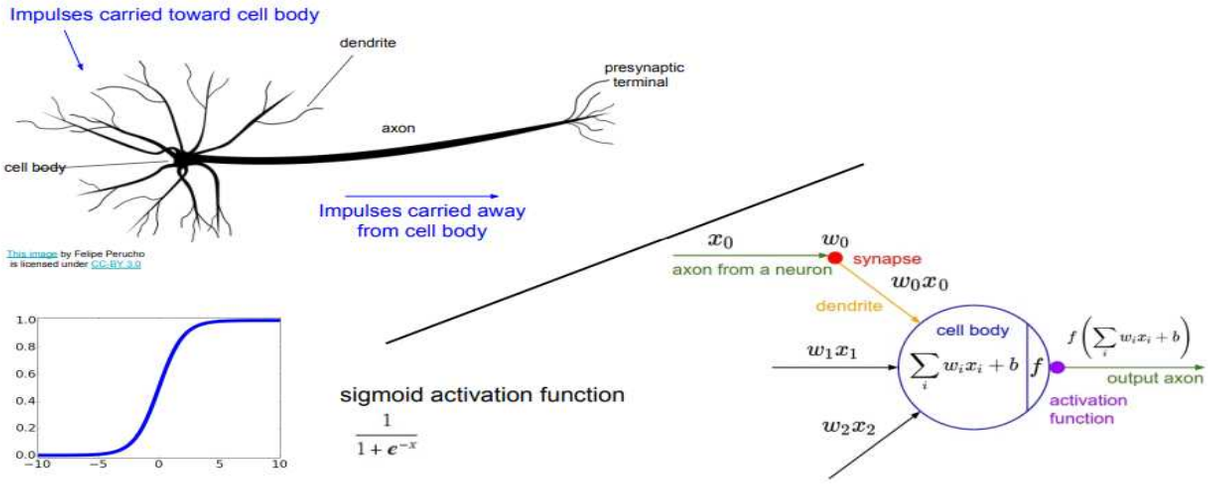
$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \sum_k 2q_k W_{k,i}$$

$$\boxed{\nabla_x f = 2W^T \cdot q}$$

주의: 변수에 대한 gradient가 변수의 모양과 같은지 항상 체크.
gradient의 요소는 변수의 각 요소가 최종출력에 얼마나 영향을 미치는지 정량화

● Neural Network



뉴런 - 자극이 들어오면 axon 거쳐서 다른 뉴런으로 들어감
neural network- input 들어오면 그것에 대한 w 곱하고 어떤 활성화함수 통해 거쳐 output으로 나감

● 활성화함수

1) tanh (Hyperbolic Tangent)

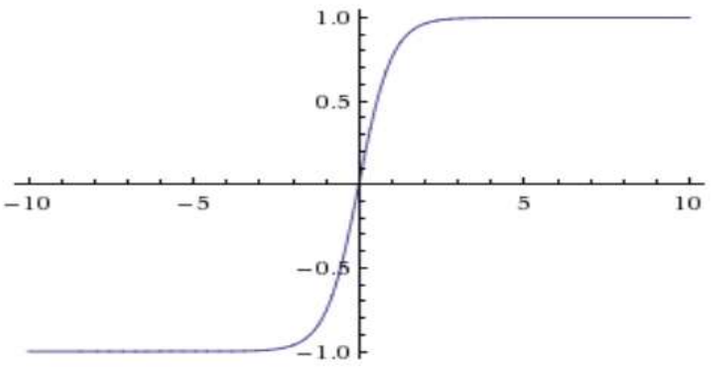
: sigmoid function 보완하기 위해 나온 함수 (Gradient Vainishing 문제)

: 입력신호를 (-1,1) 사이의 값으로 normalizing

: sigmoid보단 덜하지만 gradient vanishing 문제 여전히 발생

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

$$\frac{d}{dx}\tanh(x) = 1 - \tanh(x)^2$$



Tanh Function

2) Maxout

: Dropout의 효과를 극대화시키기 위해 고안한 활성함수. 성능이 좋은 편.

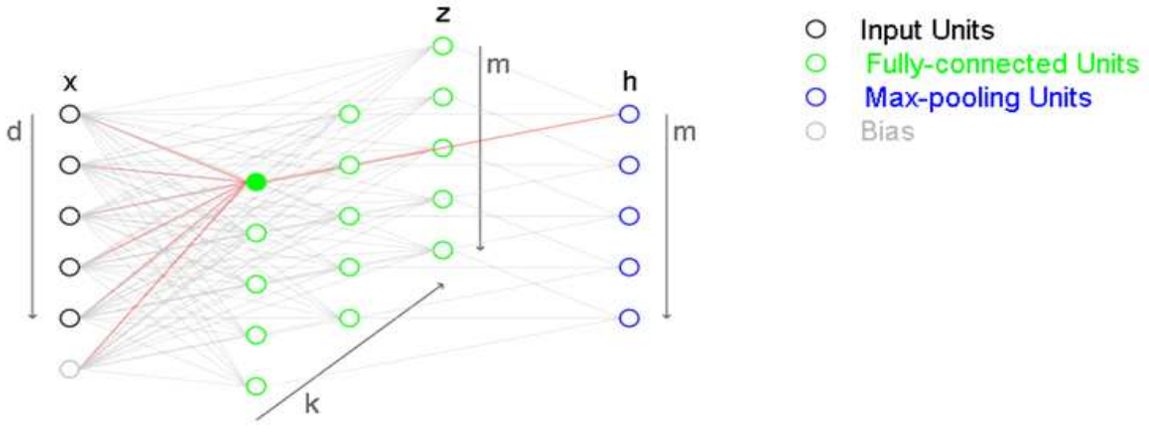
: ReLU의 장점 다 가지고 있으면서 Dying ReLU 현상을 완전히 회복

= 음의 값을 가질 경우 0을 출력해 weight가 업데이트 X

$$h_i(x) = \max_{j \in [1,k]} z_{ij}$$

$$z_{ij} = x^T W_{\cdots ij} + b_{ij}$$

함수 h = hidden layer. 입력값 중 최대값을 취함. W와 b는 학습 통해 결정되는 파라미터



: 일반적으로 hidden layer가 1개의 layer로 구성되는 것과 달리 Maxout hidden layer는 2개의 layer로 구성.

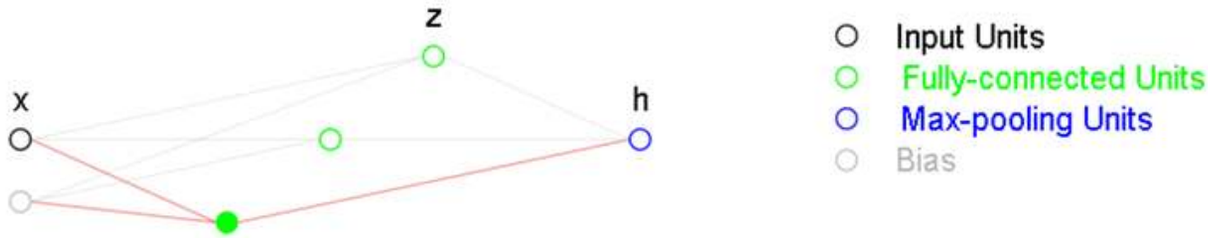
=> 녹색영역은 전통적인 활성함수가 아닌 affine function

= (단순하게 입력 x를 각각의 weight에 곱해서 더하는 형식)

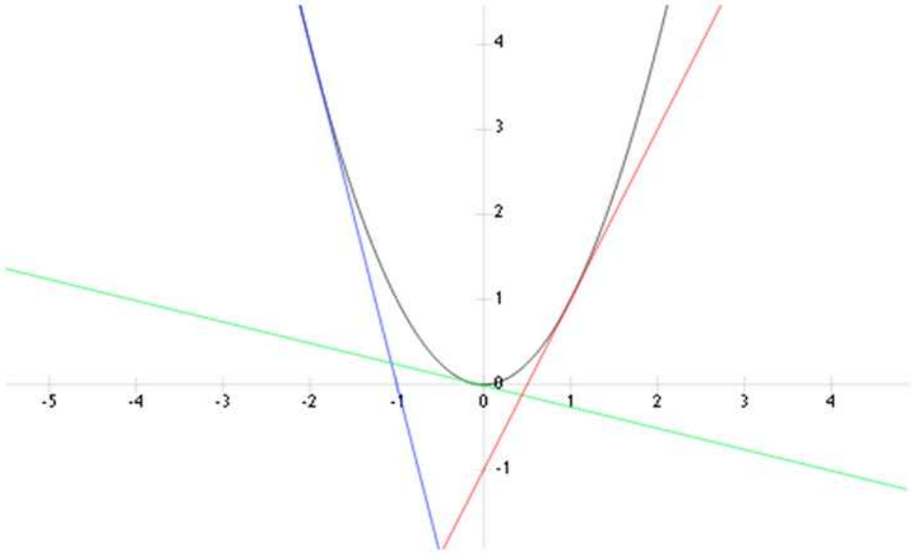
=> 위 그림 보면 k개의 column이 있는데 동일 위치에 있는 것 중 가장 큰 값을 파란색 영역에서 취해 최종적으로 m개의 결과가 나오게 된다.

● maxout의 의미

ex) 입력=2, 출력=1, k=3인 maxout unit



=> 이 3개의 유닛 이용해서 $f(x) = x^2$를 근사시킨다고 하면 아래와 같은 형태



=> f(x)를 3개의 직선으로 근사 시킬 경우 나오는 모양.

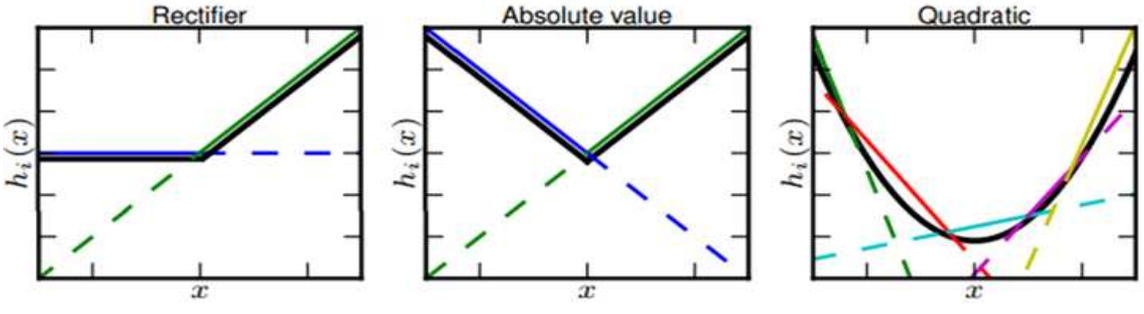
=> k값이 클수록 구간이 더 세분화 되면서 원래 곡선과 비슷한 형태를 갖게 된다.

=> 여기서는 직선으로 근사해서 오차가 커보이지만 affine 함수가 갖는 다양한 표현력 고려하면 k값이 크지 않아도 convex한 함수를 거의 표현할 수 있다.

=> 그런 의미에서 Maxout은 universal approximator라고 볼 수 있다.

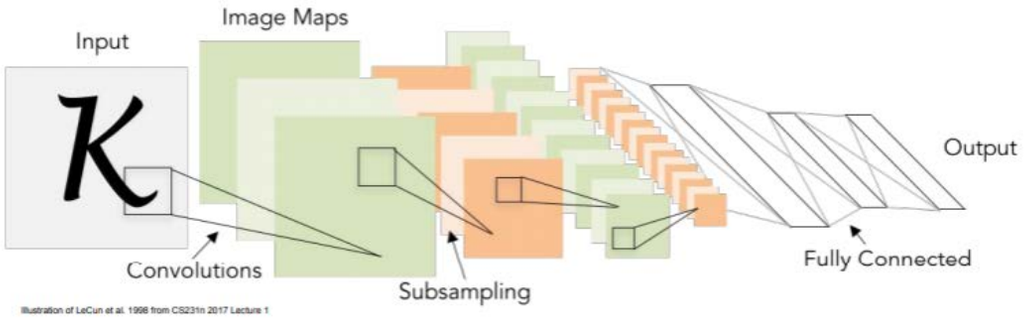
=> Maxout은 affine 함수 부분과 최댓값 선택하는 부분 이용해서 임의의 convex한 함수를 piecewise linear approximation하는 것이라 할 수 있다.

ex)



Lecture 5. CNN

Convolutional Neural Networks (CNN)

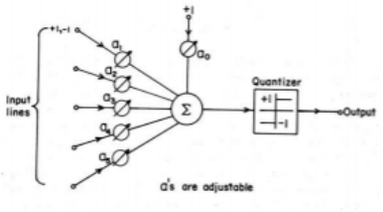


Neural Network의 역사

- Frank Rosenblatt 1957: Mark I Perceptron
  - 첫 perceptron algorithm
  - 이때 사용한 Machine은 사진기에 연결되어 400 픽셀 이미지를 출력.
  - 스코어 함수 wx+b와 비슷하고 → 출력 값은 0 또는 1

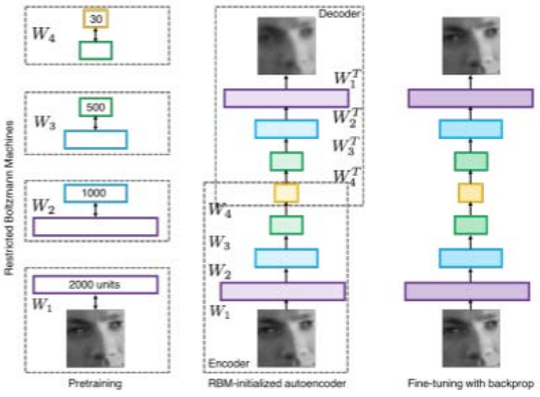
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

  - 여기서 w를 결정해주는 update rule은 back propagation과 비슷함
- Widrow and Hoff, ~1960: Adaline/Madaline
  - Linear layer perceptron network를 multi-layer perceptron network로 쌓기 시작함
  - 이때도 여전히 backpropagation을 통해 학습하여 weight을 구하는 방법은 사용하지 않았음



- Rumelhart et al. 1986:
  - Chain rule을 이용한back propagation이 사용되기 시작함

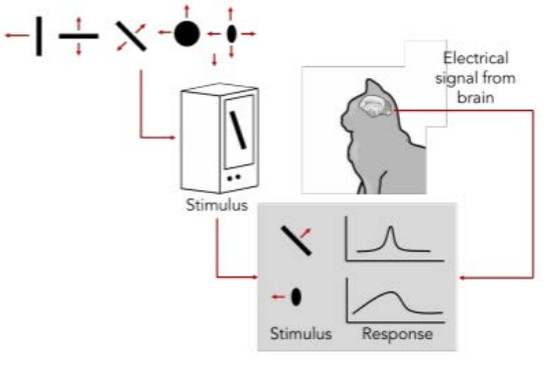
- Hinton and Salakhutdinov 2006:
  - Deep Neural Network를 효과적으로 학습할 수 있다는 것을 보여줌
  - 하지만 back propagation을 사용하려면 initialization이 까다로웠음.
  - Pre-training stage
    - 각 은닉 층을 Restricted Botlzmann Machines을 통해 모델링하고, 각 layer를 iteration을 통해 학습한 뒤 은닉 층을 이용해 full neural network를 초기화해서 back propagation을 수행함.

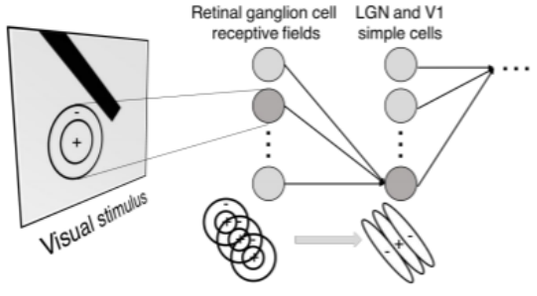


- First strong results: Neural Network 2010, 2012
  - Neural Network를 이용하여 음성인식 (2010)과 이미지 분류 (2012) 수행
  - Error을 많이 줄일 수 있었음
  - Classification의 benchmark
  - 이때 이후 CNN에 널리 사용됐다.

CNN 개요

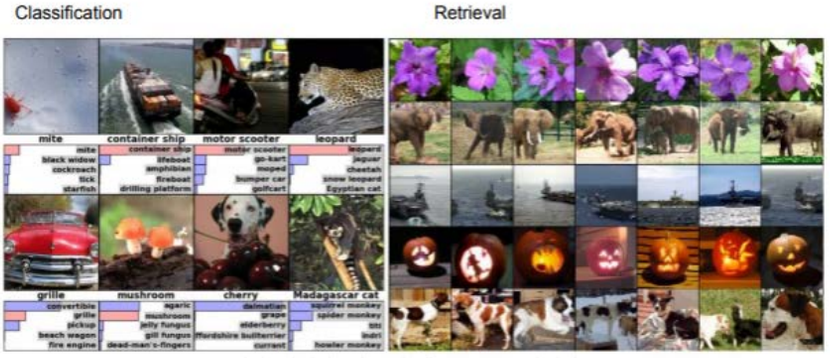
- CNN의 idea는 Hubel & Wiesel의 실험 아이디어로부터 나옴
  - 고양이에 다양한 모양의 시각적 자극을 준 뒤, neuron이 어떻게 반응했는지 측정함
  - Neuron이 위계적 구조를 가진다는 것을 발견 (세포가 더 깊은 층으로 갈수록 더 복잡한 구조에 반응)
    - Retinal ganglion cell: 동그란 자국에 반응
    - Simple cells: 명암과 반응
    - Complex cells: 명암과 동작에 반응



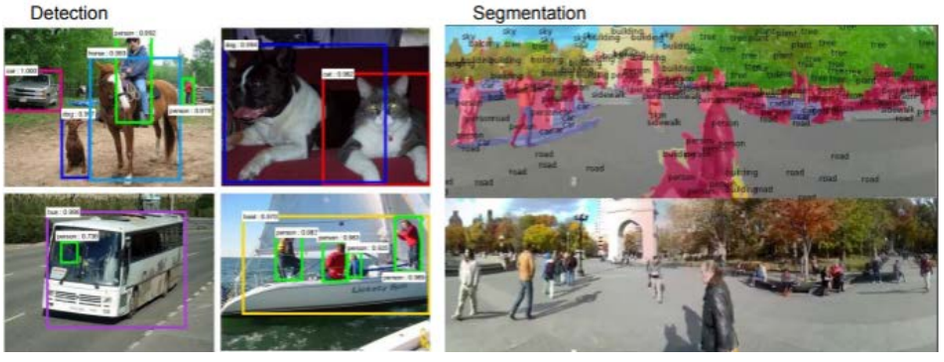


- Hypercomplex cells: 더 복잡한 구조에 반응 (모서리와, 색칠된 부분 구분)

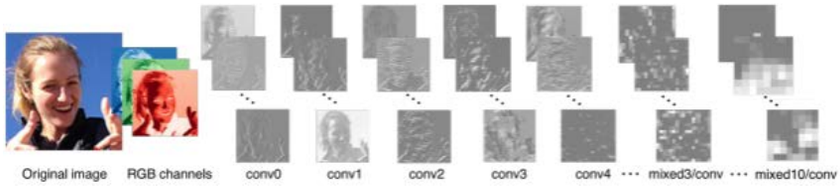
- CNN은 어디에서나 사용된다.
  - Classification
  - Image retrieval (사진 복구)



  - Detection (탐지)
    - 사진 속 사물을 파악하고 bounding box (경계 박스)를 그림
  - Segmentation
    - Bounding box 보다 정교하게 사물의 윤곽까지 pixel 단위로 파악할 수 있다.

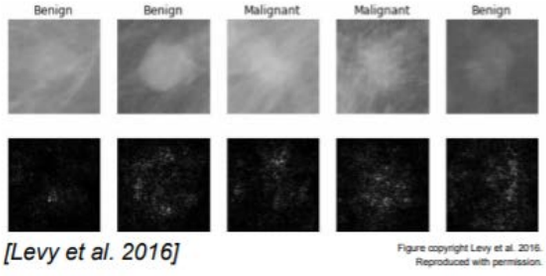


  - 자율주행자동차
    - Parallel processing을 하여 CNN을 학습하는 GPU를 사용
  - 얼굴 인식
    - 얼굴을 input으로 넣고 output으로 누구인지 파악



- 영상 분류
- 포즈 인식

- 의료 사진 분석 및 진단

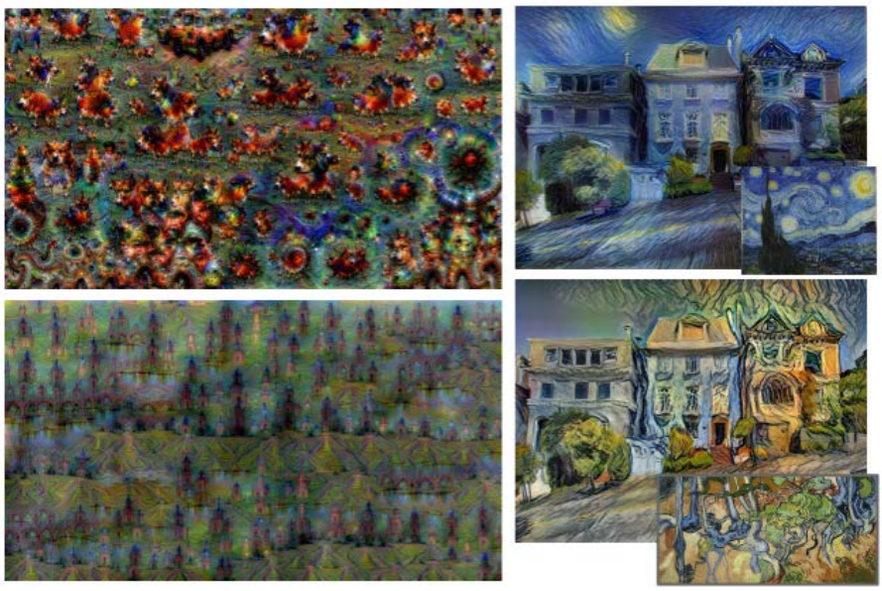


[Levy et al. 2016]

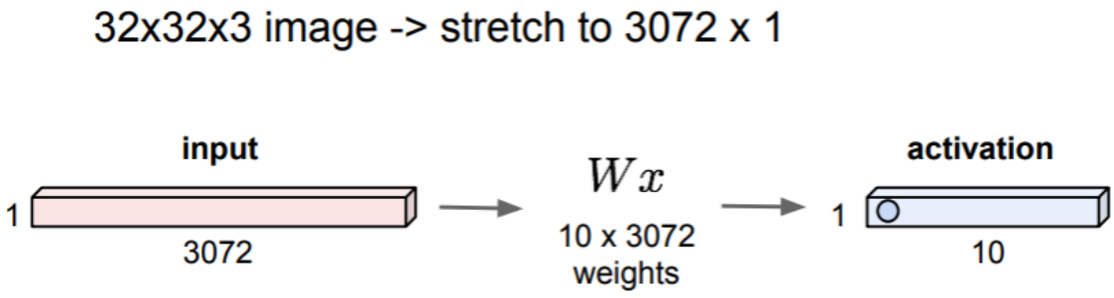
Figure copyright Levy et al. 2016.
Reproduced with permission.

- 은하계 분류
- 전광판 분류
- 고래 인식
- 지도에서 도로 분류
- Image captioning: 이미지를 input, 이미지를 표현하는 문장을 출력

- 예술
  - 환각 이미지
  - Neural style: 이미지를 input, output을 반 고흐 스타일 이미지로 출력



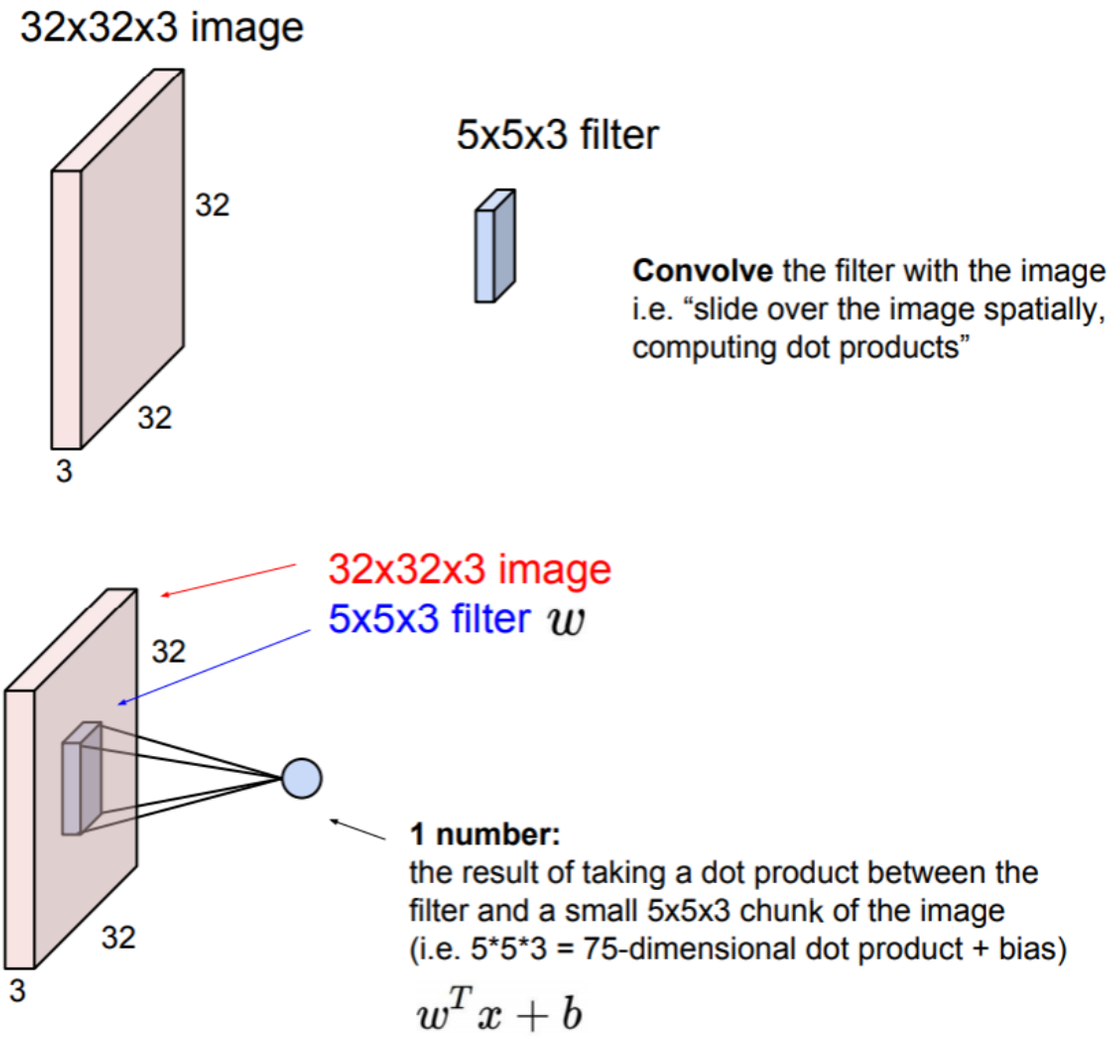
Fully Connected Layer



- 32x32x3 이미지 → 모든 픽셀을 3072x1 벡터로 늘려서 input으로 넣는다
- Input x를 weight 행렬 W로 곱하고 (벡터 내적 이용) output을 activation이라고 한다.

Convolution Layer

- **공간 구조를 유지**한다는 점에서 Fully Connected Layer와 다르다
- 32x32x3 이미지 형태를 그대로 유지하고, weight로 작은 필터를 사용하여 이미지를 훑어가며 내적을 수행한다.



- ex) 5x5x3 filter의 각 요소와 대응되는 32x32x3 이미지의 각 요소를 곱해서 더해준다.
- 필터와 그에 대응하는 픽셀 판에 element-wise multiplication 수행
- 이는 필터와 해당하는 이미지 픽셀 판을 벡터로 만들어 내적한 것과 같다.
- Filter의 깊이는 항상 input의 깊이와 같아야 한다
- 두 signal (여기서는 필터와 이미지) 간의 convolution은 수학적으로 다음과 같이 표현

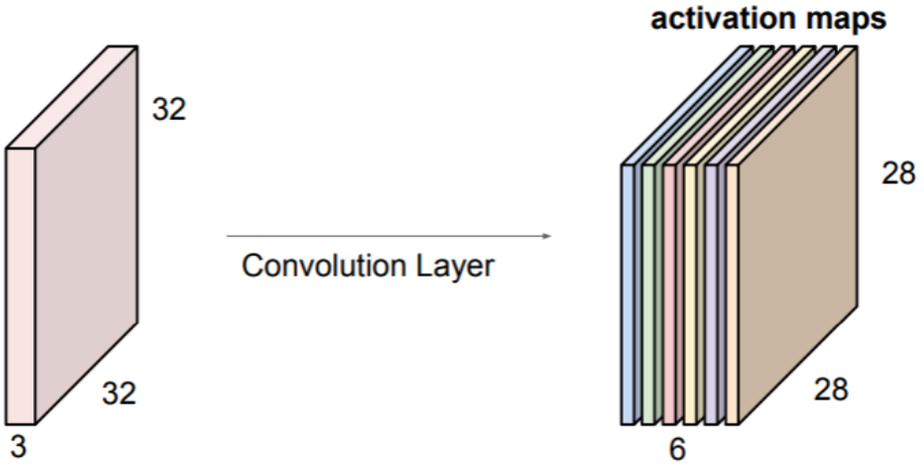
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

- 어떻게 훑는가?
  - 왼쪽 위에서 시작 → 각 위치마다 dot product 수행하여 하나의 값 →

이미지 전체를 훑어가며 (convolve/slide over) convolution 수행

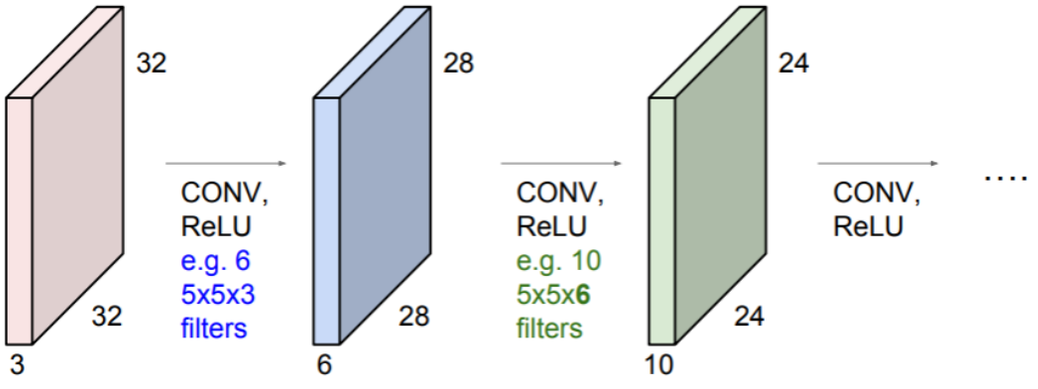
- Value of that filter at every spatial location

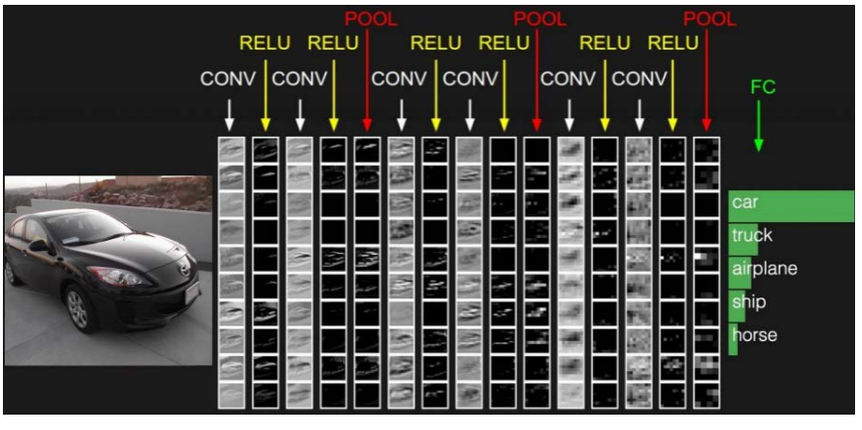
- Activation map
  - Convolution을 전체 input에 수행을 해서 나온 결과를 activation map이라고 함.



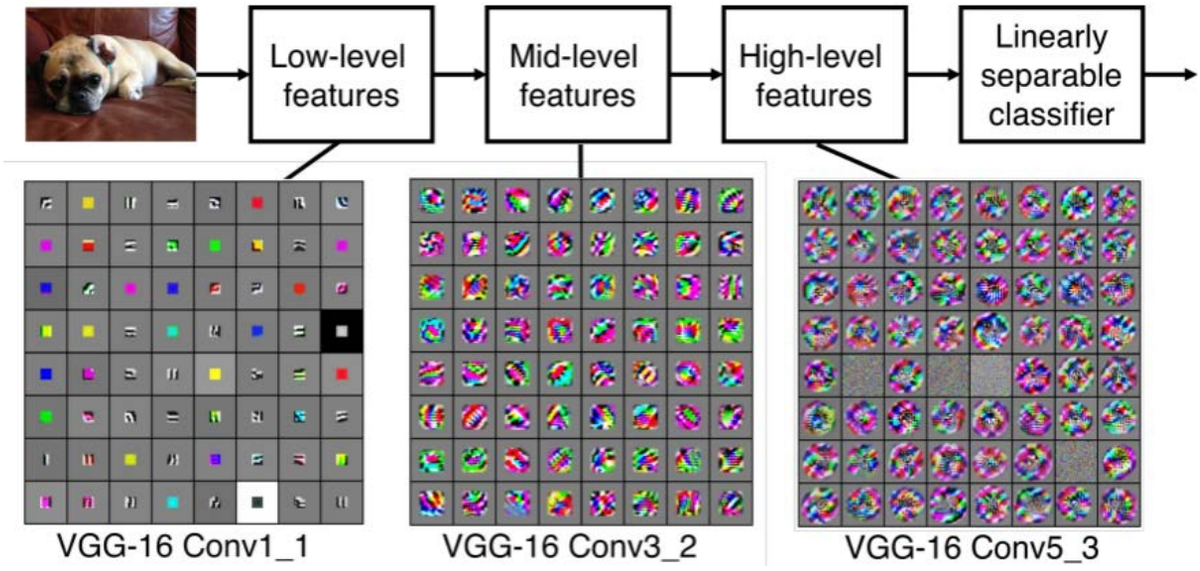
Convolutional Network (ConvNet)

- Convolution Network는 convolutional layer sequence

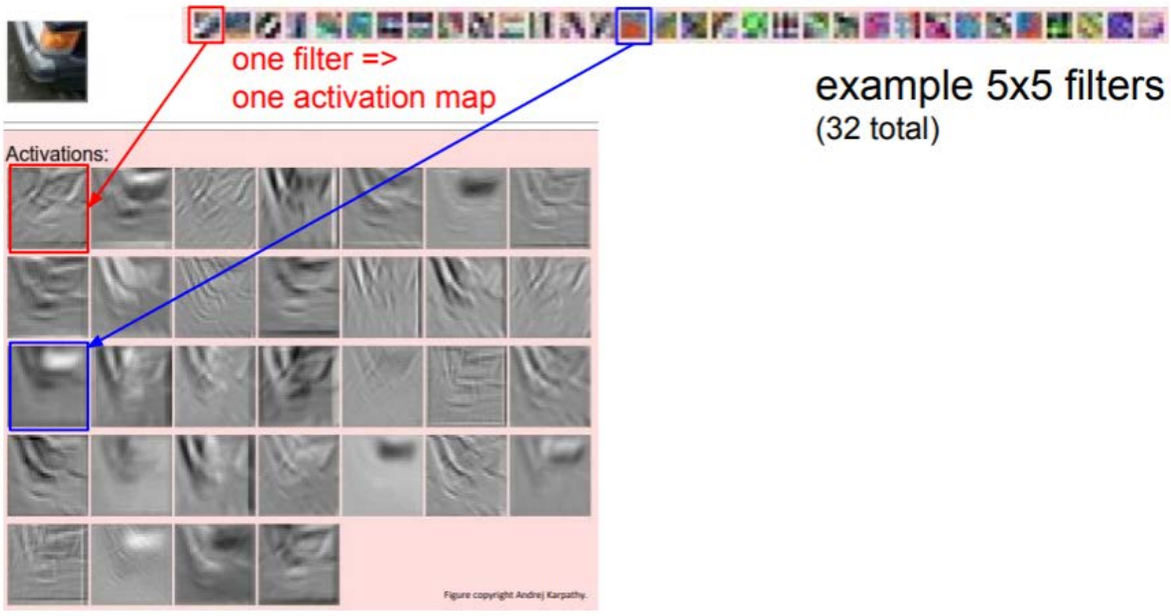




- 각 layer에서 convolution, ReLU (activation function), pooling layer 등 수행함
  - 여기서 나온 output은 다음 layer의 input이 됨
- Layer마다 필터도 여러 개 있고, 따라서 activation map도 여러 장이다



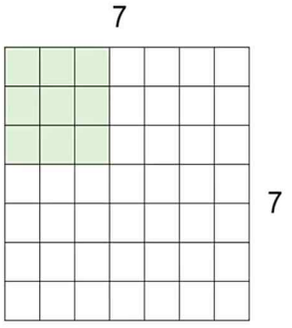
- ConvNet에 여러 장의 layer가 있으면 위계를 갖는 filter를 학습하게 된다.
  - 초기 Layer에서는 가장자리와 같이 low-level 특성을 학습.
  - 중간 Layer에서는 색칠된 부분 또는 모서리와 같이 조금 더 복잡한 특성을 학습
  - 상위 Layer에서는 더 복잡한 개념을 학습
  - Layer가 깊어질수록 Simple → Complex features
    - 고양이 뉴론 실험과 비슷한 구조
- Layer를 얼마나 깊게 쌓을 것인지는 설계자의 선택에 따라 다르다
  - 모델 성능을 보고 경험적으로 판단
  - Size of filter, stride 등 선택
- Conv_1: 첫 번째 Convolution Layer
  - 작은 상자 한 개는 뉴론 한 개 → Back propagation을 수행해 해당 뉴론의 activation 값을 최대화하는 input이 상자 안에 이미지다



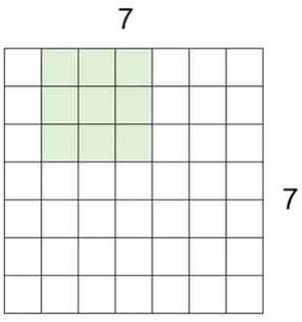
- 위 사진은 자동차 모서리를 학습하며 각 필터가 생성한 activation map을 보여준다
  - 우측 상단은 5x5 filter를 나열
  - 아래는 activation map: 각 필터가 강조하는 부분(orientation)은 더 높은 값(흰색)으로 출력

[Filter 의 적용]

7*7 입력에 3*3 필터를 적용했다고 해 보자.



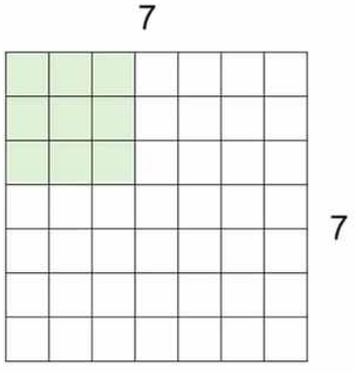
7x7 input (spatially)
assume 3x3 filter



7x7 input (spatially)
assume 3x3 filter

- 위같이 필터를 이미지 왼쪽 상단부터 적용하여, 해당 값의 내적을 시행한다.
- 움직여가면서 내적을 시행하면 결국 5*5 출력을 얻게된다.
- 이 경우 좌우방향 5번 수행 가능하므로 5*5 의 ouput을 얻게된다.
- 움직이는 칸을 stride 라 한다. stride = 1 이면 한칸씩만 움직인다.
- slide 가 2 이면 2칸씩 움직인다. 위 경우에 stride 가 2 이면 출력은 2가 된다.

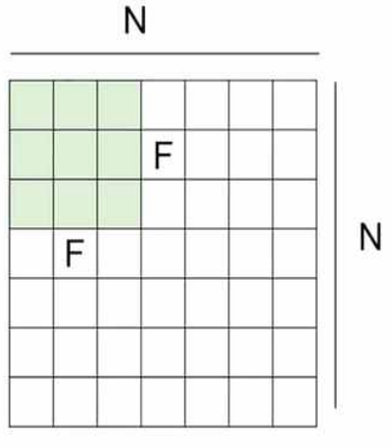
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

• 주의 : 위 예시에서 stride 가 3이면 모든 이미지를 cover 하지 못한다. 이는 맨 끝의 edge 정보를 이용하지 못하게 되기 때문에 불균형한 결과를 낳을 수 있다.
이런 불균형한 결과를 사전에 방지할 수 있는 방법이 없을까?



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

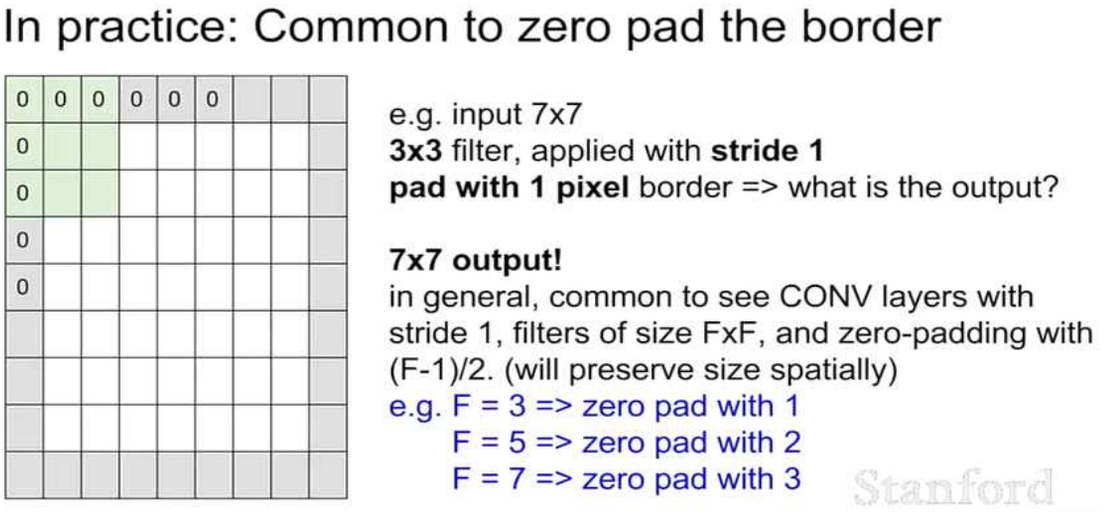
• 위 경우처럼 모든 output size를 계산할 수 있는 좋은 공식이 있다.

• N 은 이미지의 크기이고 F는 필터의 크기이다. stride(보폭의 크기라고 이해하면 쉽다)와 함께 3개의 값을 이용해 ouput size 의 크기를 알아볼 수 있다.
• stride = 3을 대입해보면 2.33 이 나옴을 알 수 있다. 즉 잘 떨어지지 않는 경우는 filter 가 전체 이미지에 딱 맞게 적용되지 않는다는 의미고, 이는 불균형한 결과를 낳을 수 있다.

[Q. 각 필터는 어떻게 적용해야할까?]
각 필터는 모든 depth 에 대해 ouput 의 depth 는 우리가 가진 필터의 개수가 된다.
7*7*3 의 input 에 3*3 filter를 10개 적용하면 그 output 은 7*7*10 이 된다.

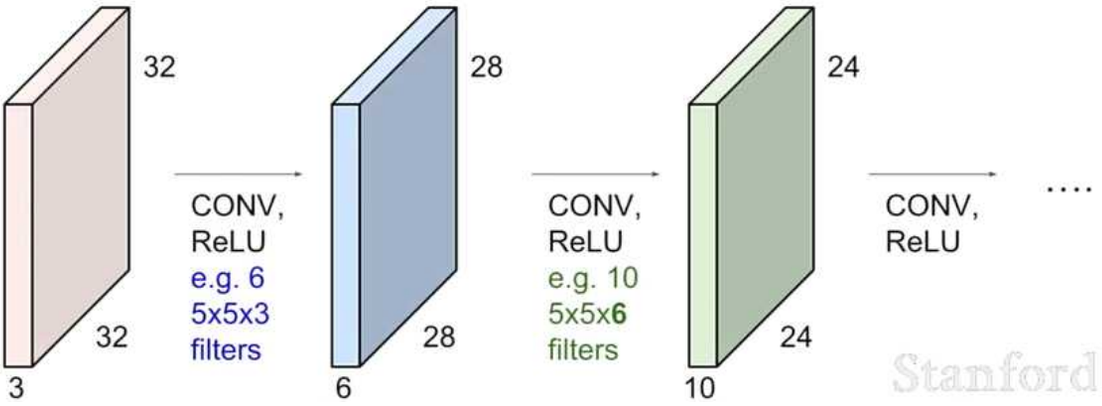
[Q. 이미지가 정사각행렬이 아니라 다른 모양이면 filter 도 다른모양을 사용하나요?]
이미지 모양이 달라도 보통 정사각형 모양의 filter를 사용한다.

[Zero padding]



• 제로패딩은 내가 원하는 출력을 얻고싶을 때 사용한다.
• 주로 출력과 입력의 사이즈를 유지하기 위해 사용하고, 그 때의 패딩값은 (F-1)/2 가 된다.

[Q. 왜 zeropadding을 사용할까?]
• padding을 하게되면 출력 사이즈를 유지시켜주고 필터의 중앙이 닿지 않는곳도 연산할 수 있다.
• 일반적으로
3*3 filter stride 1
5*5 filter stride 2
7*7 filter stride 3 을 쓴다.
• 레이어가 여려겹 쌓이고, zero padding을 하지 않는다면, 출력사이즈는 매우 빨리 줄게된다.



• 위 예시를 보면 32 -> 28 -> 24 ... 로 매우 빨리 줄어들고 있음을 볼 수 있다.
• activation map 이 점점 줄어들게되고, 그러면 각 코너의 정보를 충분히 학습하기 전에 잃게 되는것이고 원본 이미지를 충분히 학습할 수 없을 것이다.

[Q. 0을 padding 하면 쓸데없는 정보가 추가되는것이 아닐까?]
padding을 하는 이유중 하나는 모서리부분에서 값을 얻고자 함이다. 그래서 필터가 닿지 않는 모서리 부분에서 값을 얻게하려는 것. 물론 값을 extend(맨 끝의 값을 늘린다)하는 방식도 할 수 있다.
0 padding 은 약간의 인위적인 방식이지만, 그래도 잘 작동된다.

[중간요약]

아래 그림은 위 내용을 총 정리한 것이다.

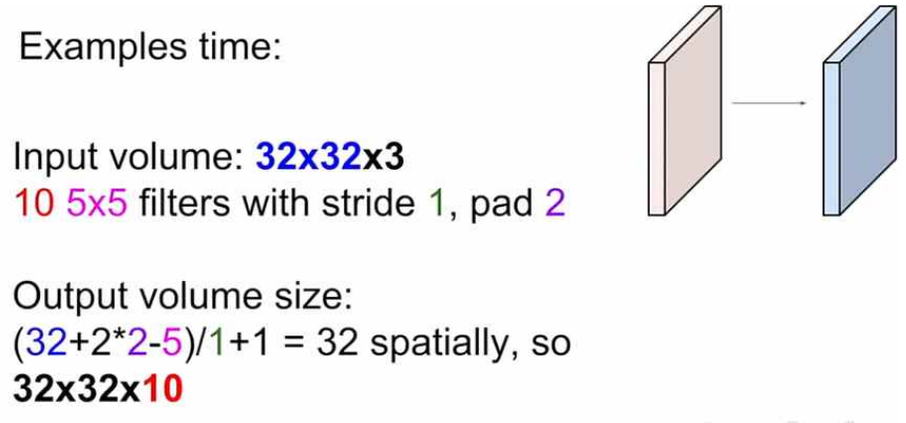
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

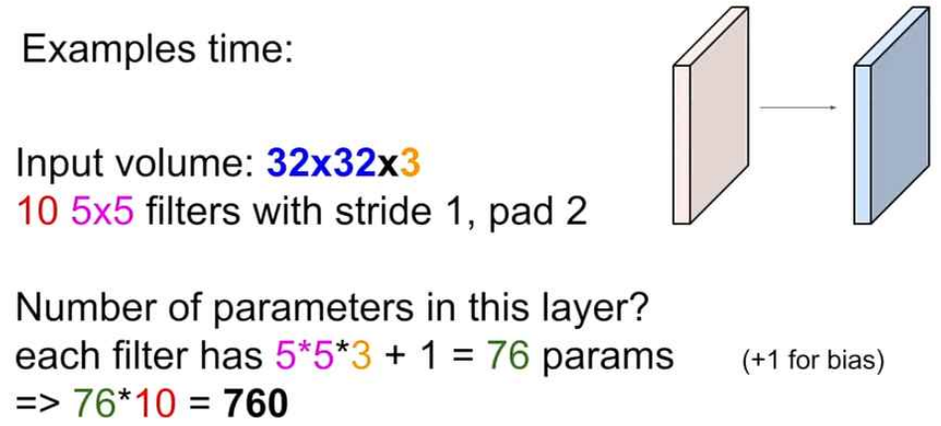
• n 차원의 입력에 대해서 몇 개 필터를 쓸것인지, 필터의 크기는 어떻게 정할것인지, stride의 크기, zero padding의 크기 등등은 알아서 결정해야한다.
• 필터 사이즈는 3*3 / 5*5 / 7*7을 자주 사용한다.
• 보통 필터의 개수는 2 제곱수로 한다. 16 32 ..... 이는 그냥 관습이라고 한다.

[Examples]

예시를 풀면서 위 내용들을 정리해보자.

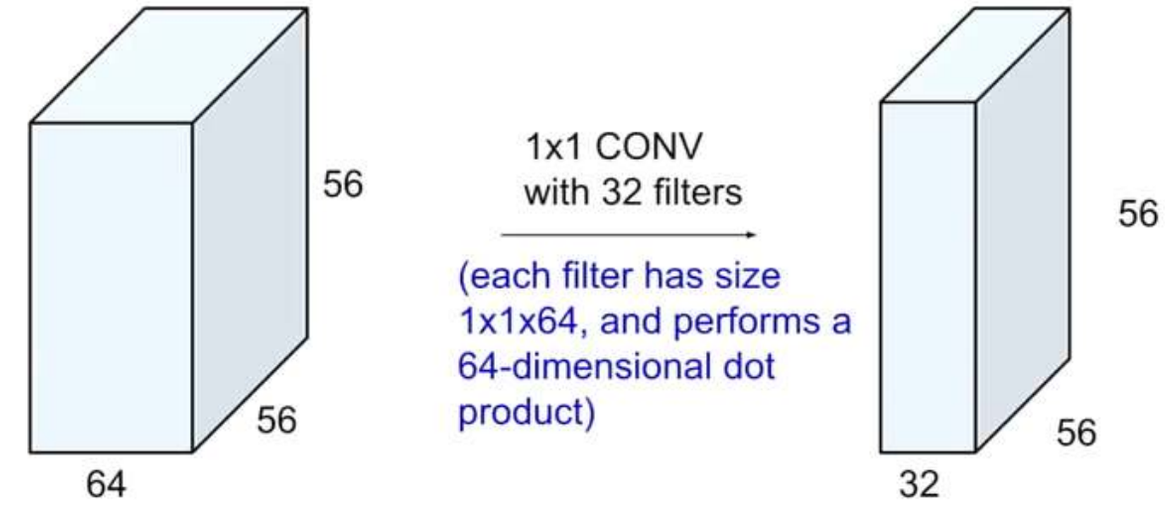


• input 이 32*32*3, 10개의 5*5 filter with stride 1, padding size 2를 쓰게 된다면 ouuput size 는 32*32*10(filter의 수) 가 된다.



• 위 예시는 pramater 의 수를 계산하는 것이다. 각 filter는 5*5*3 개의 가중치와 1개(bias) 로 가지고 있으므로 76개의 parameter를 가지고 있다. 또 이는 총 10개이므로 760개

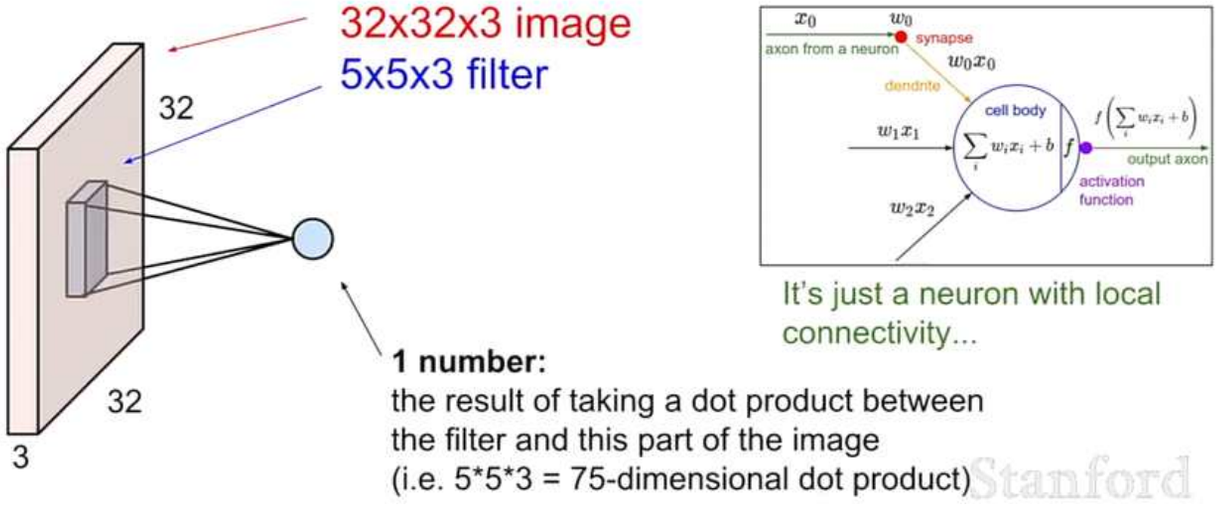
[1*1 Convolution]



• 필터는 depth 만큼 연산을 수행한다. 즉 전체 depth 에 대한 내적을 수행하는것과 같다.
• 슬라이딩하면서 연산을 하지만 1*1 이기에 공간적인 정보는 없다. 그저 전체 depth에 대한 내적을 수행한다.
• 입력은 56 *56 * 64 이고 여기에 32개의 filter를 적용하면 56 56 32 의 출력값이 나온다. (parameter를 크게 줄일 수 있음)

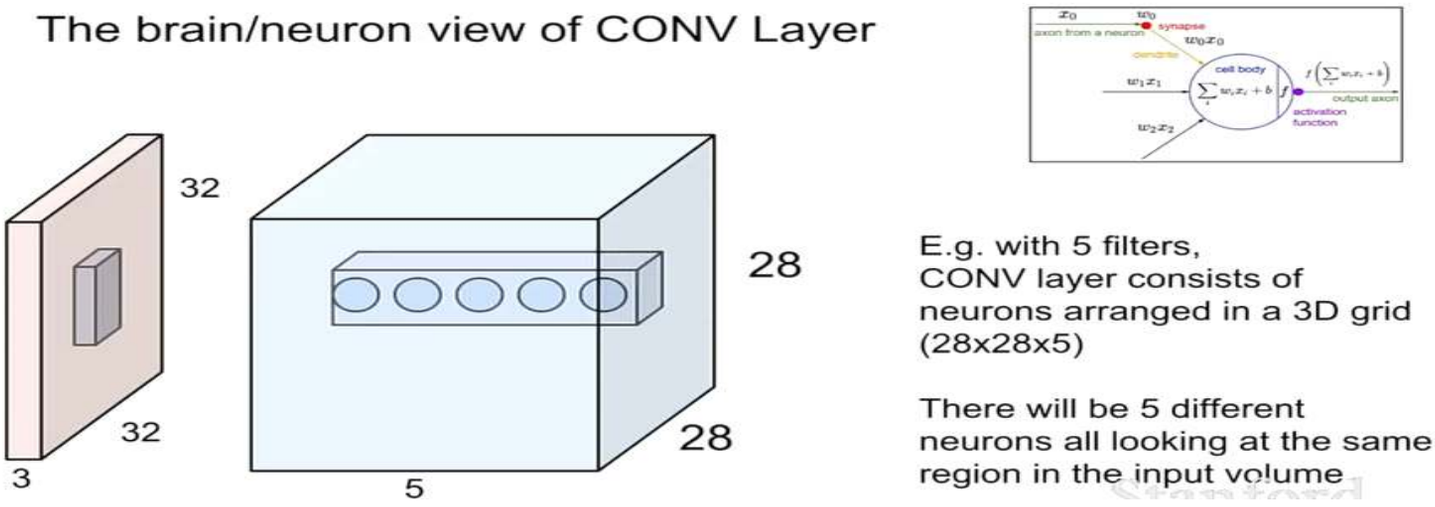
[Q. stride를 선택하는데 가질수 있는 직관이 있나요?]
stride를 크게 할수록 출력은 점점 작아진다. -> 즉 이미지를 downsampling 하는 역할이다.
activation map 의 사이즈를 줄이는 것은 모델의 전체 파라미터의 개수에도 영향을 끼친다.
만약 stride를 심하게 주면 convlayer 의 출력이 너무 작아져 fc 가 이용할 수 있는 정보가 적을 것
파라미터 수 모델사이즈 오버피팅 등은 서로 trade off 가 있고, stride를 몇으로 할지는 이와 동시에 고려되어야 한다.

[Compare With NN]



• 이미지의 일부분과 filter가 겹치는 부분에서 내적을 구하는 것과 같은 방식은 NN의 관점에서 각각의 x에대해 w이라는 weight를 줘서 내적을 구하는 것과 동일
• NN과 conv의 차이점은, conv에서는 뉴런이 local connectivity를 가진다는 점, 즉, entire input에 대해 각각의 뉴런이 연결되어있는 것이 아니라, filter가 지나는 이미지의 일부분에 대해 뉴런이 연결되어있다고 생각하면 됨.

※ 잠깐! 용어상식 [receptive field]
• 한 뉴런이 한번에 수용할 수 있는 영역
• 5*5 필터가 있다 -> 한 뉴런의 receptive field (한 뉴런이 한번에 수용할 수 있는 영역) 가 5*5 라는 것

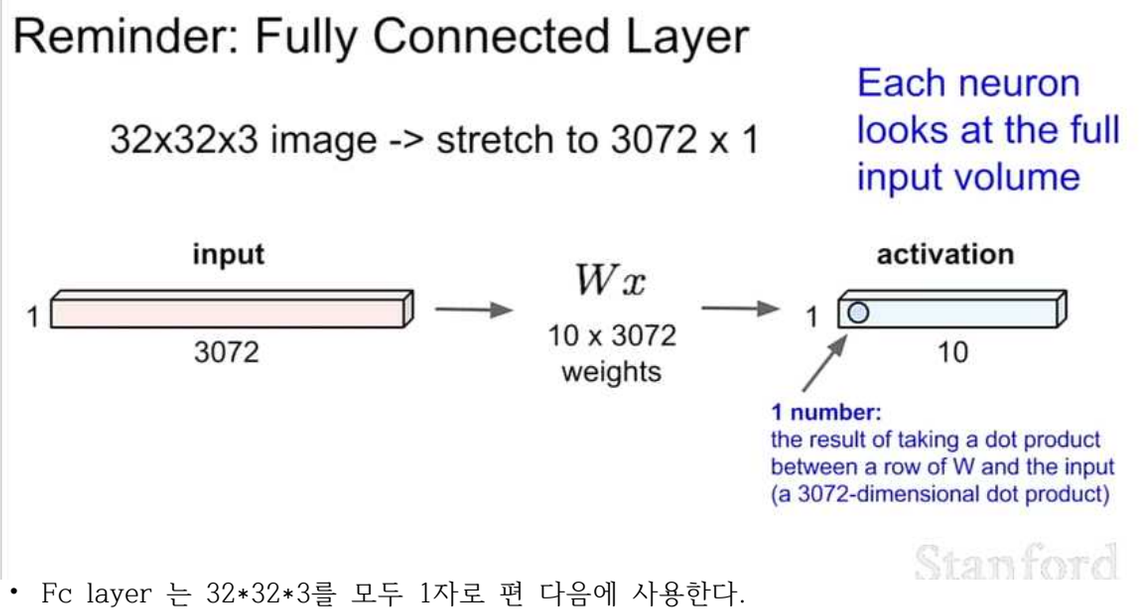


• 필터가 5종류이면 출력은 28*28*5 가 된다.

• 한 점을 찍어서 depth 방향으로 바라보면 (파란색 map 안의 5개의 점) 5개의 점은 같은 지점에서 추출된 서로 다른 특징이다. 즉 각 필터는 이미지에서 같은 지역을 돌더라도 서로 다른 특징을 뽑아낼 수 있다.

• 동일 depth 내의 뉴런들은 parameter sharing, 동일 위치의 뉴런들은 input 이미지의 같은 곳을 쳐다보게 된다.

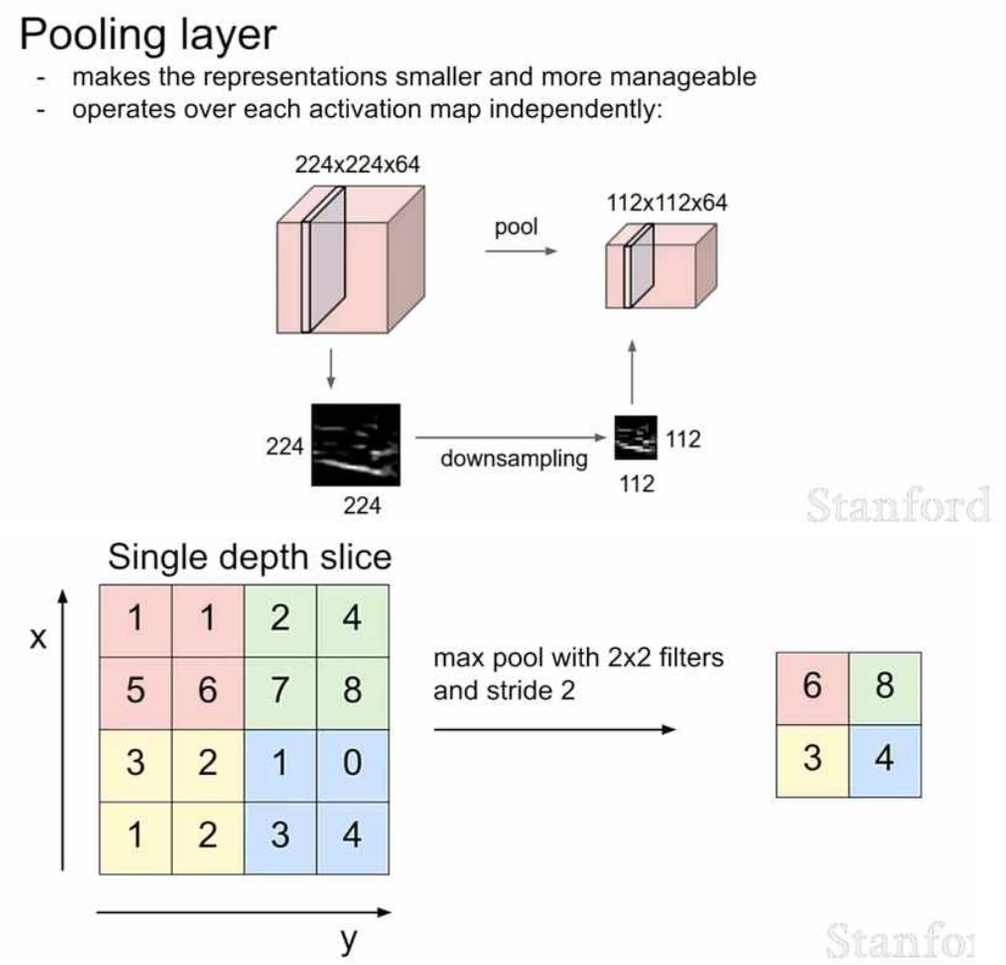
[Fully Connected Layer]



• Fc layer 는 32*32*3를 모두 1자로 편 다음에 사용한다.

• conv 는 local 한 정보를 이용하는것 과는 대비되는 특성이다.

[Pooling Layer]



- 풀링 레이어는 RESRESENTATION을 더 작고 관리하기 쉽게 만든다.
- 작아지면 파라미터의 수가 줄고, 공간을 줄여준다. 즉 downsampling을 해준다.
- depth 에는 아무짓도 하지 않는다.
- conv layer가 했던 것처럼 슬라이딩하면서 수행된다.
- stride가 존재하며 filter size도 존재한다..
- max pooling이 주로 사용된다

[Q, pooling을 할 때 겹치지않는 것이 일반적인가요?]
기본적으로 Downsample을 하고싶은 것이기 때문에 겹치지 않는 것이 일반적이다.

[Q. Maxpooling이 왜 Average pooling 보다 좋은가요?]
MAX를 사용하는지에 대한 직관은, 우리가 다루는 값은 이 뉴런이 얼마나 활성화되었는지를 알고싶은데, 큰 값은 신호에 대해 얼마나 그 필터가 더 많이 활성화되었는지를 알려준다. 그래서 maxpooling을 사용하게 된다. 그 값이 어디에 있었는지보다는 그 값이 얼마나 큰지가 중요한 것.
물론 average pooling을 사용할수도 있다.

[Q. Pooling 의 Stride 나 똑같은 역할을 하고있는것 아닌가요?]
둘다 Downsampling 의 한 종류라고 할 수 있다. 요즘에는 Pooling 보다는 conv 층의 stride로 down sampling 기법을 많이 사용

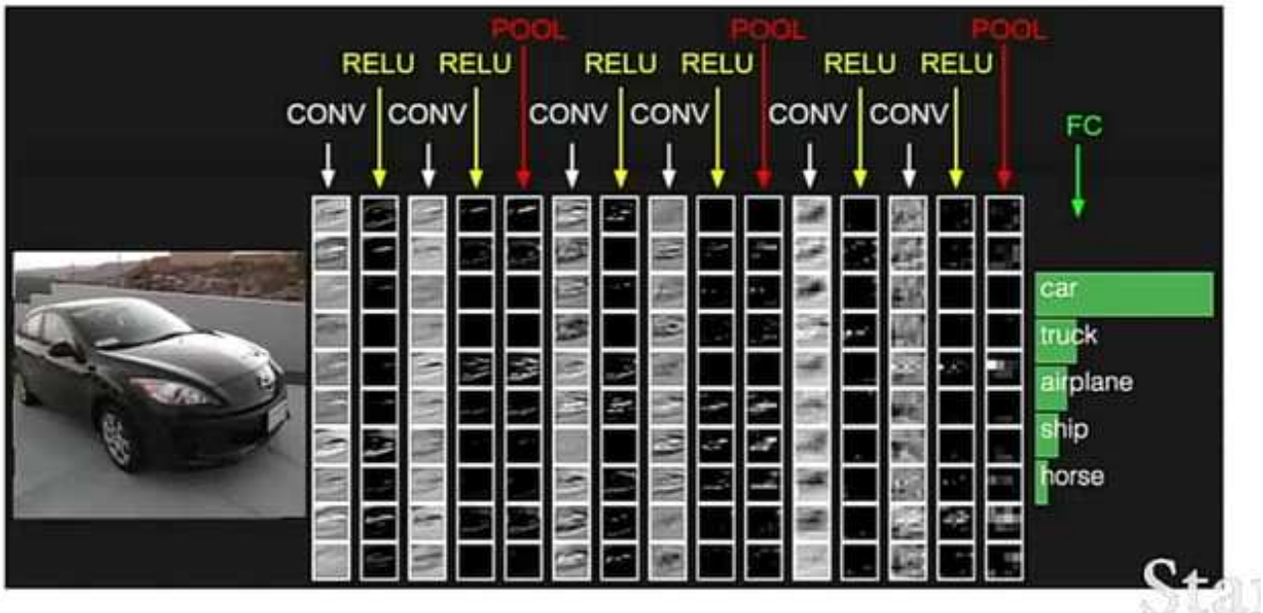
아래는 위에서 pooling layer의 ouput size 연산과정이다.

- 위 공식을 통해 Filter size 를 정해줄 수 있다. W - Filtersize / stride + 1

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

• pooling layer에서는 padding을 하지 않는다. 왜냐하면 우리는 downsampling 하고싶을 뿐이며, conv 때처럼 코너의 값을 계산못하지 않기 때문이다.
• 가장 널리 쓰이는 필터사이즈는 2*2, 3*3 이고 stride 는 둘다 2로 동일하게 쓴다.

[Summary]



• 기본적으로는 conv , pooling을 쌓아 올리다가 마지막에 fc layer로 끝이난다.
• 네트워크의 필터는 점점 작아지고 아키텍처는 점점 깊어지는 경향
• 요즘에는 pooling / fc layer를 없애고 cov layer 만 깊게 쌓는 추세이다.
• class score를 구성하기 위해 맨 마지막에 softmax 가 사용된다.
• 마지막 conv layer의 출력은 3차원 volume 으로 이루어진다.
• 이 값을 전부 펴서 1차원 벡터로 만든다. conv net의 모든 것을 사용해서 Fc layer 의 입력으로 사용한다.
• 마지막 layer 에서는 전부다 하나로 통합시키고 최종적 추론을 하기 때문에 공간적 구조를 신경쓰지 않는다.

[Q. 각 열들을 어떻게 해석해야하나요?]

각 열들의 사진들은 하나의 layer 에 대한 결과들이다.
시간이 지날수록 복잡한 패턴들을 찾아내고. 마지막에 이런 값들을 가지고 FC가 해석하고 결과(확률)을 낸다.

[Q. 어떻게 구조를 구성해야하는가?]
각자 어느정도의 직관을 가지고 구조를 직접 구성해보면서 비교해야한다.
다양한 pooling size , fiilter size 등을 가지고 cross validation을 해 본다,
어떤 하이퍼 파라미터가 좋을지는 문제에 따라 달라진다.