이미지 처리를 위한 모든 것

02. Alexnet에서부터 시작하는 딥러닝 여행

카이스트

김현우

0 본인 소개

김 현 우

카이스트, 산업및시스템공학과 대학원생

TEAM-EDA 블로그, 페이스북 페이지 운영

Recommender System KR 페이스북 그룹 운영진

깃허브 : https://github.com/choco9966/

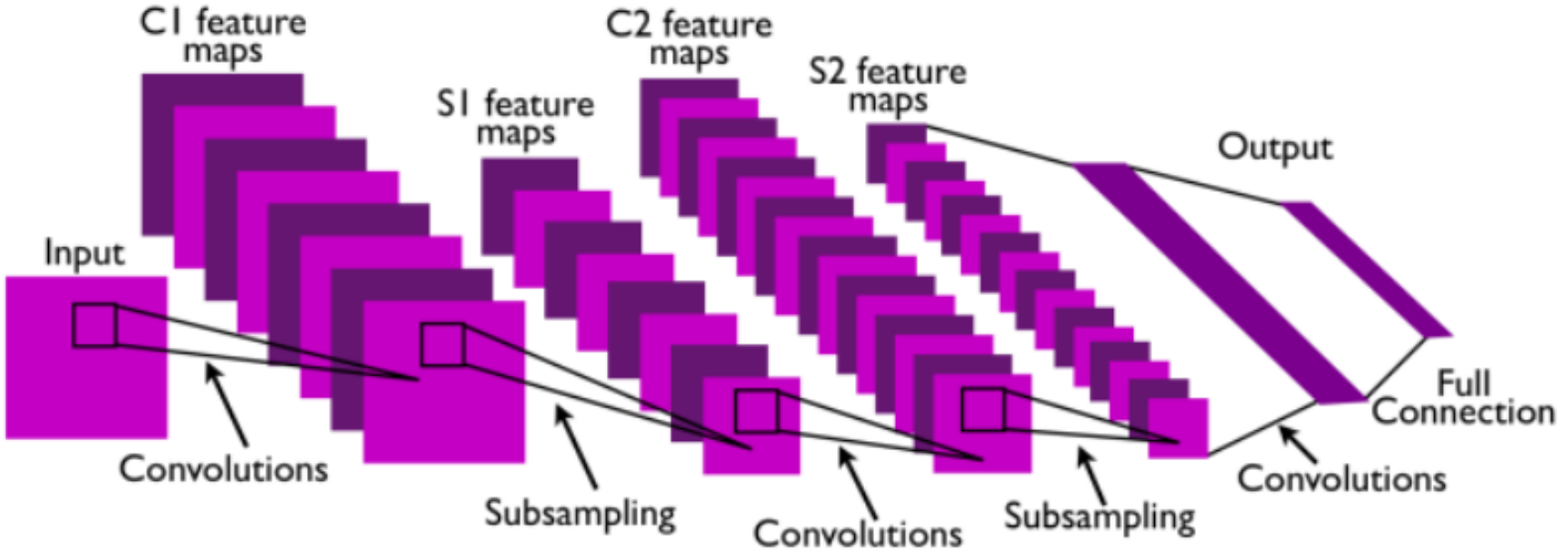
01 Alexnet (CNN)

1 CNN (Alexnet)

정의

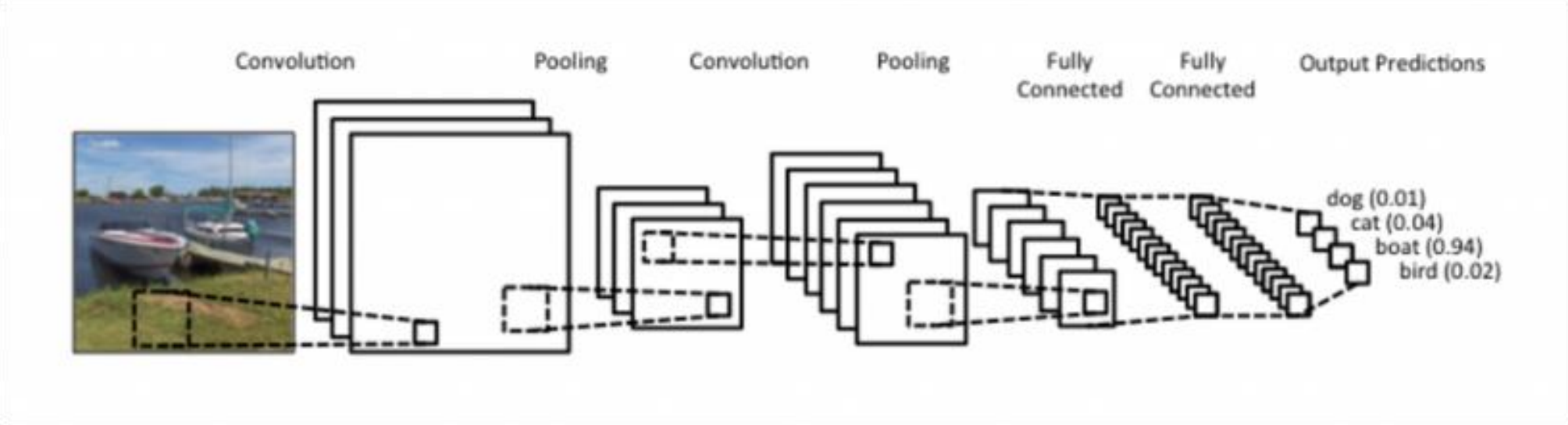
CNN은 이미지가 들어오면 3가지의 과정을 통해서 Output을 생성합니다.

1. Convolutions : Feature maps 생성
2. Subsampling : 생성된 Feature maps의 차원을 감소
3. Full Connection : Output(Fully Connected Layer)을 산출.



1 CNN (Alexnet)

과정



1. Convolutions이 사진 전체를 돌아다니면서 보트의 특징을 추출 (Feature Extraction)
- Convolutions을 Filter라고도 하고 위에는 3개의 필터를 통해서 특징 맵 3개를 추출한 것임

2. Subsampling 과정을 pooling이라는 방법을 통해서 진행
- 해당 과정을 통해서 추출된 특징의 사이즈를 줄일 수 있음

3. Fully Connected Layer는 개인지 고양이, 보트 등 최종 분류를 진행

1 CNN (Alexnet)

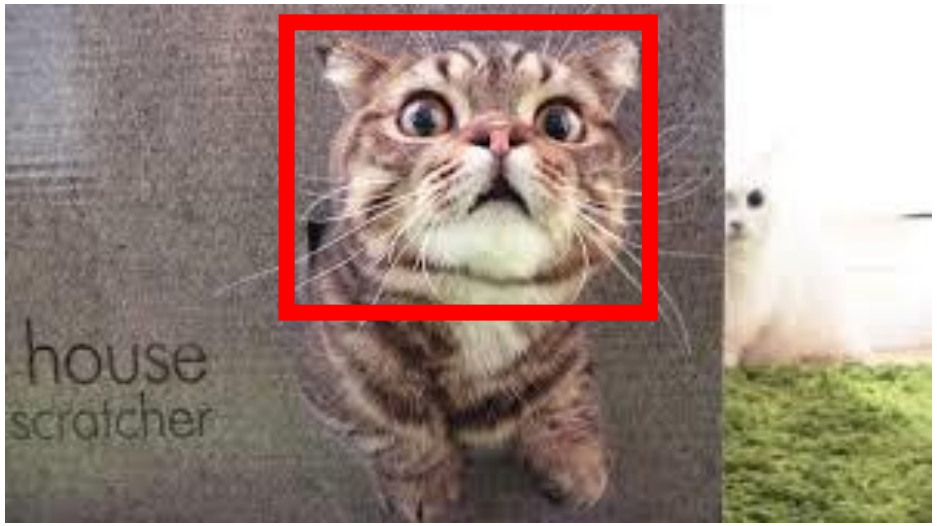
가정

“They also make strong and mostly correct assumptions about the nature of images (namely, **stationarity of statistics** and **locality of pixel dependencies**).”

1 CNN (Alexnet)

Stationarity of statistics

이미지는 위치에 관계없이 동일한 패턴들이 반복되는 특징을 가지고 있다.
- 즉, 특정 위치에서 학습한 파라미터를 이용해서 다른 위치에 있는 동일한 특징을 추출할 수 있음



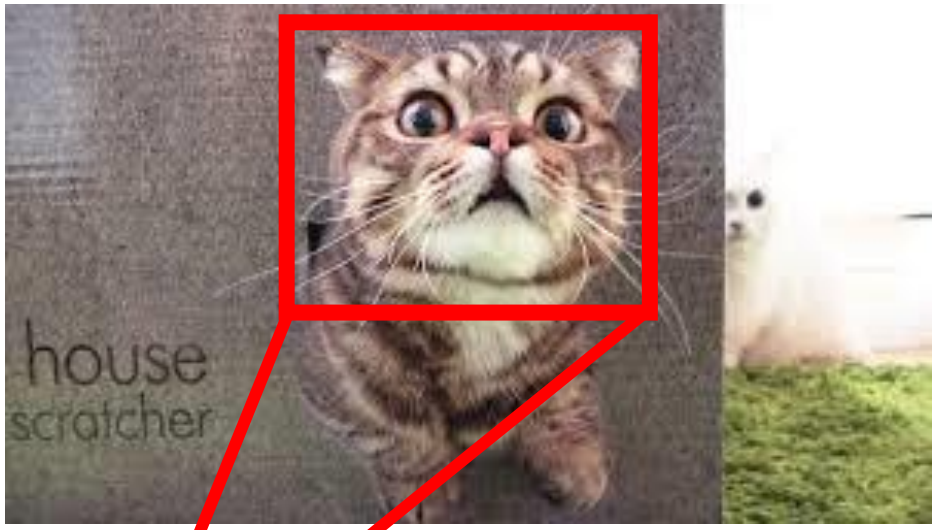
- 위의 그림에서는 고양이의 얼굴이라는 특징을 빨간색 네모라는 파라미터가 특징을 추출함
- 이미지에 위치에 상관없이 동일한 특징이 존재한다는 가정이기에 Convolution 연산이 잘 작동

1 CNN (Alexnet)

Translation Equivariance

입력의 위치가 바뀌면 출력도 동일하게 위치가 바뀐다.

2	5	2	1
3	2	1	3
4	1	4	2
2	1	3	2



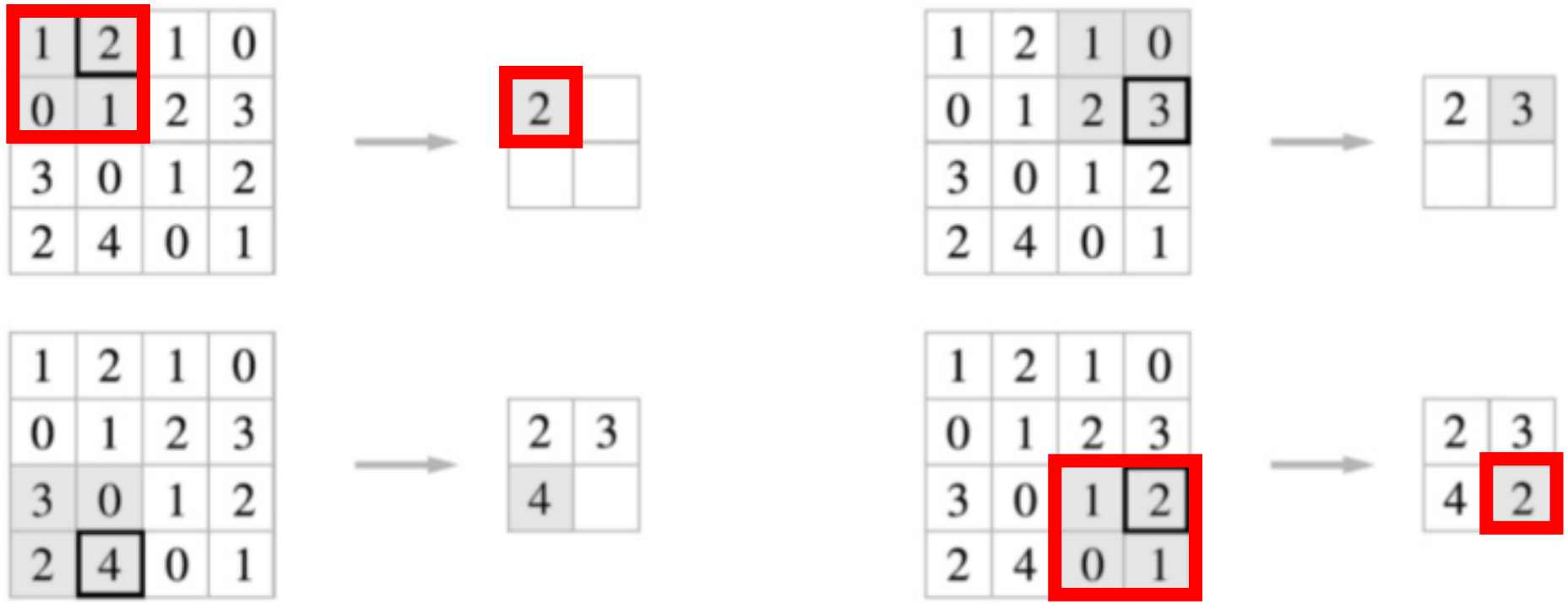
4	1	3	2
3	2	5	3
4	1	2	2
2	1	3	2

강아지의 얼굴을 의미하는 출력의 위치가 달라짐

1 CNN (Alexnet)

Translation Invariance

입력의 위치가 바뀌어도 출력은 그대로이다. (Max Pooling)

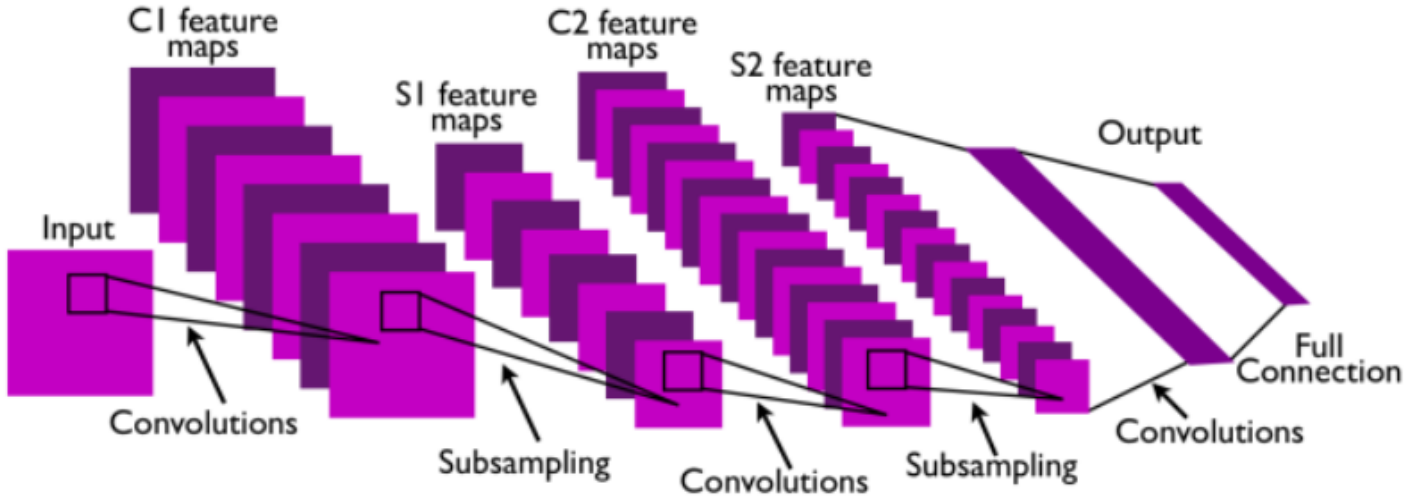


[[1, 2], [0, 1]] 이라는 동일한 입력의 위치가 달라도 출력은 2로 동일함

1 CNN (Alexnet)

Translation Invariance

입력의 위치가 바뀌어도 출력은 그대로이다. (Fully Connected Layer)



고양이 : 0.91
강아지 : 0.02
보트 : 0.04
새 : 0.03

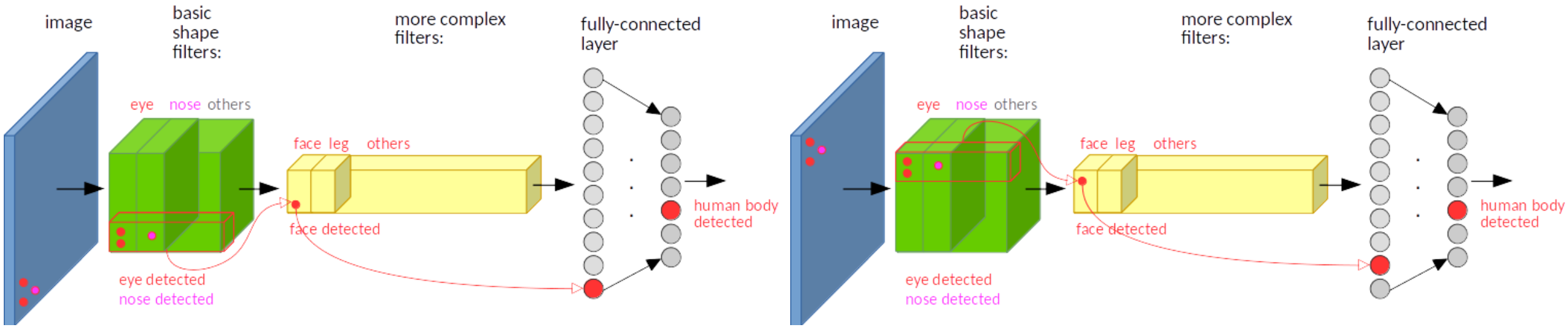
고양이 : 0.95
강아지 : 0.01
보트 : 0.02
새 : 0.02

Fully Connected Layer의 Output인 “고양이”라는 결과가 동일함

1 CNN (Alexnet)

Translation Equivariance & Invariance

CNN은 Translation Equivariance와 Invariance을 모두 포함한다.



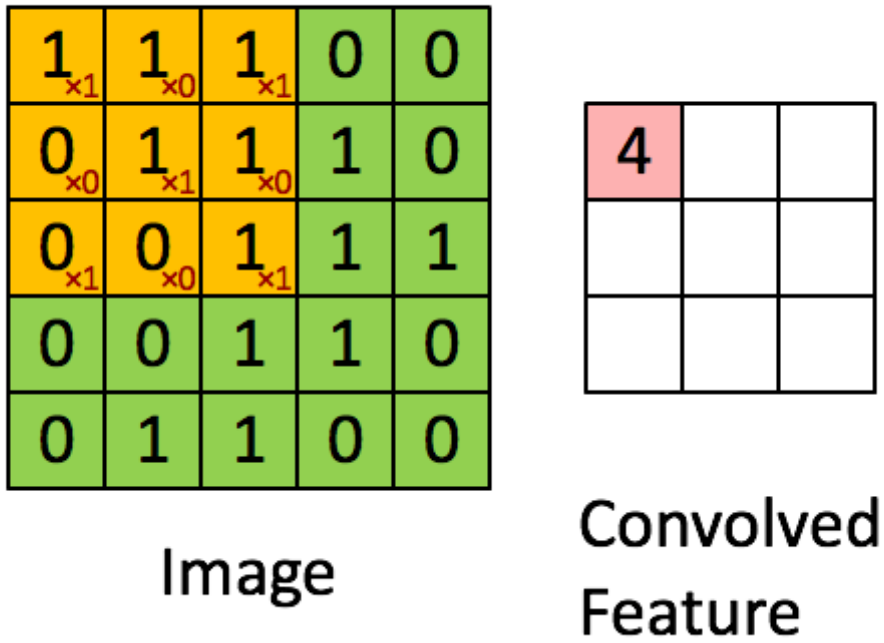
1. Equivariance : 눈, 코를 의미하는 빨간색 점의 위치가 바뀌면 Feature maps에 눈, 코를 의미하는 특징의 결과는 동일하지만 위치만 변함

2. Invariance : fc layer을 거치고 마지막 label 확률 벡터를 출력하는 부분에서는 위치와 관계없이 human body를 detected 했다고 함

출처 : https://medium.com/@seoilgun/cnn%EC%9D%98-stationarity%EC%99%80-locality-610166700979

1 CNN (Alexnet)

Convolution이란?



1. 3 x 3의 Convolutions(Filter)이 5x5의 이미지를 모두 돌면서 특징을 추출함
- 왼쪽의 Convolutions은 각각 어떤 값을 곱할지에 대한 값을 가지고 있음

2. 여러개의 Filter를 사용하면 여러개의 Feature map을 얻을 수 있음

1 CNN (Alexnet)

Stride 이란?

Stride란 Filter가 이미지를 탐색할 때, 몇칸의 간격으로 건너서 학습하는 지를 의미합니다.

[Stride가 1인 경우]

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

[Stride가 2인 경우]

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

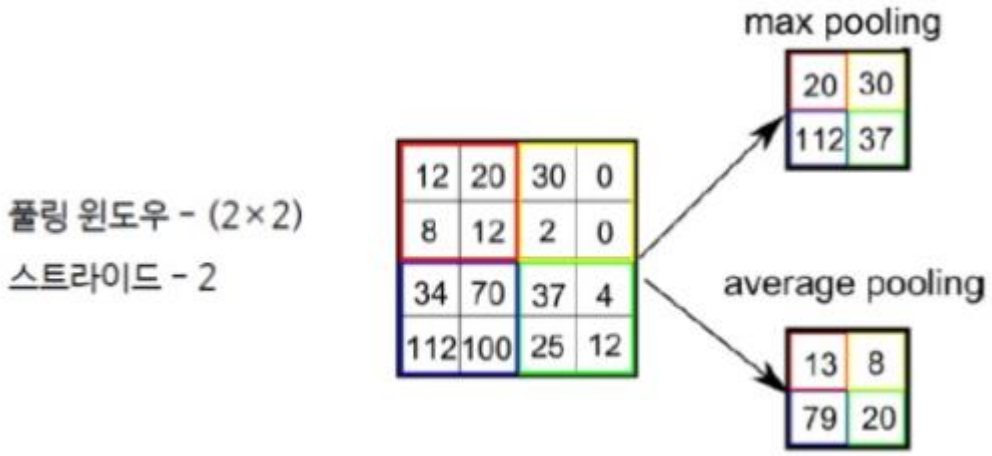
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1. Stride를 크게 할 수록 Feature Maps의 크기가 줄어들어서 pooling과 같이 이미지 축소의 역할을 함
2. Stride가 1인 경우에는 첫번째 행의 숫자 2가 2번 학습되지만, Stride가 2인 경우에는 한번만 학습되어서 오버피팅을 방지 (좀 더 자세히 말하면, Stride가 1인 경우에는 가장자리는 한번 학습되지만 내부의 값은 여러번 학습되는 값들이 있어서 서로간의 정보의 차이가 발생함)
3. zero-padding과의 차이는 가장자리의 손실을 없애면서 특징을 추출하냐 없애지 않으면서 추출하냐에 있음

1 CNN (Alexnet)

Pooling 이란?

Subsampling의 한 방법으로 Filter에 의해서 나온 Feature maps의 차원을 줄이는 방법
- 학습시간을 줄이고 shift, distortion에 덜 민감해지는 특징



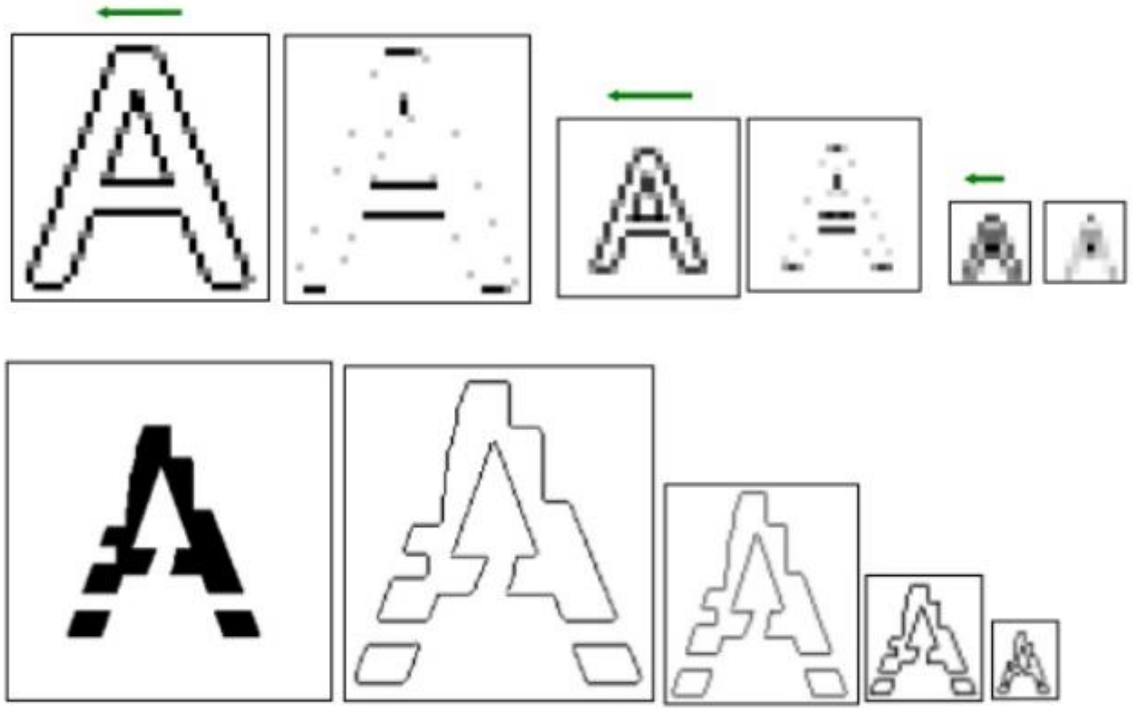
1. 대상 영역에서 최댓값 혹은 평균을 취하기에 학습할 매개변수가 필요하지 않음
2. 입력 데이터가 조금 변해도 풀링의 결과는 변하지 않음

출처 : https://www.slideshare.net/agdatalab/deep-learning-convolutional-neural-ne

1 CNN (Alexnet)

Pooling 이란?

Pooling Layer 효과 예시
- 이미지의 크기가 줄어도 모습은 동일한 것을 볼 수 있음
- Capsule Net 논문에서 Stride를 통한 차원감소가 Pooling보다 좋다고 언급됩니다.

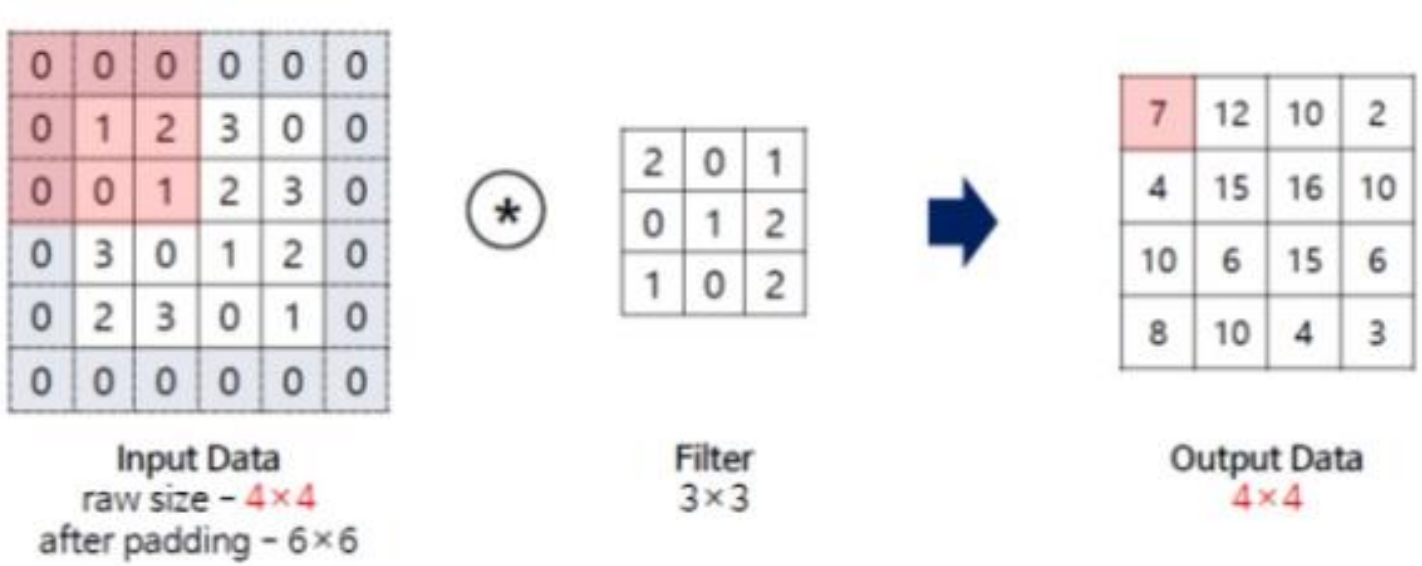


출처 : https://www.slideshare.net/agdatalab/deep-learning-convolutional-neural-ne

1 CNN (Alexnet)

Zero-padding 이란?

- 합성곱 연산을 수행하기 전에 입력데이터 주변을 특정값으로 채우는 단계로, 입력데이터와 출력데이터의 크기를 맞추기 위해서 사용

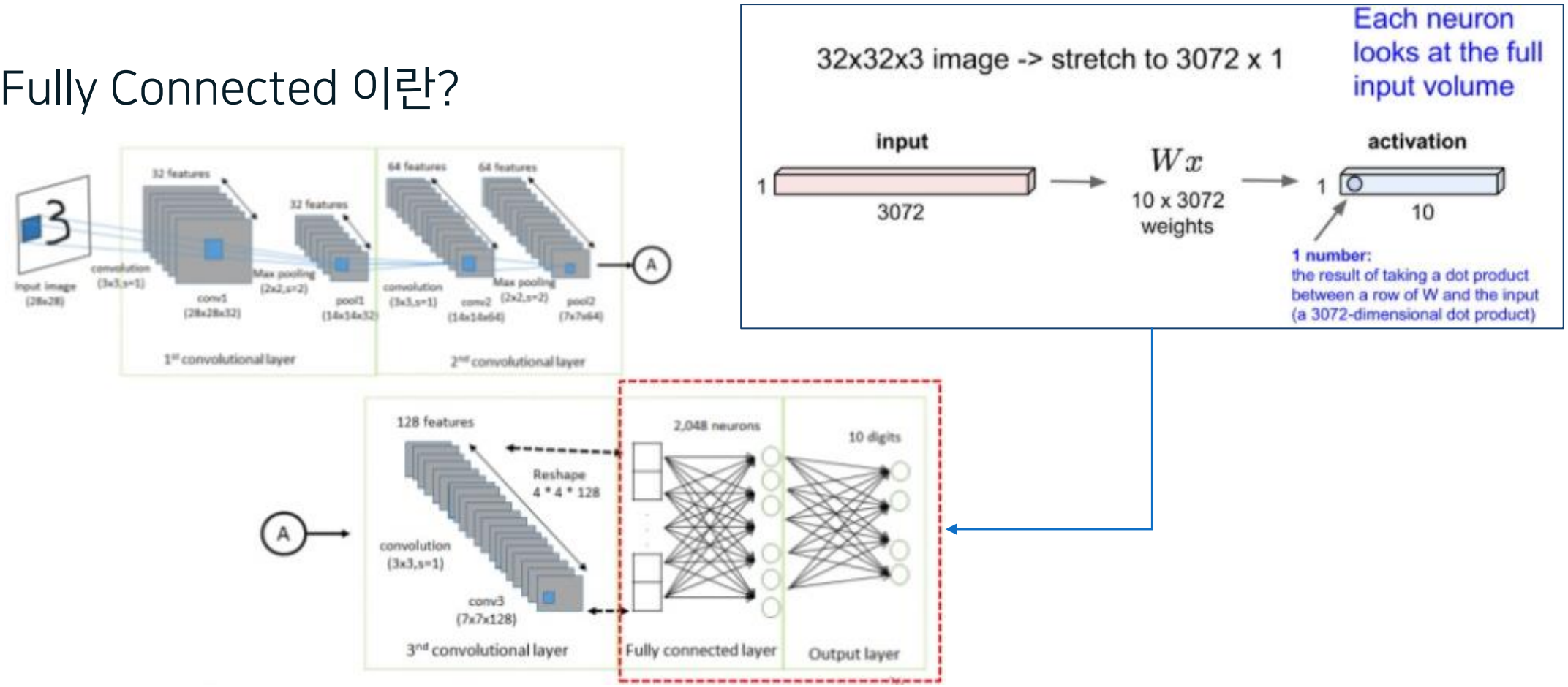


Zero-padding을 통해서 3가지의 장점을 얻을 수 있음

- Input Data 4x4가 Filter후에도 4x4으로 유지되는 것을 볼 수 있음 (이미지의 구조를 유지)
- 가장자리의 정보를 줄이지 않을 수 있음
- 0이라는 값이 Feature Maps에 반영되어서 오버피팅을 방지

1 CNN (Alexnet)

Fully Connected 이란?



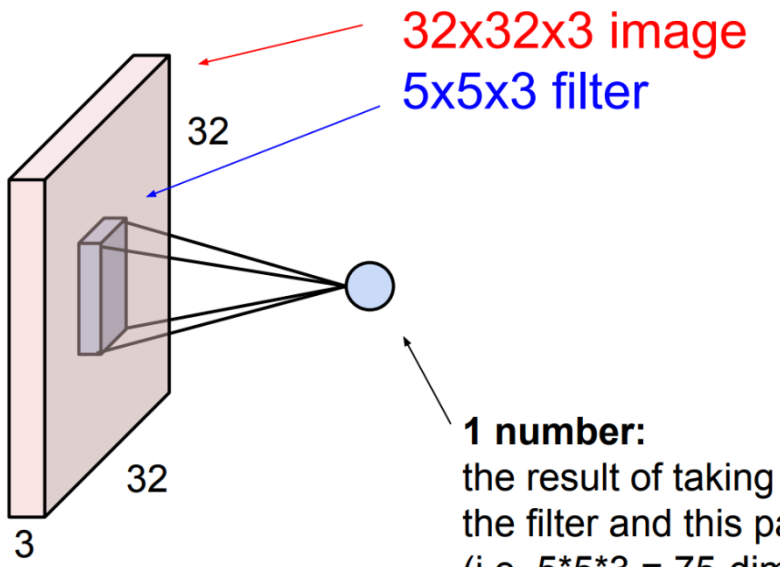
Fully Connected Layer는 이전 모든 노드의 값을 입력으로 받아서 ANN을 적용하는 방법

출처 : https://www.slideshare.net/agdatalab/deep-learning-convolutional-neural-ne

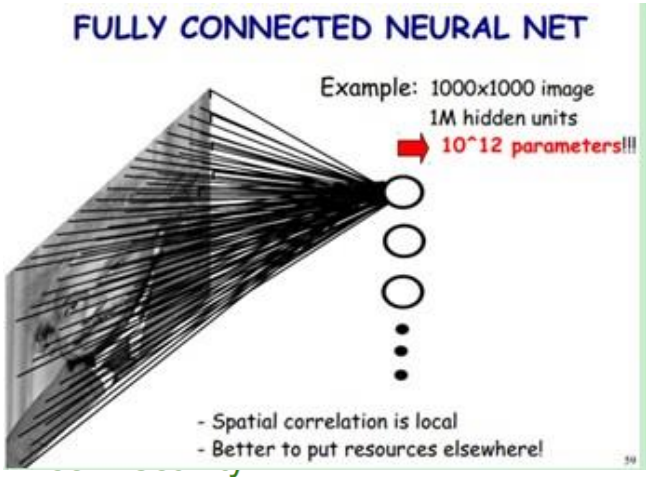
1 CNN (Alexnet)

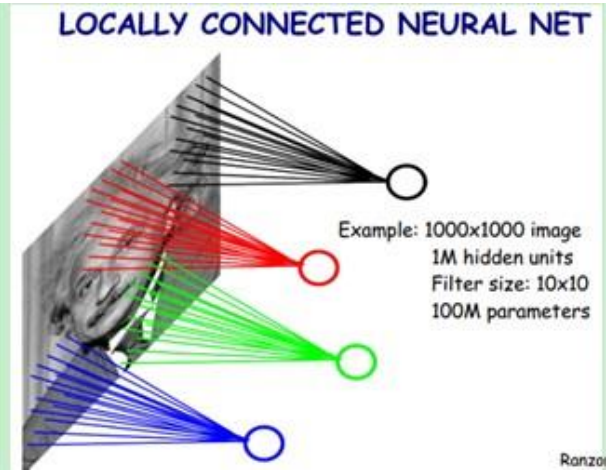
Local connectivity 이란?

- 현재 레이어의 한 뉴런을 이전 볼륨의 모든 뉴런들과 연결하는 것이 아닌, 입력 볼륨의 로컬한 영역(receptive field)에만 연결
- filter가 보는 영역이 receptive filed (kernel size)



1 number:
the result of taking a dot product between
the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)





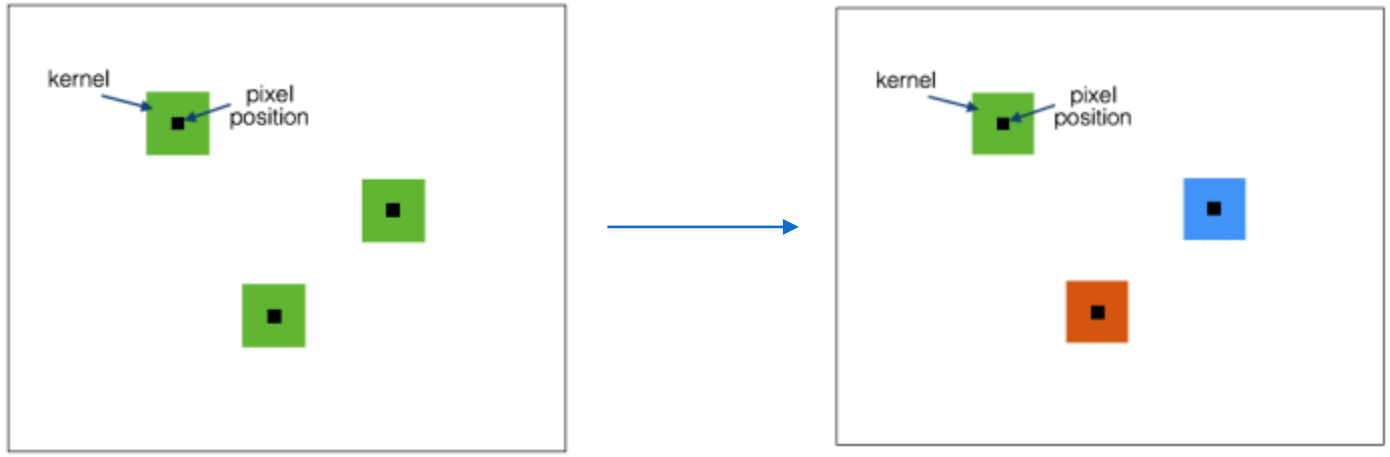
이미지 출처 : https://forum.huawei.com/enterprise/en/convolutional-neural-networks/thread/565377-887

- (32-5)+1인 28*28xfilter 수는 파라미터를 sharing함 (parameters sharing)

출처 : https://zzsza.github.io/data/2018/05/14/cs231n-cnn/

1 CNN (Alexnet)

Locally Connected Layer이란?



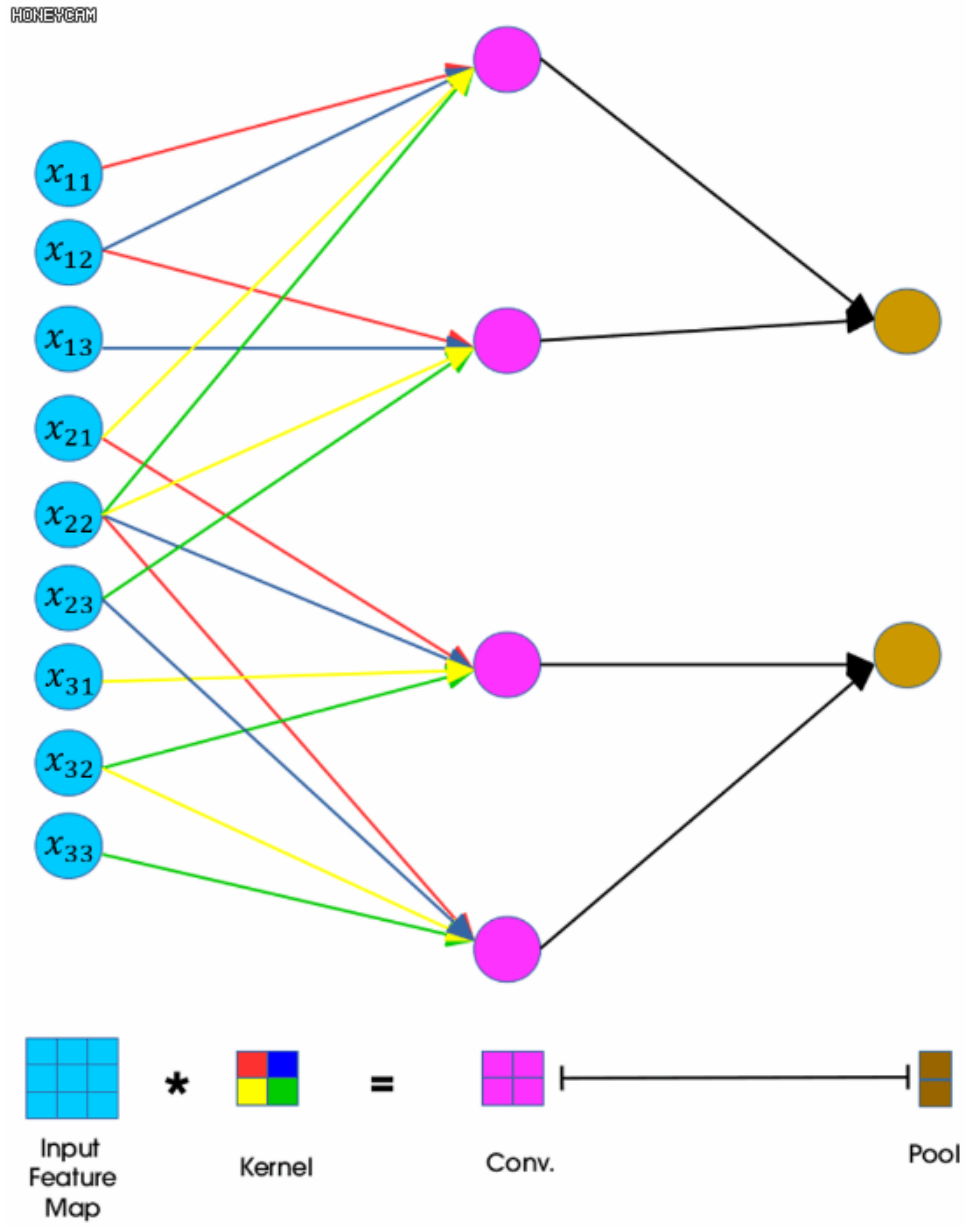
- 기존의 filter가 weight sharing을 하면 파라미터의 수는 28*28*5 입니다.
- 하지만, Locally Connected Layer에 의하면 28*28*5*(5*5) 으로 kernel size만큼 추가가 됨
  - 즉, 위치마다 filter의 weight 값이 전부 달라짐 !!!

출처 : http://aikorea.org/cs231n/convolutional-networks/

1 CNN (Alexnet)

Back propagation

위는 입력값 3x3 행렬에 2x2 kerne을 Stride1로 적용한 경우

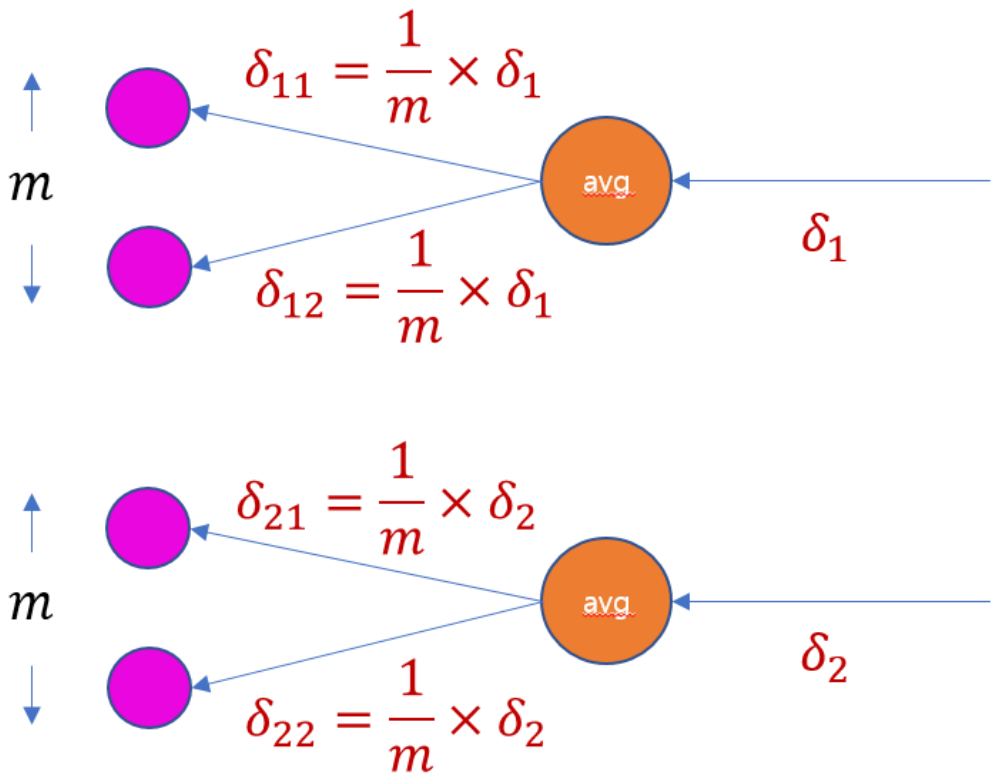


출처 : https://ratsgo.github.io/deep%20learning/2017/04/05/CNNbackprop/

1 CNN (Alexnet)

Back propagation – Average Pooling

그라디언트 값이 단순하게 1/m 로 쪼개짐

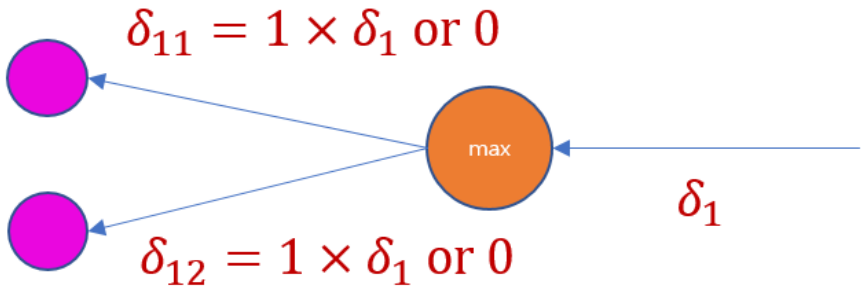


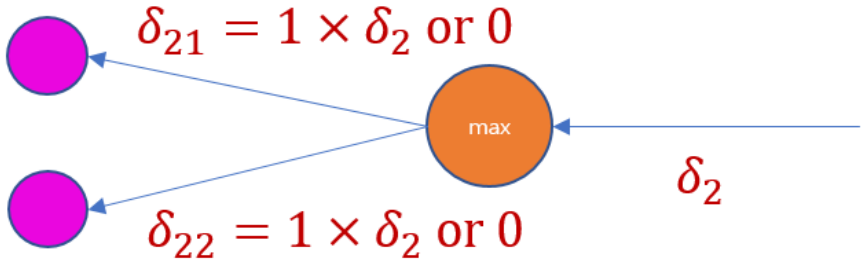
출처 : https://ratsgo.github.io/deep%20learning/2017/04/05/CNNbackprop/

1 CNN (Alexnet)

Back propagation – Max Pooling

최대값이 속해 있는 요소의 그라디언트는 1, 그렇지 않으면 0



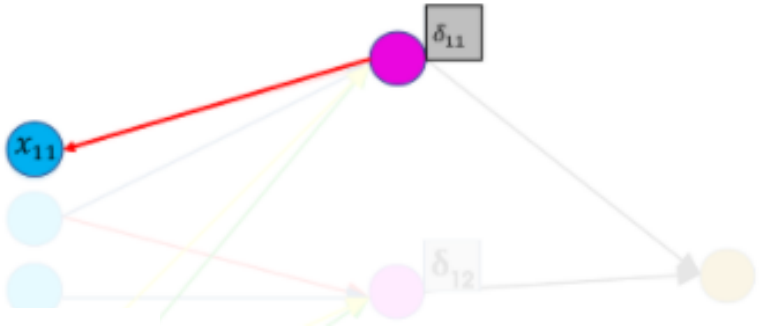


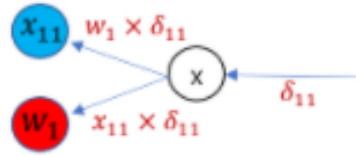
출처 : https://ratsgo.github.io/deep%20learning/2017/04/05/CNNbackprop/

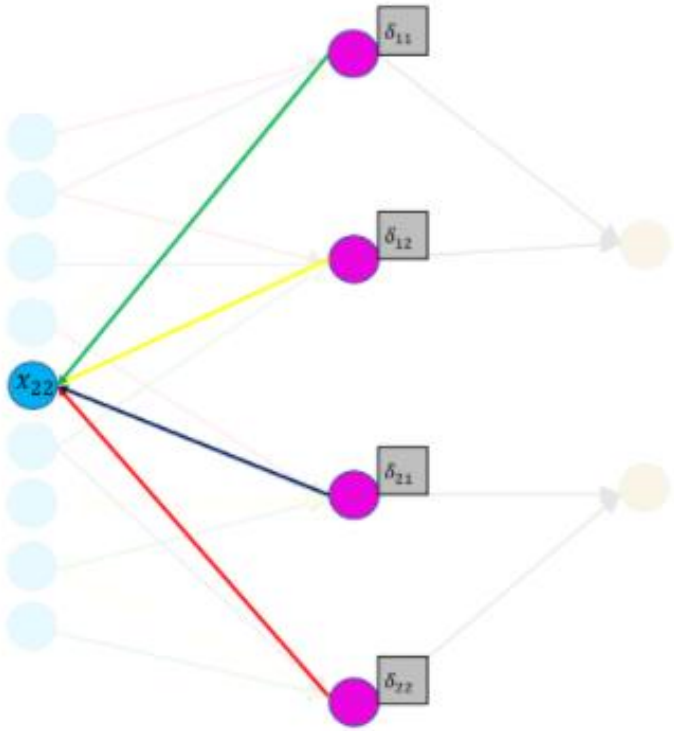
1 CNN (Alexnet)

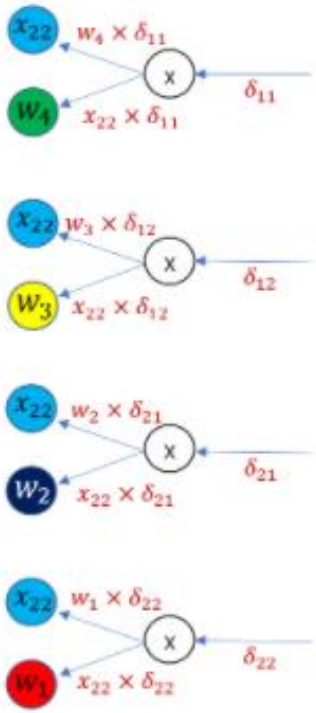
Back propagation – Conv Layer

가중치와 입력값의 상대적 변화량









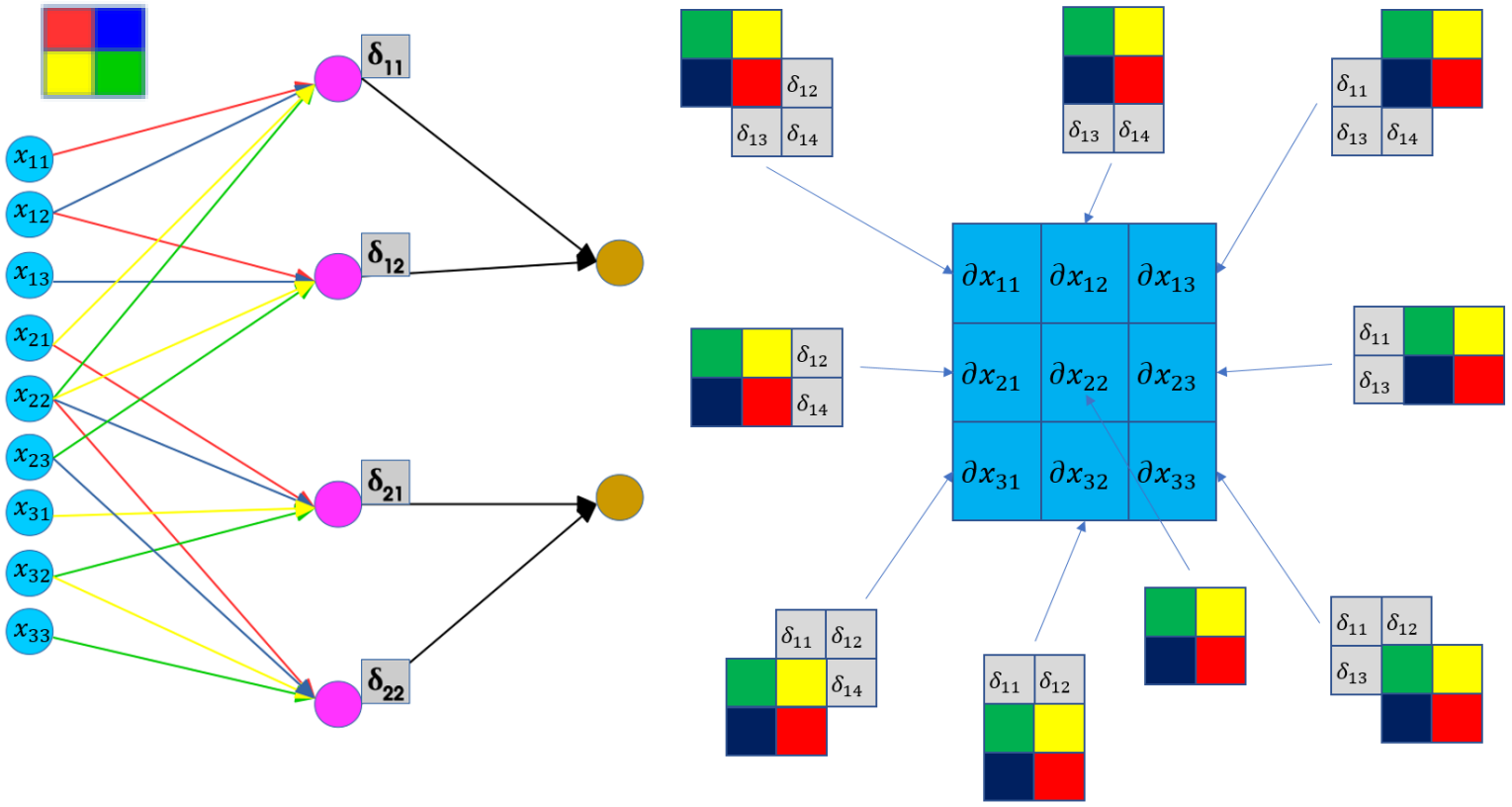
d11이 왔을때 곱의 역전파는 상대방의 값으로 계산됨

출처 : https://ratsgo.github.io/deep%20learning/2017/04/05/CNNbackprop/

1 CNN (Alexnet)

Back propagation – Conv Layer

기존의 Kernel을 뒤집은 커널과 d11을 곱하면 x11의 back prop (dx11)는 w1 * d11으로 계산됨

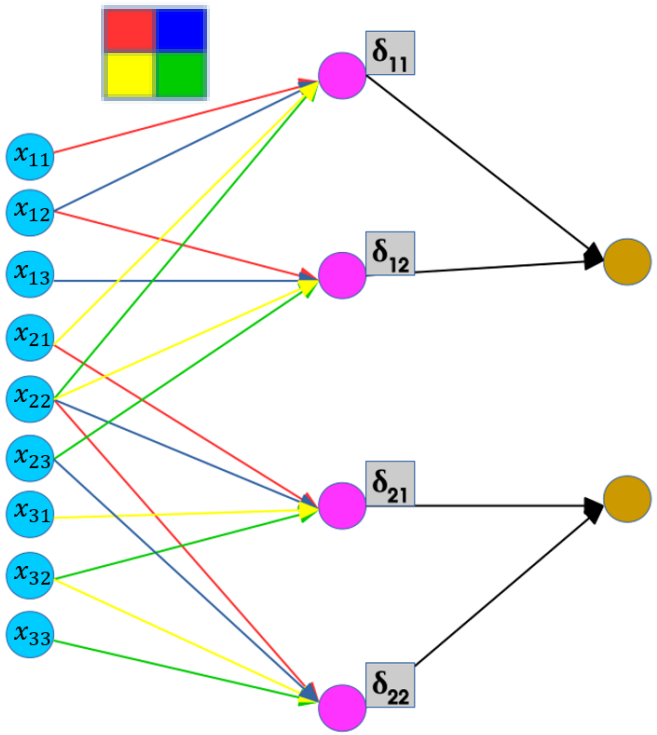


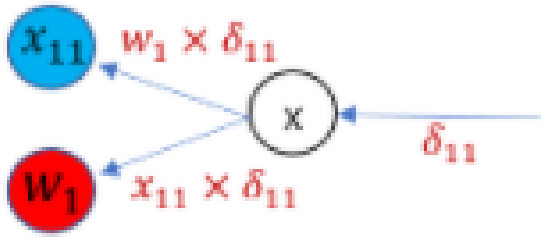
출처 : https://ratsgo.github.io/deep%20learning/2017/04/05/CNNbackprop/

1 CNN (Alexnet)

Back propagation – Filter

필터의 그래디언트 계산은 흘러들어온 그래디언트(d)에 로컬 그래디언트를 곱한 값을 모두 합한 값





dw1 = x11 * d11 + x12 * d12 + x21 * d21 + x22 * d22
(구해진 그래디언트 4개를 모두 합해서 계산)

필터에 대한 업데이트는
w1 = w1 – learning_rate * (dw1)

출처 : https://ratsgo.github.io/deep%20learning/2017/04/05/CNNbackprop/

1 CNN (Alexnet)

CNN의 장점과 단점

[장점]

1. Local Invariance : 입력의 위치가 바뀌어도 출력은 바뀌지 않는 특성 (transition Invariance)
2. 여러개의 Filter를 사용하면 여러개의 Feature map을 얻을 수 있음

[단점]

1. 멀리 있는 픽셀들간의 관계를 보려면 레이어가 깊어야 함 (*Gradient vanishing, exploding의 문제 발생)
2. object의 위치가 중요한 것이 아니기때문에 이러한 정보가 중요한 경우 잘 안먹힐 수 있음
3. 추후 논문에서는 max pooling의 단점을 지적
   - 2와 3을 극복하기 위해서 캡슐 넷(Capsule Net)이 나옴

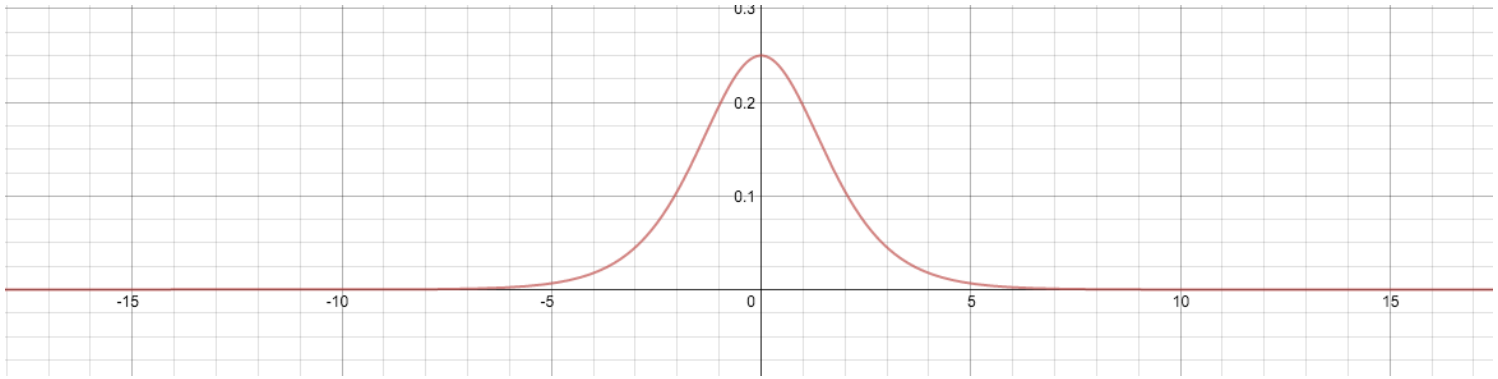
1 용어 정리 - Gradient Vanishing

Gradient Vanishing

Backpropagration을 반복하다보면 미분 값이 어느새 0이 되어서 기울기가 소실되어버리는 현상

예시

Sigmoid에 대한 미분 값은 S(1-S)으로 값의 범위가 0부터 0.25 사이에 위치함

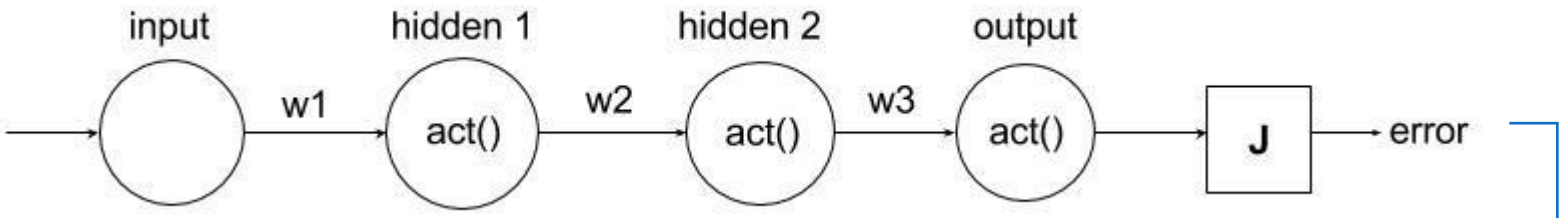


$$\frac{1}{1+e^{-\alpha}}\left[1-\frac{1}{1+e^{-\alpha}}\right]$$

1 용어 정리 - Gradient Vanishing

예시

가중치 w1에 대한 에러의 미분값



$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \boxed{\frac{\partial output}{\partial hidden2}} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

$$z_1 = hidden2 * w3$$

$$\frac{\partial output}{\partial hidden2} = \frac{\partial Sigmoid(z_1)}{\partial z_1} w3$$

1 용어 정리 - Gradient Vanishing

예시

가중치 w1에 대한 에러의 미분값

$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

$$z_2 = hidden1 * w2$$

$$\frac{\partial hidden2}{\partial hidden1} = \frac{\partial Sigmoid(z_2)}{\partial z_2} w2$$

$$\frac{\partial output}{\partial hidden2}\frac{\partial hidden2}{\partial hidden1} = \frac{\partial Sigmoid(z_1)}{\partial z_1} w3 * \frac{\partial Sigmoid(z_2)}{\partial z_2} w2$$

※참고

error에 대한 output의 미분 값은
error = 0.5 * (target – output)^2이 되고
outpu으로 미분시에 target-outpu이 됨

hidden1에 대한 w1의 미분 값은
hidden1 = w1 * input + bias 이므로
w1으로 미분시에 input만 남음

Sigmoid에 대한 미분값이 0.25보다 작거나 같으므로 누적될 수록 최댓값이 작아짐 !!!

1 용어 정리 - Gradient Exploding

Gradient Exploding

Gradient Vanishing과는 반대로 그래디언트가 너무 커져서 학습이 제대로 이뤄지지 않는 현상

- Models loss가 Nan으로 발산함
- loss의 변화량이 매우 커질 것임
- weight의 값이 매우 커지게 됨

1 용어 정리 - Gradient Exploding

Gradient Exploding

1. GSC(Gradient Scale Coefficient) 방법을 통해서 그라디언트 폭발을 측정

f_l : l번째 layer의 출력
θ_l: l번째 layer의 파라미터
J_k^l: f_k 에 대한 f_l 의 자코비안
T_k^l: θ_k 에 대한 f_l 의 자코비안
$||\cdot||$: norm

컨셉 : $||J_k^l||$이나 $||T_k^l||$의 크기가 커지면 l에서 k로 backpropagation이 진행되는 동안에 크기가 커진다고 보는 관점. 하지만, 단순하게 이런 식으로 정의하면 멀쩡한 네트워크도 Gradient Exploding 하다고 판단 할 수 있음

-> 예) 임의의 학습이 잘 되는 네트워크에 $R^{-l}(R < 1)$을 곱해서 학습이 안되도록 방해하면 모델을 학습이 잘 되게 만들기 위해서 그라디언트를 폭발시켜야 함

1 용어 정리 - Gradient Exploding

Gradient Exploding

1. GSC(Gradient Scale Coefficient) 방법을 통해서 그라디언트 폭발을 측정

$$GSC(k, l, f, \theta, x, y) = \frac{||\mathcal{J}_k^l||_{qm}^2 ||f_k||_2^2}{||f_l||_2^2}$$

$$||A||_{qm} = \sqrt{\frac{s_1^2 + s_2^2 + .. + s_{\min(m,n)}^2}{n}}$$

$$||A||_{qm} = \mathbb{Q}_u ||Au||_2$$

m x n 행렬 A의 Quadratic mean norm 은 행렬 A의 singular value들의 제곱 합의 평균에 sqrt를 취한 값. 즉, u를 균등분포를 따르는 임의의 단위벡터라고 하면 "행렬 A가 벡터의 길이에 미치는 영향의 기댓값 "

출처 : http://keunwoochoi.blogspot.com/2018/01/gradients-explode-deep-networks-a

1 용어 정리 - Gradient Exploding

Gradient Exploding

2. GSC(Gradient Scale Coefficient) 가 $c * r^{k-l}$ 보다 크거나 같으면 네트워크가 $f(\theta)$가 r의 속도/비율로 그라디언트 폭발이 일어남 (c와 r에 대한 값이 모호하기에 단순하게 GSC가 지수함수로 근사되면 그라디언트 폭발이 일어난다는 의미)

$$GSC(k, l, f, \theta, x, y) = \frac{||\mathcal{J}_k^l||_{qm}^2 ||f_k||_2^2}{||f_l||_2^2}$$

목적식을 다시 보면 l번째 layer에서 k번째 layer로의 backward pass를 k번째 layer에 대한 l번째 layer의 변화로 곱함

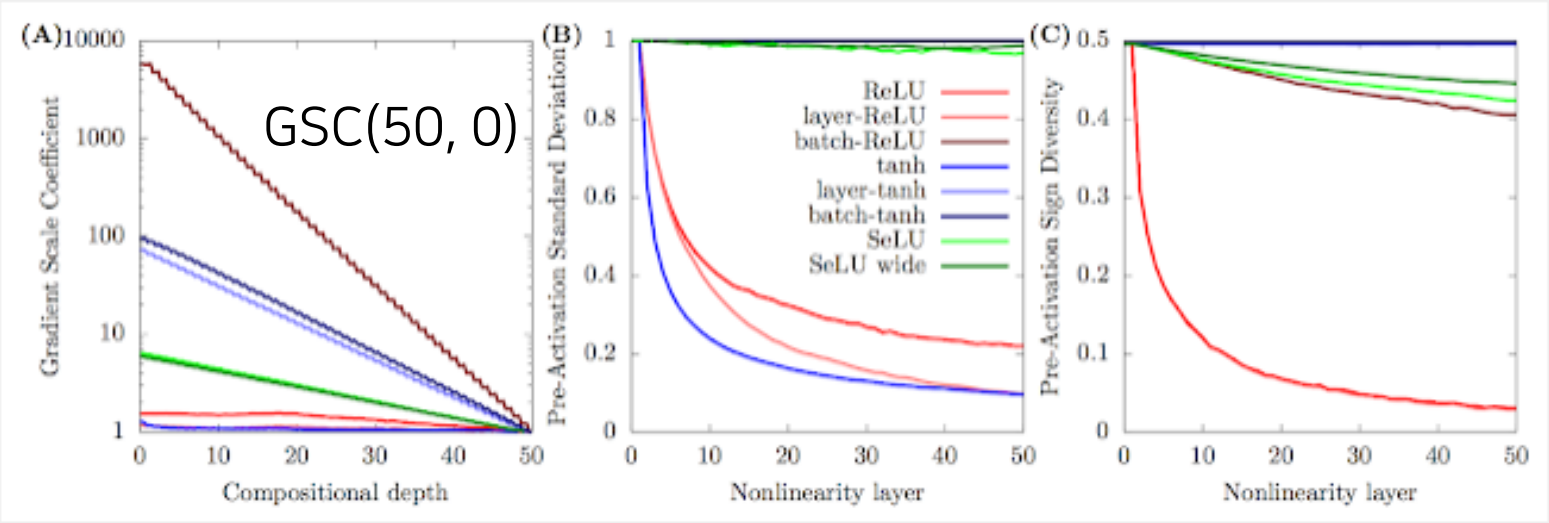
출처 : http://keunwoochoi.blogspot.com/2018/01/gradients-explode-deep-networks-a

1 용어 정리 - Gradient Exploding

그라디언트가 50번째에서 0번째 layer로 진행되면서 혹은 논문 처음에 0번째 레이어를 output layer로 정의해서인지는 모르겠지만, depth가 깊어짐에 따라 GSC가 커지는 것을 의미

Gradient Exploding

3. batch-ReLu, layer-tanh, batch-tanh, SeLU 모두 그라디언트가 log축에서 선형적으로 증가
   - 즉, 그라디언트가 폭발한다는 의미, 그래프에서는 로그를 취했기에 선형의 GSC를 가진다는 것은 지수형태의 모습을 가진다는 의미. (A)는 가중치 초기화와 layer normalization (layer)에 따른 변화



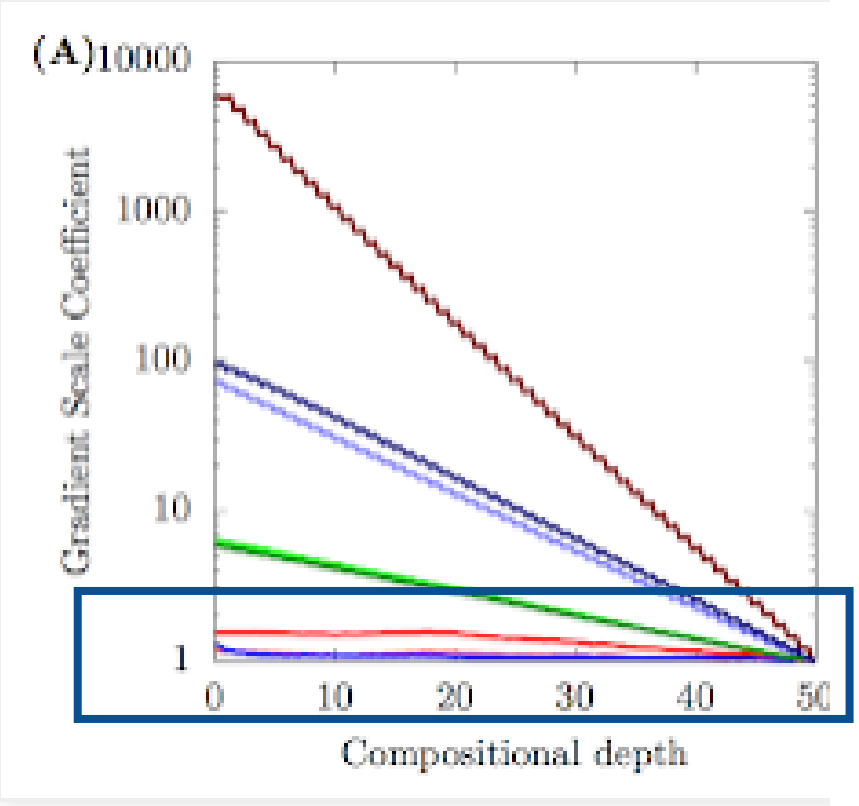
4. 반면 ReLU, layer-ReLU, tanh는 그라디언트 폭발이 일어나지 않았지만 문제가 있음
   - Pre activation SD(Standard Deviation)과 Sign Diversity가 낮음
   - 위는 Pesudo-linearity를 유발(선형함수가 됨)

출처 : http://keunwoochoi.blogspot.com/2018/01/gradients-explode-deep-networks-a

1 용어 정리 - Gradient Exploding

Gradient Exploding

5. GSC는 깊이와 너비와는 무관



1. SeLU에 대해서 depth의 증가에 따라
GSC가 거의 변화하지 않는 것을 볼수가 있음

2. 뉴런이 100개와 200개간의 SeLU 차이가 거의 없는 것을 볼 수 있음

위의 두가지를 근거로 깊이와 너비와는 무관하다고 주장 !!!

출처 : http://keunwoochoi.blogspot.com/2018/01/gradients-explode-deep-networks-a

1 용어 정리 - Gradient Exploding

Gradient Exploding

해결책

1. Layers의 수를 줄이기 (위의 논문에서는 층을 쌓는 것보다 너비를 증가하는 것을 추천)
2. Batch Normalization, Layer Normalization
3. Weight Initialization
4. Residual connections

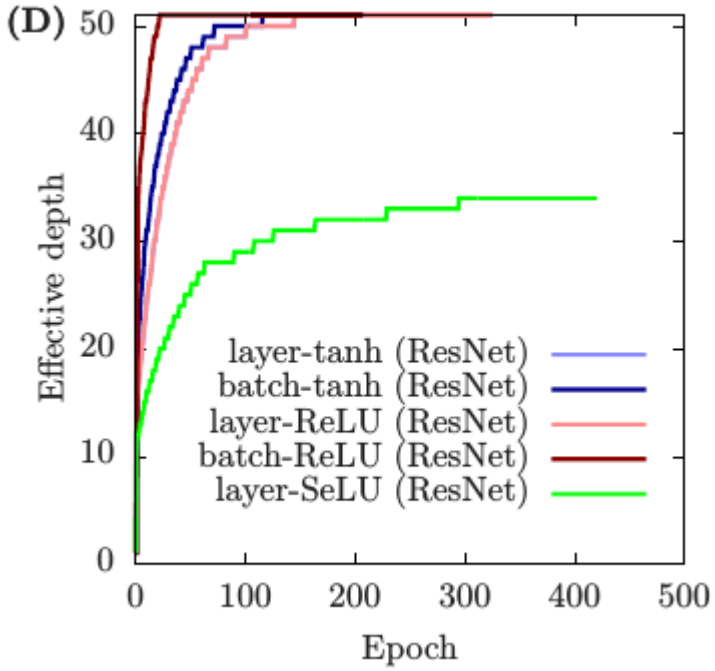
하지만, 2와 3은 완벽한 해결책은 아니고 어느정도 도움을 줄 뿐입니다. 실제로 3번 그래프에서 봤듯이 1, 2, 3은 큰 상관은 없고 어느정도 도움만 주는 것을 알 수 있습니다.

출처 : http://keunwoochoi.blogspot.com/2018/01/gradients-explode-deep-networks-a

1 용어 정리 - Gradient Exploding

Gradient Exploding

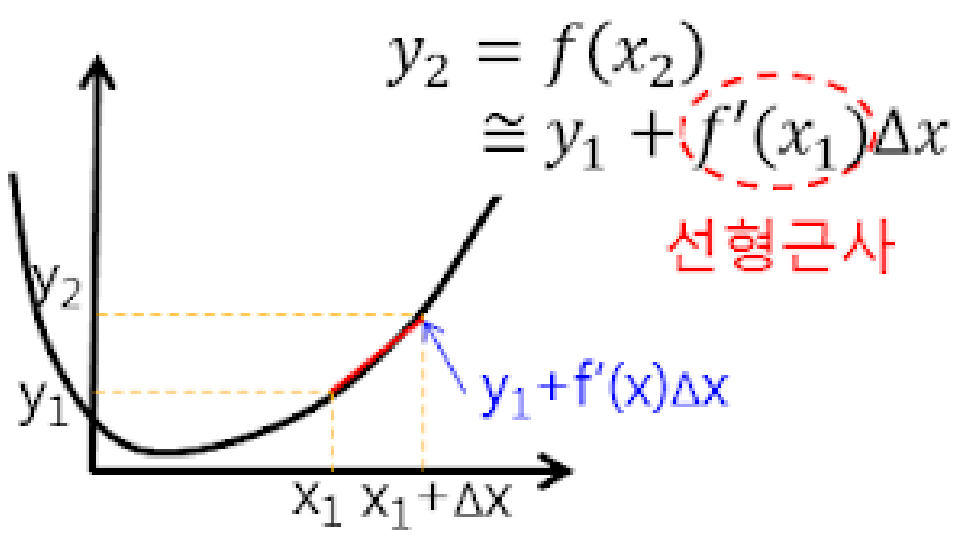
Resnet을 적용시에 그렇지 않은 경우 보다 Effective Depth가 깊음



1 용어 정리 - Jacobian

Jacobian

복잡하게 얽혀있는 식을 미분을 통해 linear approximation 시킴으로서 간단한 근사 선형식을 만들어 주는 것.
다변수 함수인 경우에는 다변수 함수일 때의 미분값으로 정의



$$\frac{d\boldsymbol{y}}{d\boldsymbol{q}} = \frac{f(\boldsymbol{q})}{d\boldsymbol{q}} = \begin{pmatrix} \frac{df_1}{dq_1} & \cdots & \frac{df_1}{dq_n} \\ \vdots & \ddots & \vdots \\ \frac{df_m}{dq_1} & \cdots & \frac{df_m}{dq_n} \end{pmatrix} \equiv J \ (Jacobian)$$

$$\therefore d\boldsymbol{y} = J d\boldsymbol{q}$$

$$\boldsymbol{y_1} + f' d\boldsymbol{q} = \boldsymbol{y_1} + J d\boldsymbol{q}$$

출처 : http://t-robotics.blogspot.com/2013/12/jacobian.html#.X0OwjcgzaUk

1 용어 정리 - Jacobian

Jacobian 행렬의 의미

1. 주어진 비선형 변환이 매우 국소적으로는 선형 변환처럼 취급할 수 있는데, 이 '선형 변환 ' 에 해당되는 **행렬**은 무엇인가?
   - 이때의 행렬을 자코비안 행렬이라고 표현

2. 각 포인트 마다 이 행렬을 나타내면? (변수를 도입해서 표현)
   - (u, v) -> (x, y)
   - 이때의 넓이는 du * dv 가 dx * dy으로 선형변환 J에 의해서 바뀌는 것이고, 식으로 표현하면 dx*dy=|J|*du*dv 으로 표현할 수 있음 !!! (선형변환은 기하학적으로 넓이를 변화시킴)

$$J = \begin{pmatrix} x_u & x_v \\ y_u & y_v \end{pmatrix}$$

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = J \begin{bmatrix} du \\ dv \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} \qquad J = \begin{bmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{bmatrix}$$

$$dx = a \times du + b \times dv$$

$$dy = c \times du + d \times dv$$

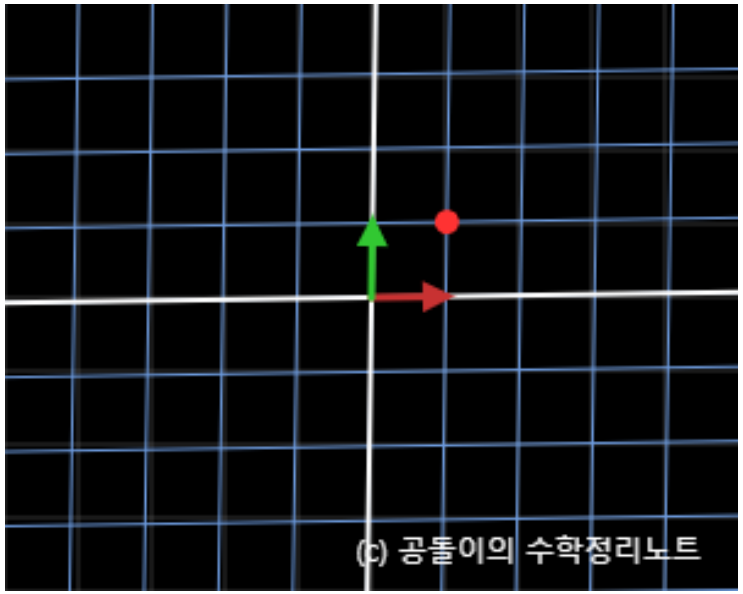
출처 : https://angeloyeo.github.io/2020/07/24/Jacobian.html

1 용어 정리 - Jacobian

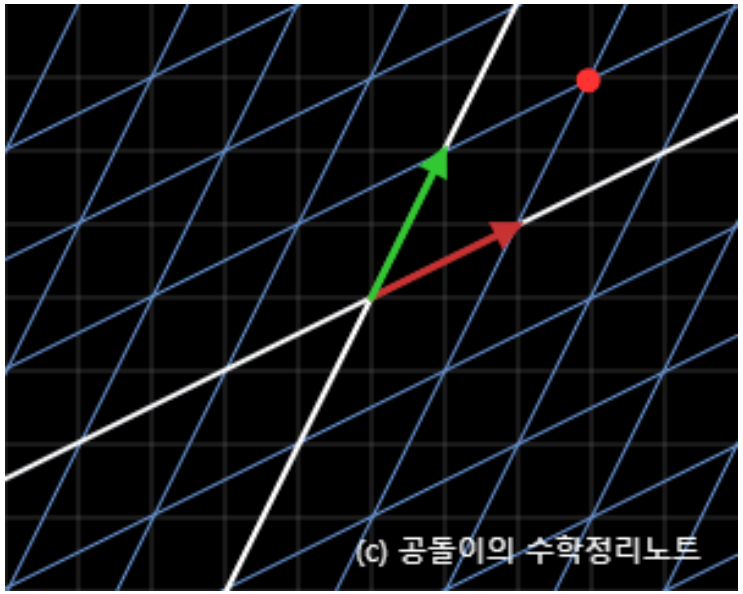
선형변환

1. 변환 후에도 원점의 위치가 변하지 않는다.
2. 변환 후에도 격자들의 형태가 직선의 형태를 유지한다.
3. 격자간의 간격이 균등한다.
4. 기하학적으로는 넓이가 1이 ad-bc만큼으로 변함

[변환전]



[변환후]

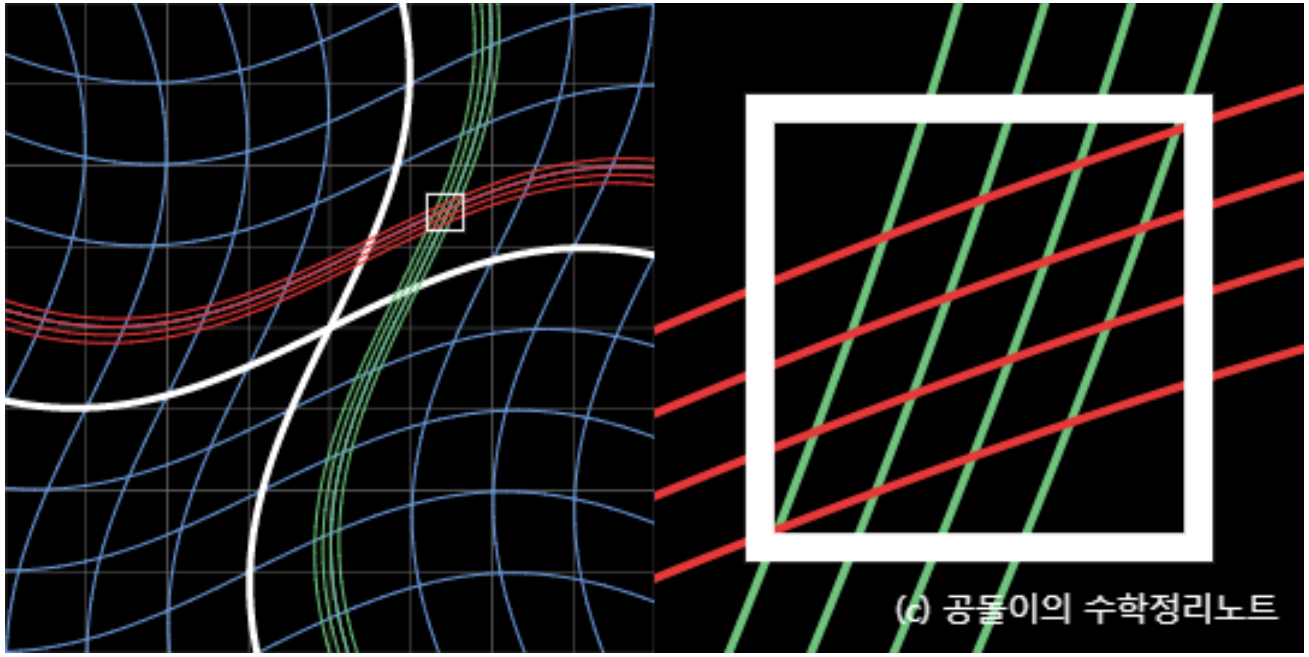


출처 : https://angeloyeo.github.io/2020/07/24/Jacobian.html

1 용어 정리 - Jacobian

자코비안의 비선형 변환 결과

흰색의 국소적인 공간에서는 선형 변환이 일어남



출처 : https://angeloyeo.github.io/2020/07/24/Jacobian.html

02 CNN Parameter

2 CNN Paramets

Convolution 레이어 출력 데이터 크기 산정

입력 데이터에 대한 필터의 크기와 strid의 크기에 따라서 Feature Map의 크기가 결정

- 입력 데이터 높이: H
- 입력 데이터 폭: W
- 필터 높이: FH
- 필터 폭: FW
- Stride 크기: S
- 패딩 사이즈: P

$$OutputHeight = OH = \frac{(H + 2P - FH)}{S} + 1$$

$$OutputWeight = OW = \frac{(W + 2P - FW)}{S} + 1$$

단, *output*의 *Height*와 *Weight*는 정수이어야함

출처 : http://taewan.kim/post/cnn/

2 CNN Paramets

Pooling 레이어 출력 데이터 크기 산정

Pooling 사이즈를 Stride 같은 크기로 만들어서, 모든 요소가 한번씩 Pooling 되도록 설정
- Output의 크기는 Pooling Size로 나눈 값

$$OutputRowSize = \frac{InputRowSize}{PoolingSize}$$

$$OutputColumnSize = \frac{InputColumnSize}{PoolingSize}$$

출처 : http://taewan.kim/post/cnn/

2 CNN Paramets

Flatten 레이어 출력 데이터 크기 산정

CNN의 데이터 타입을 Fully Connected 형태로 변경하는 레이어 (파라미터가 존재하지 않고, 입력 데이터의 shape 변경만 수행)

- 입력 데이터 Shape =(2, 1, 80)
- 출력 데이터 Shape =(160, 1)

출처 : http://taewan.kim/post/cnn/

2 CNN Paramets

Softmax 레이어 출력 데이터 크기 산정

네트워크의 분류 클래스의 수에 따라 변경

- 입력 데이터의 shape : (160, 1)
- 출력 데이터의 shape : (10, 1)

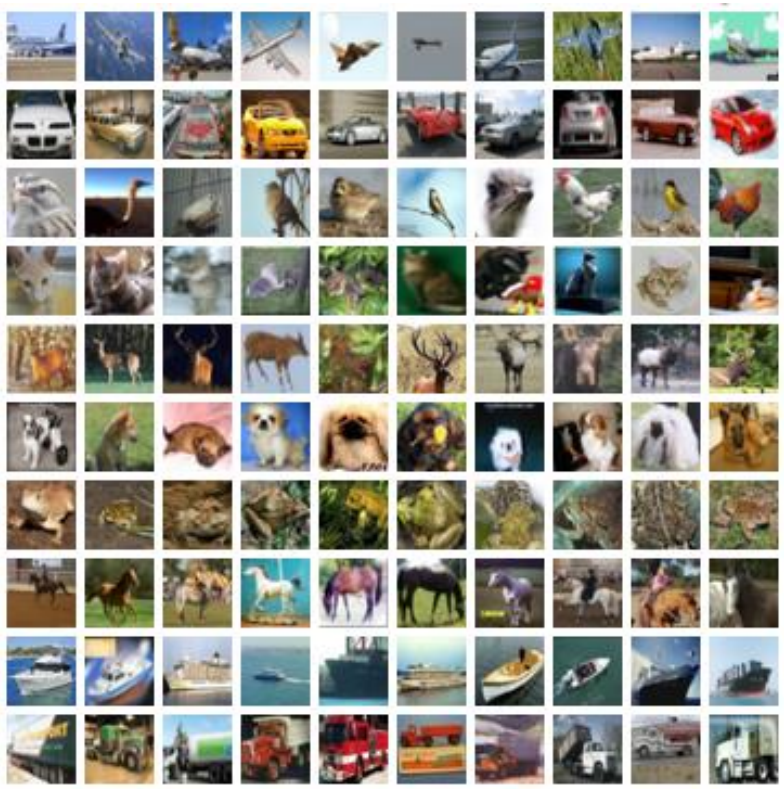
출처 : http://taewan.kim/post/cnn/

03 실습 (Pytorch)

3 CNN (Alexnet) – Pytorch 구현

데이터

CIFAR-10 데이터를 이용
링크 : https://www.kaggle.com/c/cifar-10



60000개의 32x32 color images를 10개의 object class로 분류

3 CNN (Alexnet) – Pytorch 구현

구조 1개 실험

Layer Type	CIFAR-10
Convolution + ReLU	$3 \times 3 \times 64$
Convolution + ReLU	$3 \times 3 \times 64$
Max Pooling	2×2
Convolution + ReLU	$3 \times 3 \times 128$
Convolution + ReLU	$3 \times 3 \times 128$
Max Pooling	2×2
Flatten	
Fully Connected + ReLU	256
Dropout	0.5
Fully Connected + ReLU	256
Fully Connected	10

https://www.researchgate.net/publication/326816043_FAWCA_A_Flexible-greedy_Approach_to_find_Well-tuned_CNN_Architecture_for_Image_Recognition_Problem

3 CNN (Alexnet) – Pytorch 구현

코드

깃허브 링크 : https://github.com/Kaist-Master/Image-analysis-with-Deep-Learning/tree/master/002.%20Alexnet%20(CNN)/code

실험 목록

- v1 : 모든 Train을 학습에 사용해서 predict
- v2 : KFold 방법을 이용해서 Ensemble
- v3 : pytorch에서 유용하게 사용하는 utilization tool을 추가

4 참고 자료

[CNN]

1. https://medium.com/@seoilgun/cnn%EC%9D%98-stationarity%EC%99%80-locality-610166700979
2. https://www.slideshare.net/agdatalab/deep-learning-convolutional-neural-network
3. http://aikorea.org/cs231n/convolutional-networks/
4. https://www.researchgate.net/publication/326816043_FAWCA_A_Flexible-greedy_Approach_to_find_Well-tuned_CNN_Architecture_for_Image_Recognition_Problem
5. https://www.researchgate.net/figure/Model-architectures-for-the-MNIST-and-CIFAR-10-models_tbl10_324558570
6. https://ratsgo.github.io/deep%20learning/2017/04/05/CNNbackprop/
7. https://taewan.kim/post/cnn/
8. https://zzsza.github.io/data/2018/05/14/cs231n-cnn/

4 참고 자료

[Gradient Vanishing & Exploding]

1. https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b
2. http://keunwoochoi.blogspot.com/2018/01/gradients-explode-deep-networks-are.html

[Jaccobian]

1. http://t-robotics.blogspot.com/2013/12/jacobian.html#.X0OwjcgzaUk
2. https://angeloyeo.github.io/2020/07/24/Jacobian.html

감사합니다