DATA
SCIENCE LAB

CNN 1

23.02.14 / 8기 정건우

0. INTRO

CNN1

Computer Vision이란? CNN은 왜 등장했을까? CNN 기본 개념

Convolutional Layer / Padding, Stride / Batch Normalization, …

CNN2

지금까지는 **어떤 CNN 모델**들이 있었지? CNN 모델의 발전과정

CNN Architecture / Comparing Model Complexity

CNN3

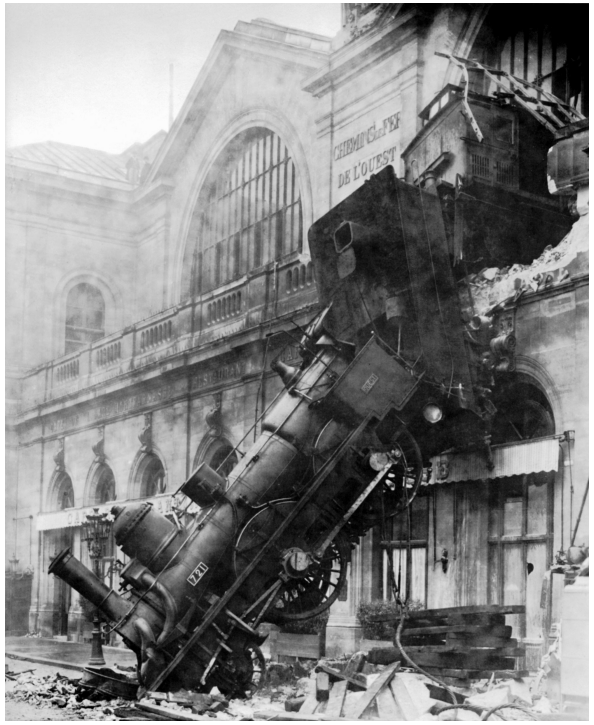
그래서 **성능 좋은 (CNN) 모델**을 만들려면 어떻게 해야 되는데?

CNN을 이용하는 이유 / **전이학습(Transfer Learning)**

CONTENTS

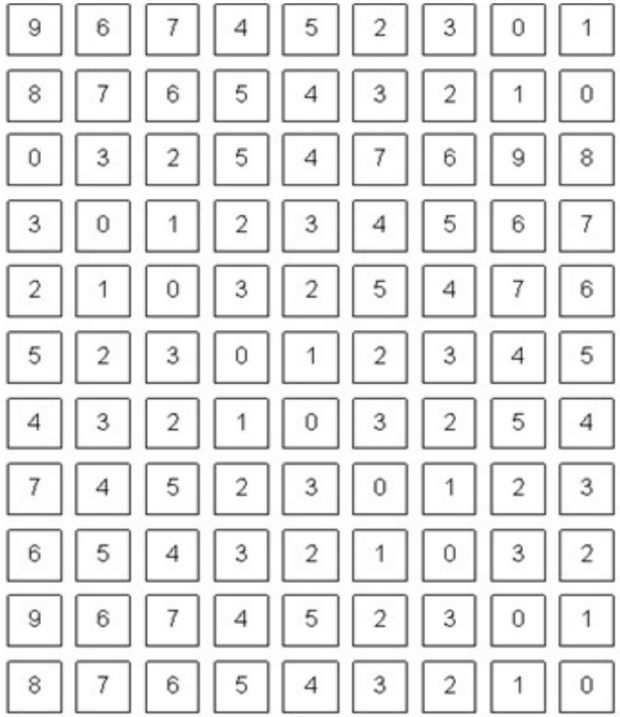
1. Computer Vision

What is CV ?



What we see

" 기차가 무너져있네?"
인식할 수 있다



What a computer sees

컴퓨터는 단순 pixel 값을 본다

[Computer Vision의 목표]

단순 pixel 값(0~255)에서
정보를 추출하자 !

↓

Image Classification

Object Detection

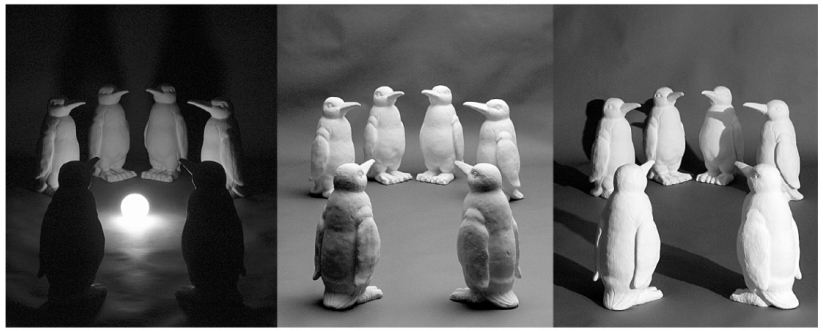
Image Captioning

...

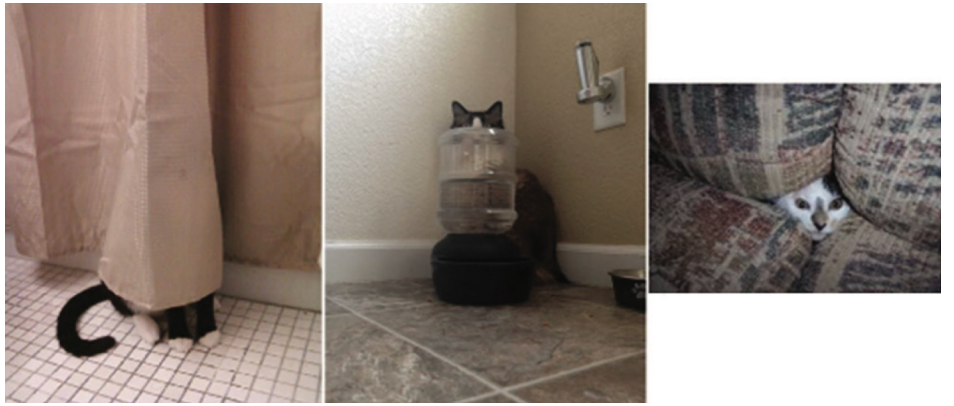
1. Computer Vision

Why is CV difficult? = So many variation !

Viewpoint



illumination

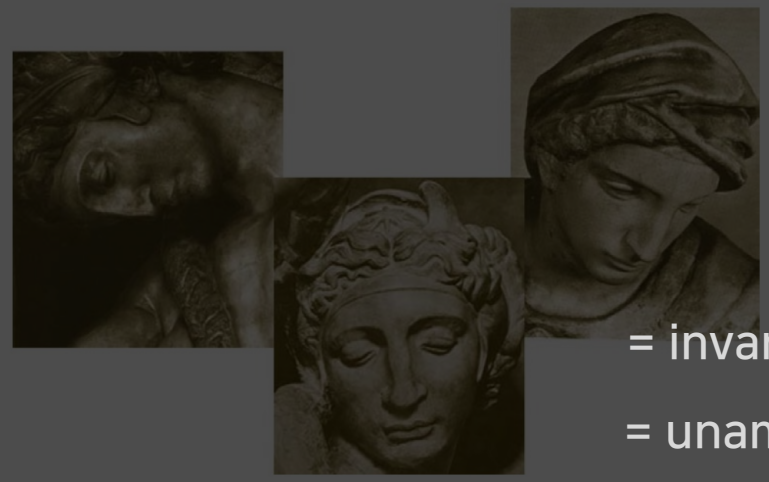


Occlusion

Intra-class variation

1. Computer Vision

Why is CV difficult? = So many variation !



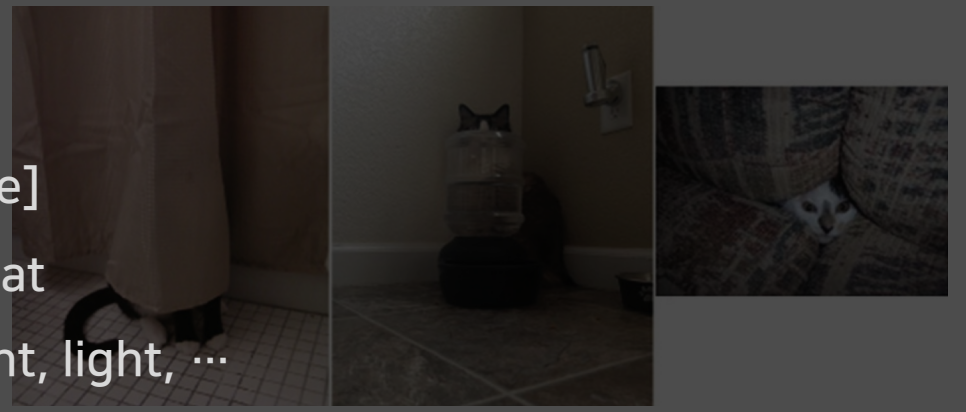
Viewpoint

[Good Feature]

= should repeat

= invariant to viewpoint, light, ···

= unambiguous for classification



Occlusion

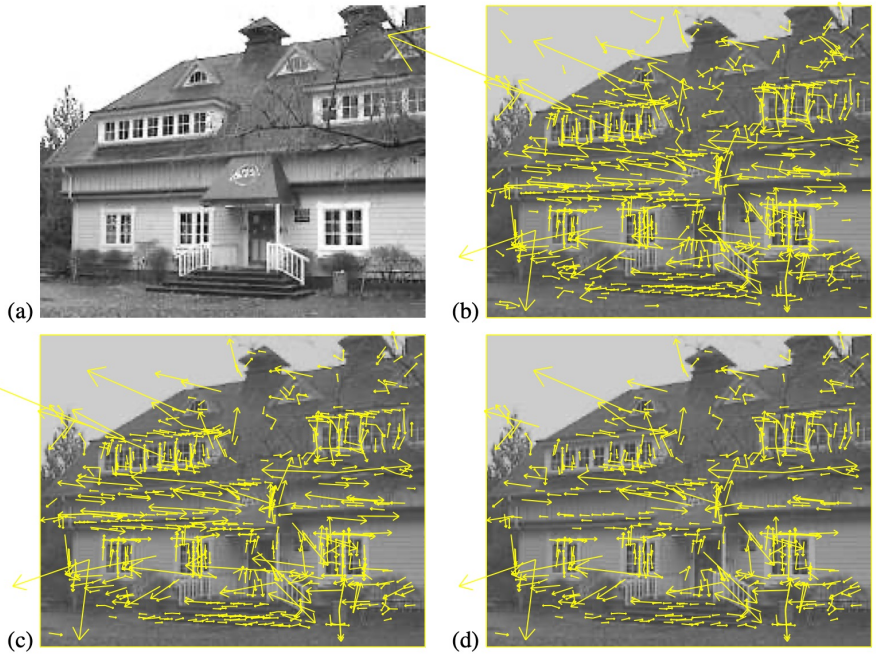
illumination

Intra-class variation

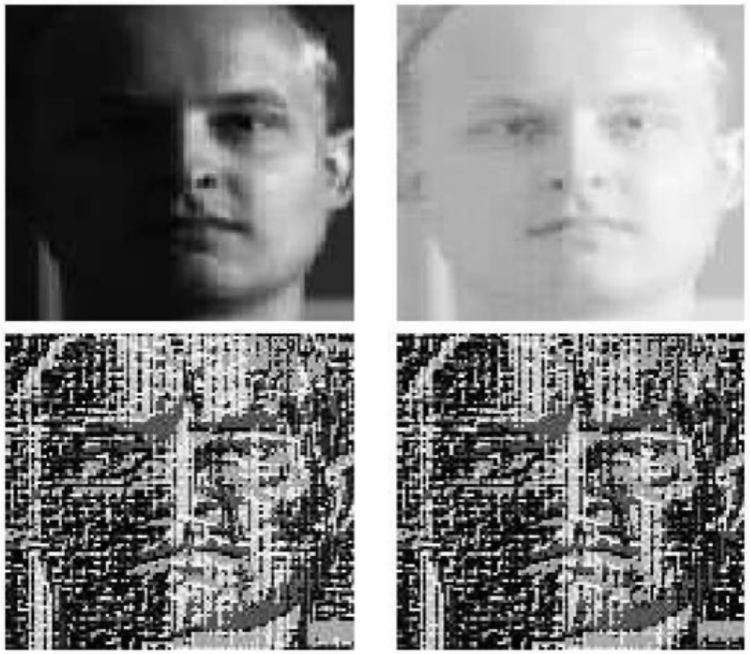
1. Computer Vision

Feature Extraction

SIFT (Scale-Invariant Feature Transform)



MCT (Modified Census Transform)



Haar

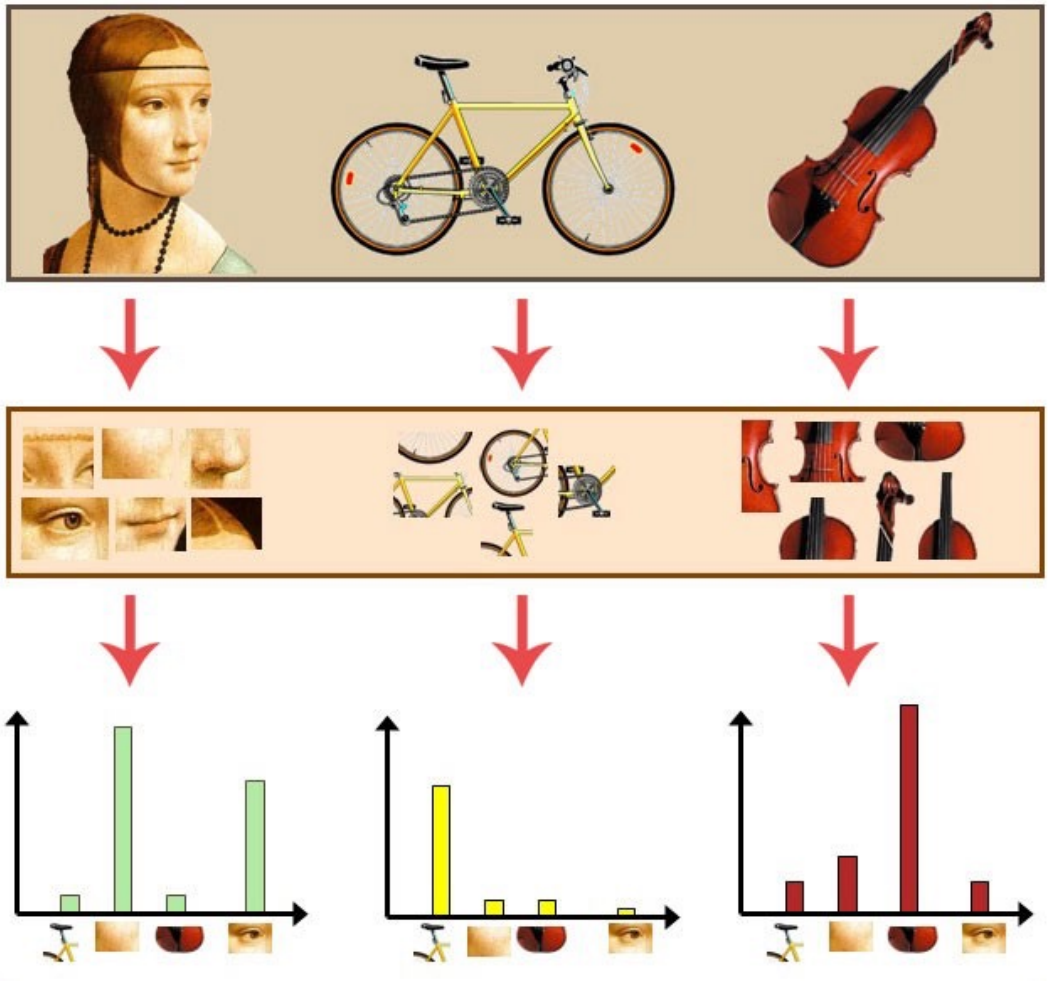
Ferns

LBP

...

1. Computer Vision

Feature Extraction



[Bag of Features]

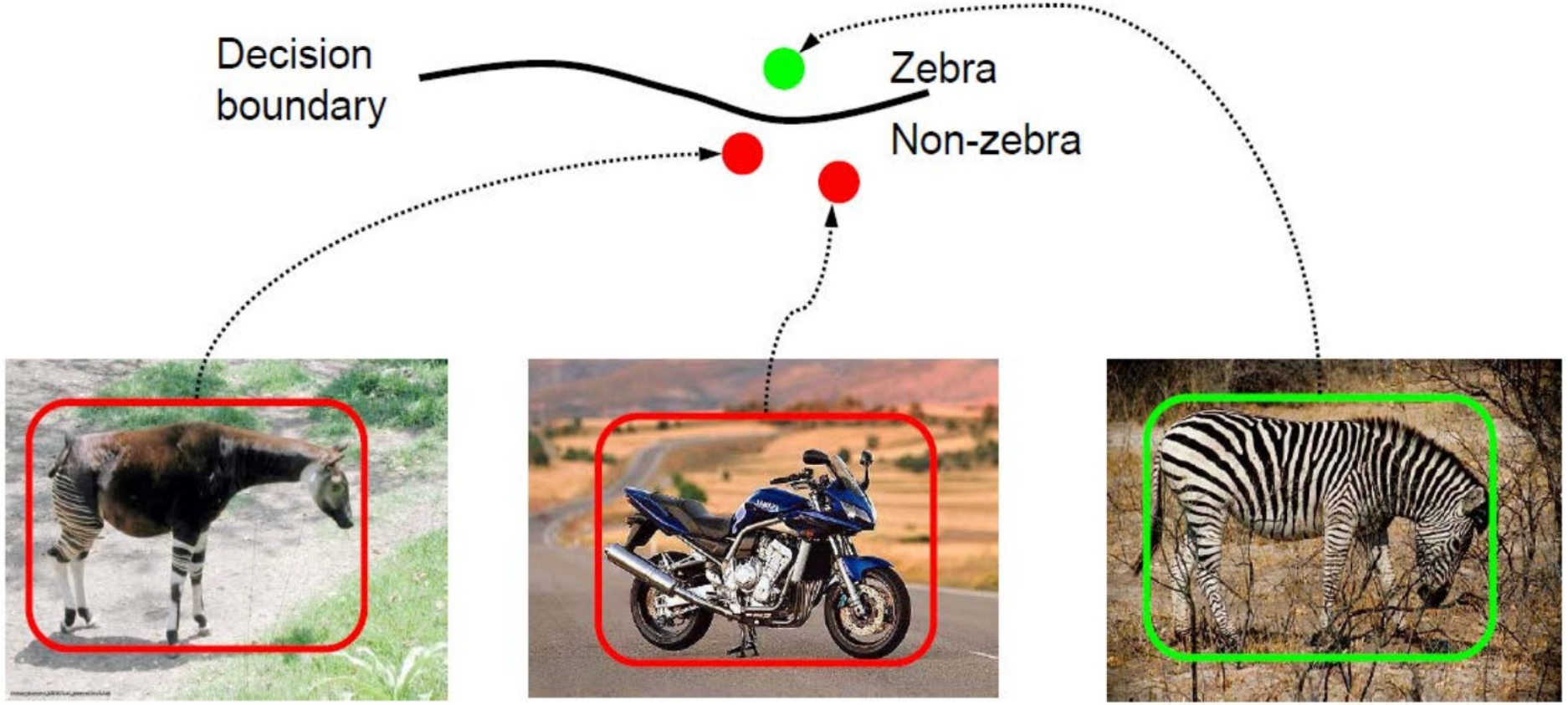
Image가 각 feature를 얼마나 많이 가지는지 계산

(Image Representation)

→ 4-dimentional vector

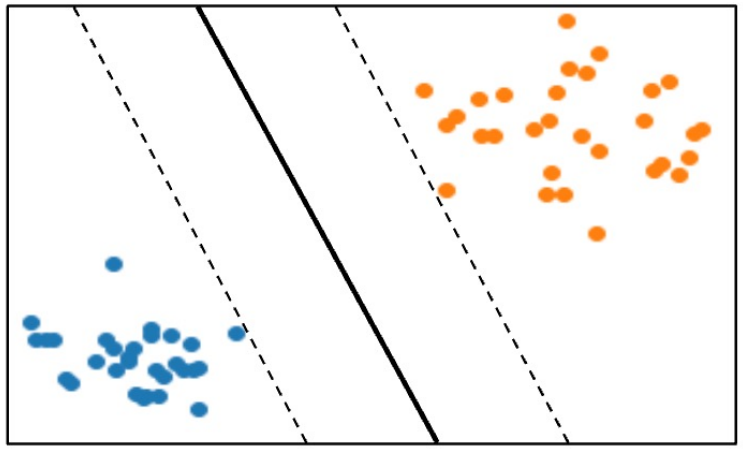
1. Computer Vision

Classification



1. Computer Vision

Classification



Dicision Boundary? - SVM !

= Finding best decision boundary

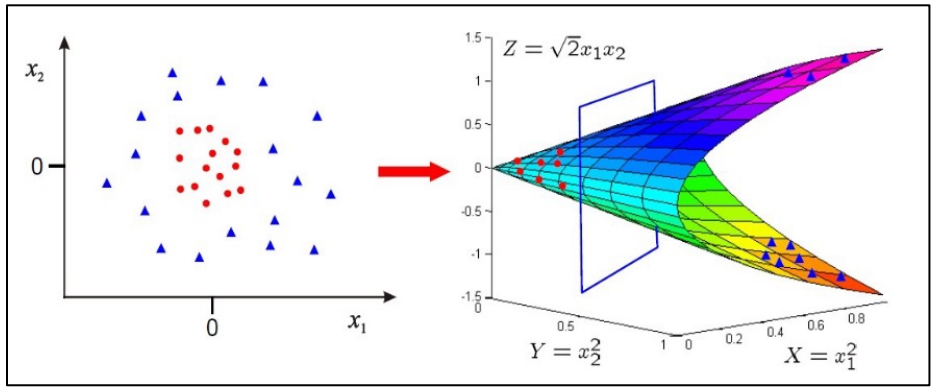
= Learning(=Training) (SVM) Model

= Finding best hyperplane

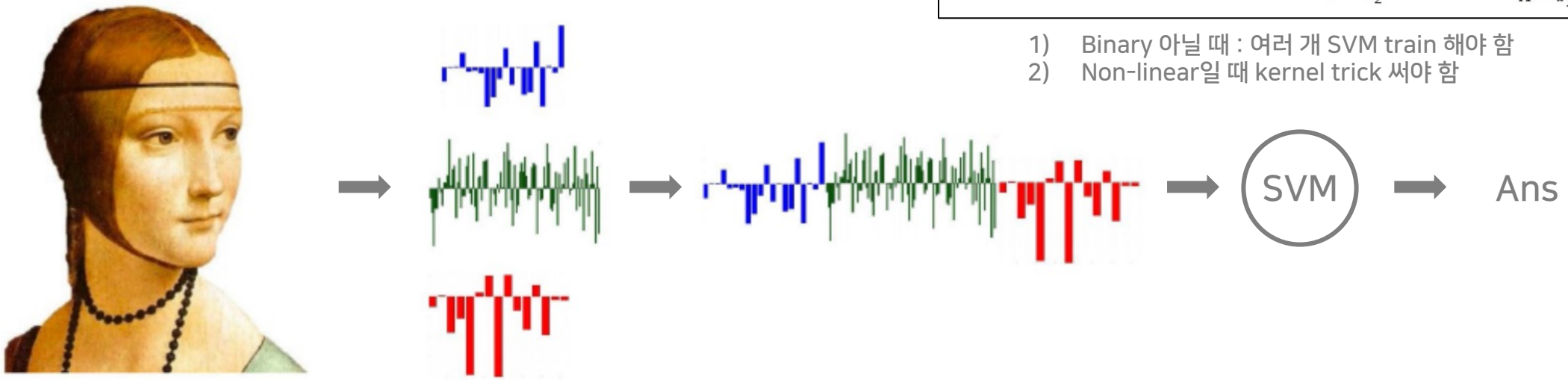
→ Image Classification

1. Computer Vision

Overall Process



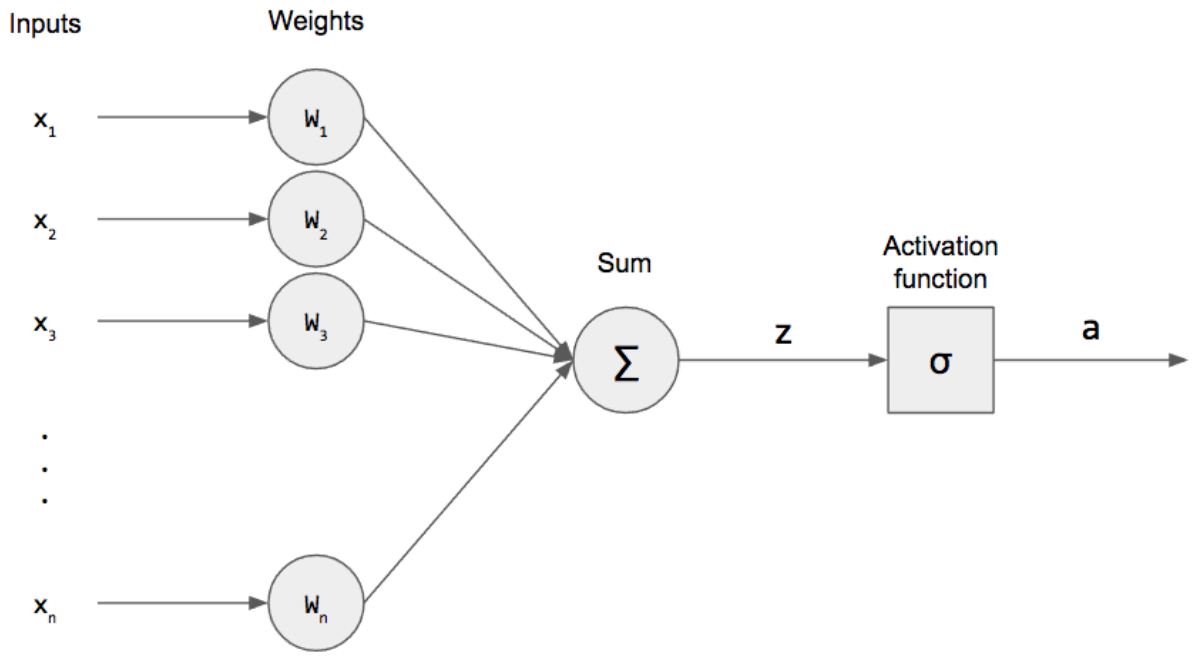
1) Binary 아닐 때 : 여러 개 SVM train 해야 함
2) Non-linear일 때 kernel trick 써야 함



Input Image (pixel) → Extract features → Concatenate into a vector → Classification

2. Neural Network

Multi-layer Perceptron

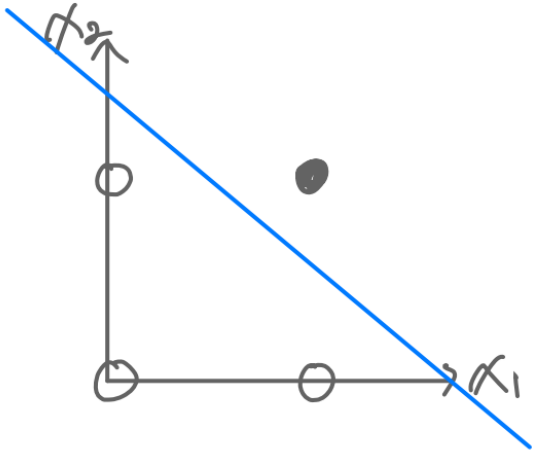


Single-Layer Perceptron

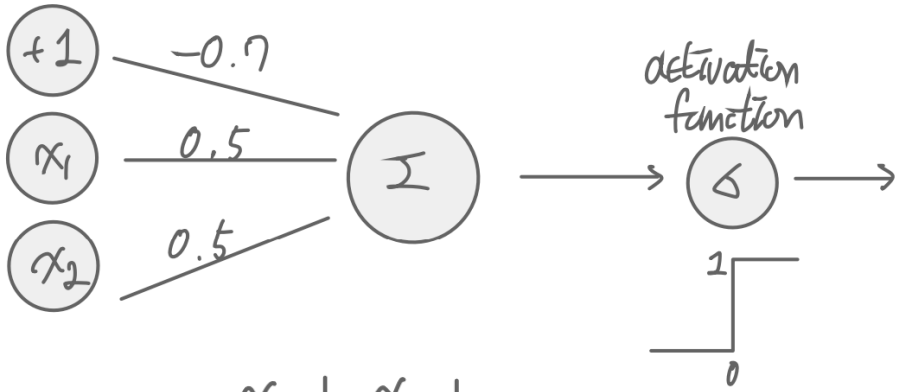
2. Neural Network

Multi-layer Perceptron

Linear Classification



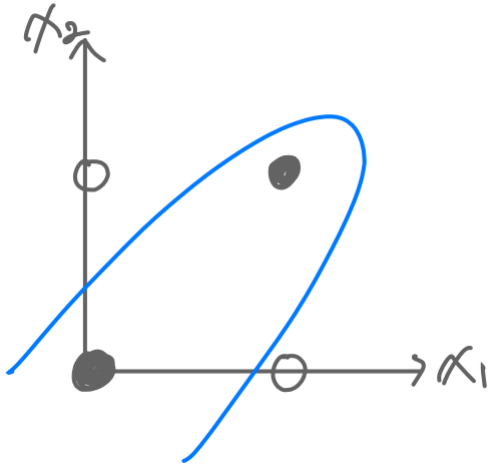
x1 AND x2



x_1	x_2	
0	0	0
1	0	0
0	1	0
1	1	1

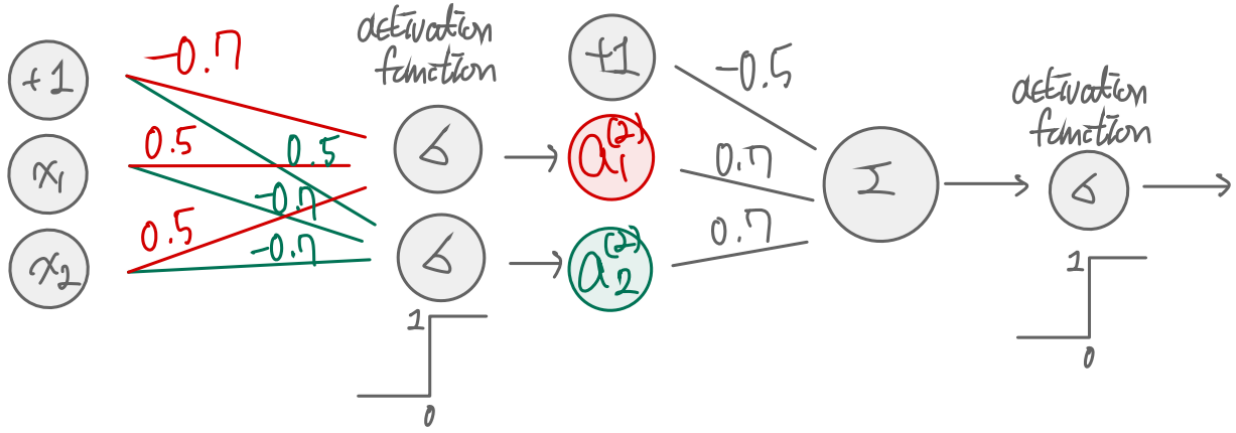
2. Neural Network

Multi-layer Perceptron



XNOR gate

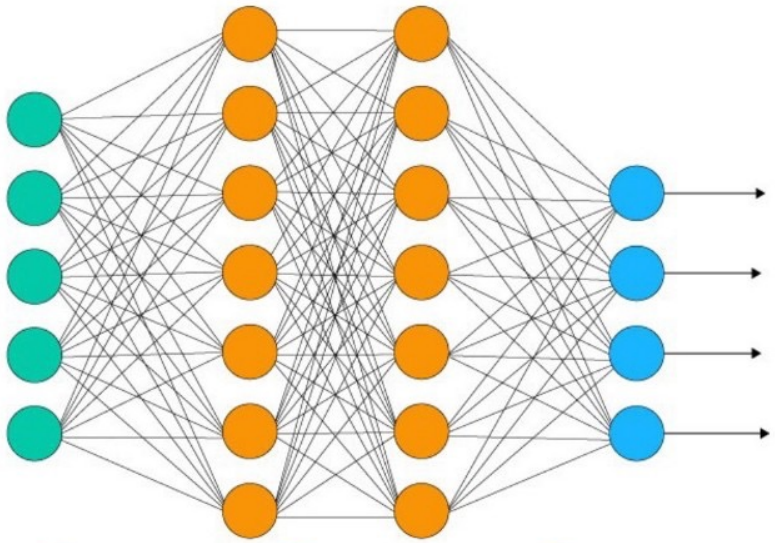
Non-linear Classification



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	
0	0	0	1	1
1	0	0	0	0
0	1	0	0	0
1	1	1	0	1

2. Neural Network

Multi-layer Perceptron



Fully Connected Layer

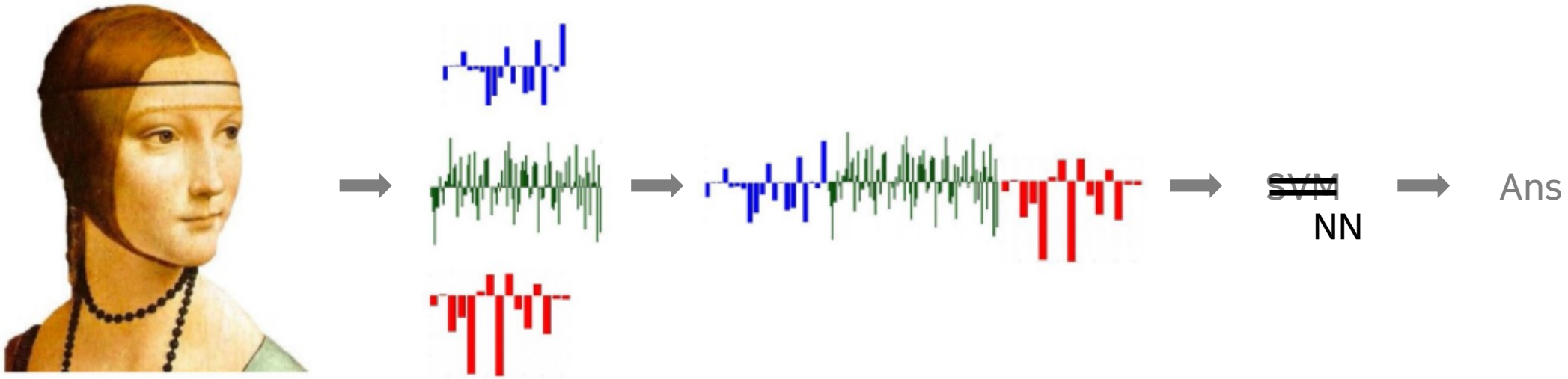
[NN is better classifier than SVM]

Feature만 잘 뽑아서 input 한 뒤,
Activation function 잘 설정하고
weight train 잘 하면 (SGD, Adam, …)

SVM보다 효율적으로
Linear, Non-linear classification
Multi-class classification
수행할 수 있다 !

2. Neural Network

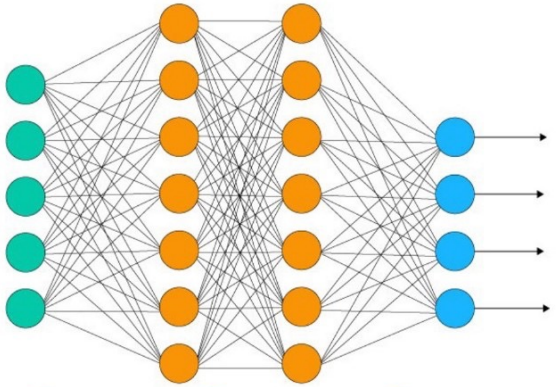
Multi-layer Perceptron



2. Neural Network

Kernel = Filter

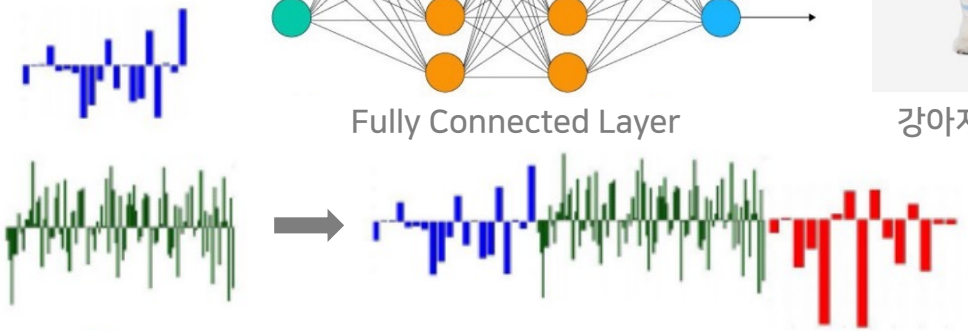
Overfitting



Fully Connected Layer

강아지라고 학습했는데

강아지가 아니라고 판단

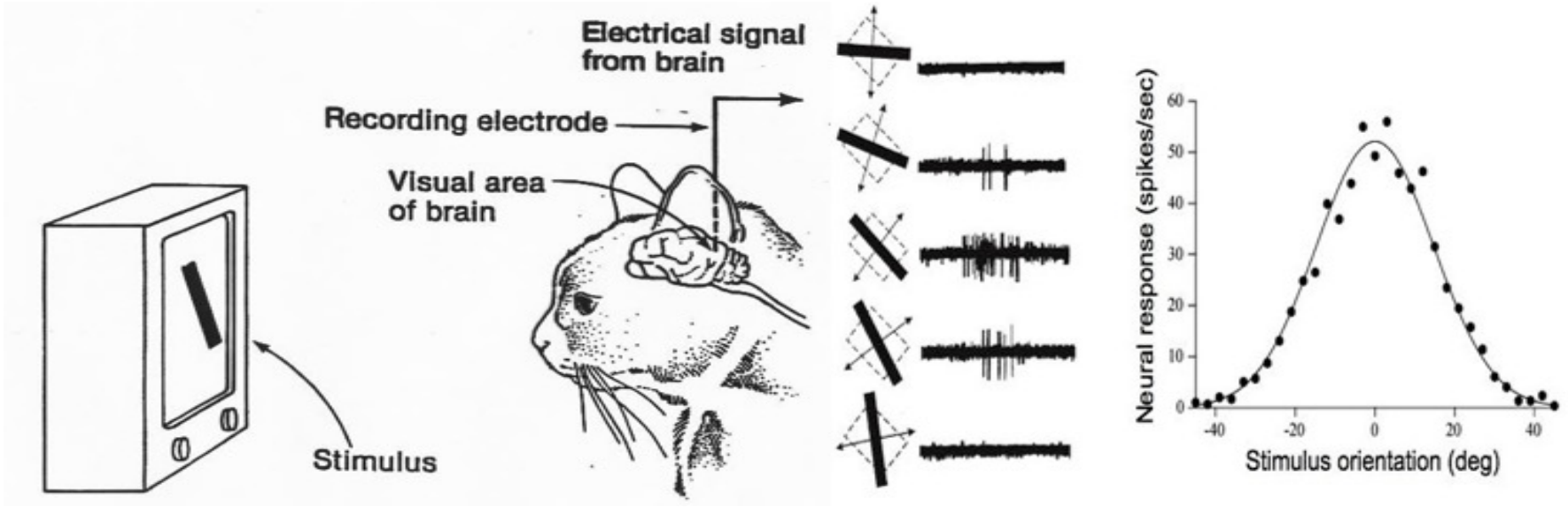


SIFT, MCT, ···

Input Image (pixel) → Extract features → Concatenate into a vector → Classification → ~~SVM~~ NN → Ans

2. Neural Network

Kernel = Filter



고양이 시야의 한 쪽에 자극을 주었더니 전체 뉴런이 아닌 특정 뉴런만이 활성화

→ Fully Connected Layer 말고 Convolutional Layer 쓰자 !

2. Neural Network

Kernel = Filter

[Convolution]

= calculation of accumulated reaction

$$m[n] * f[n] = \sum_{k=-\infty}^{\infty} m[k] f[n-k]$$

$$= \sum_{k=-\infty}^{\infty} m[k] f[-(k-n)]$$

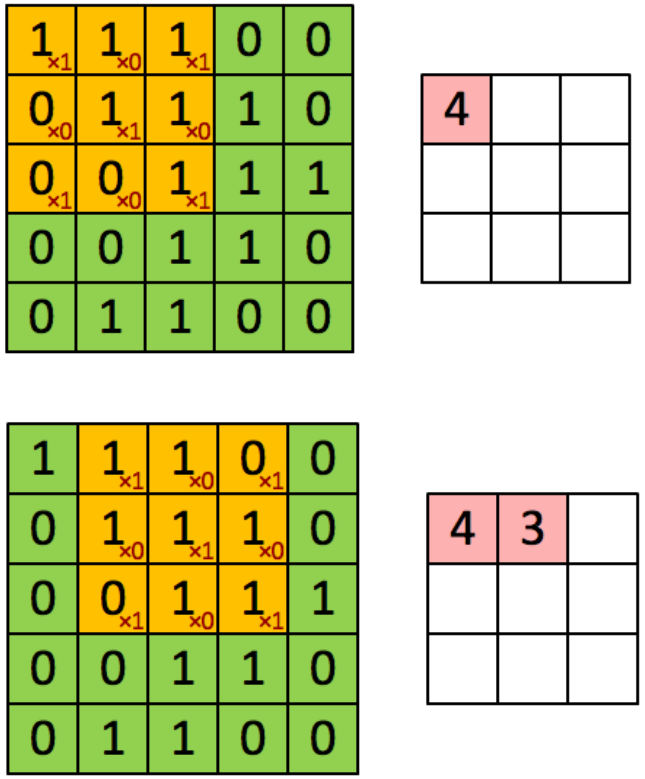
CNN에서는,

이미지를 훑어보면서

현재 위치에서 지금 filter로 구하고 있는

feature의 정도가 얼마나 되는지 계산

→ 행렬 간의 내적 연산



2. Neural Network

Kernel = Filter

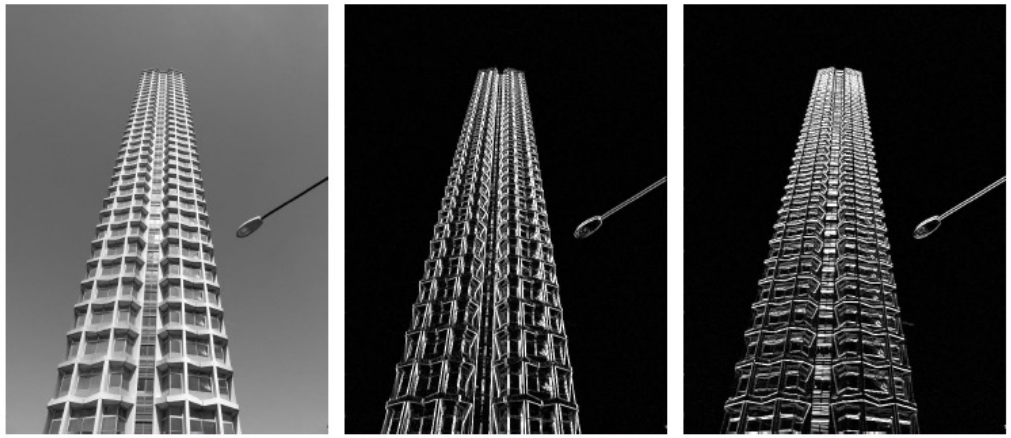
Ex. Sobel filter

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

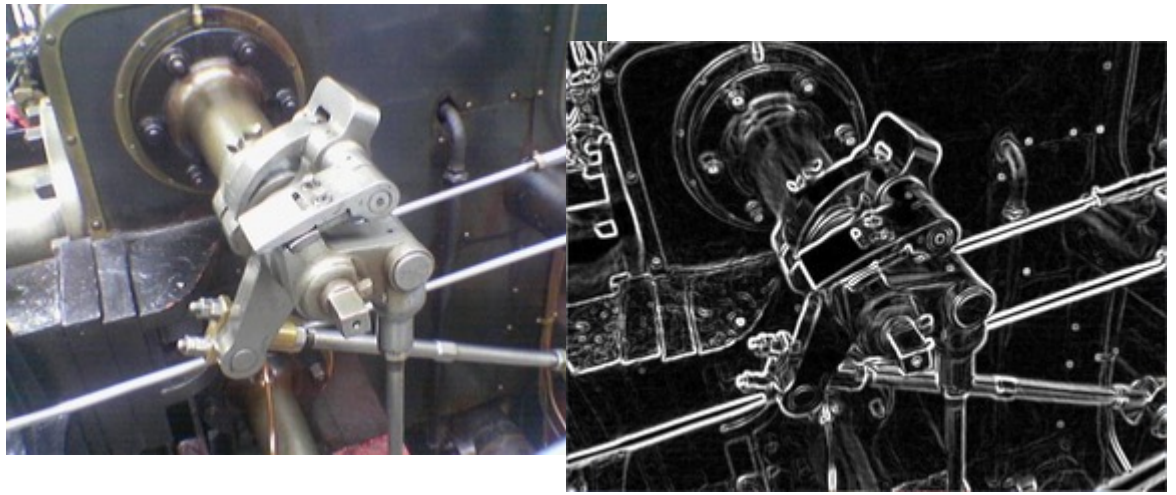
x filter

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

y filter



original | horizontal Sobel filter | vertical Sobel filter



2. Neural Network

Kernel = Filter

Ex. Sobel filter

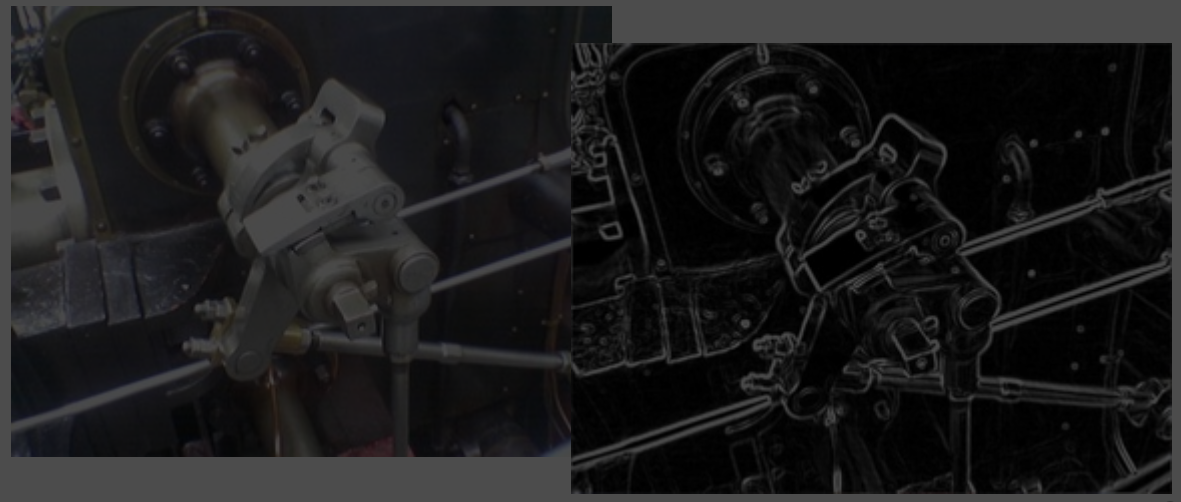
-1	0	+1
-2	0	+2
-1	0	+1

x filter

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

y filter

하나의 Filter(=Kernel)을 씌웠더니
하나의 Edge(=Feature)가 나온다 !



2. Neural Network

Kernel = Filter

Filter를 계속 씌우다 보면

→ Image를 분류할 수 있을 만큼 specific한 feature를 찾을 수 있을 것 !
(edge 찾고 → circle 찾고 → eye 찾고 → …)

→ 어떤 filter를 써야 'good' feature를 찾을 수 있을까 ?

→ Filter에 있는 값도 parameter로 두고 학습시키면, 모델이 good feature를 찾는 filter를 알아낼 것 !

→ 모델 앞쪽에 Filter를 복잡하게 깔고, (image representation = feature extractor)
Classifier에서만 MLP(FC layer)를 사용해서 NN 모델 학습하면 image classification !

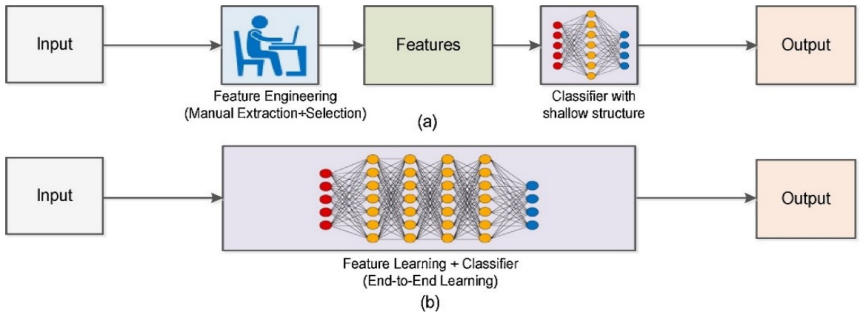
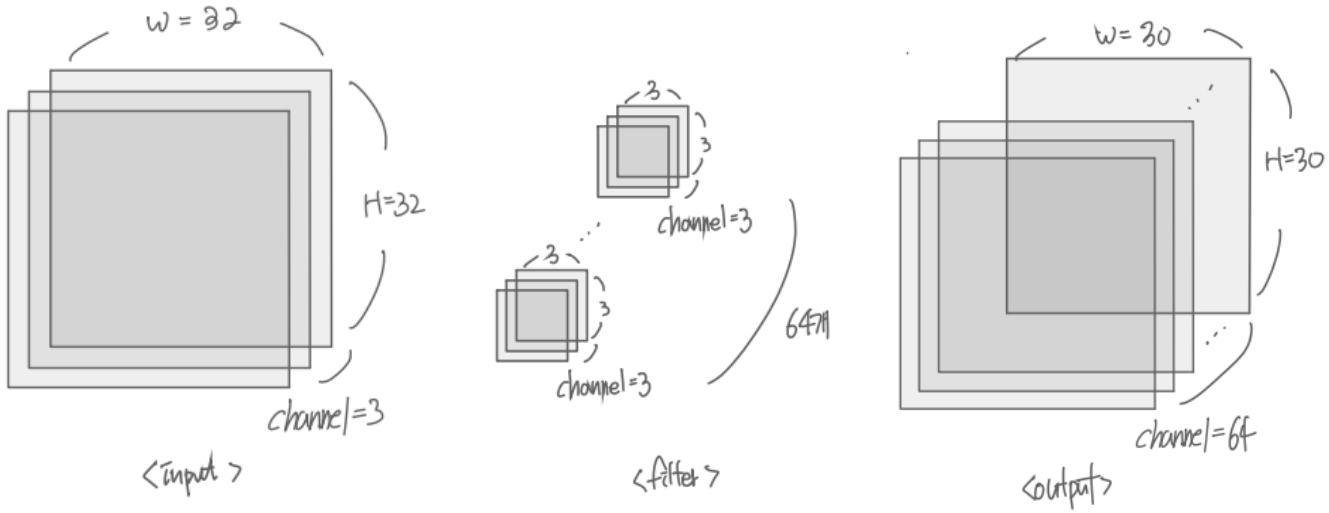


Fig. 1. (a) Traditional Computer Vision workflow vs. (b) Deep Learning workflow. Figure from [8].

3. CNN

Convolutional Layer



예시) Convolutional layer : 3*3 filter 64개

Filter의 channel은 input의 channel에 맞춰진다
→ 우리가 고려할 필요는 없음

이번 layer에서 parameter(=가중치, =filter값) 개수는 64*3*3*3 = 1,728개

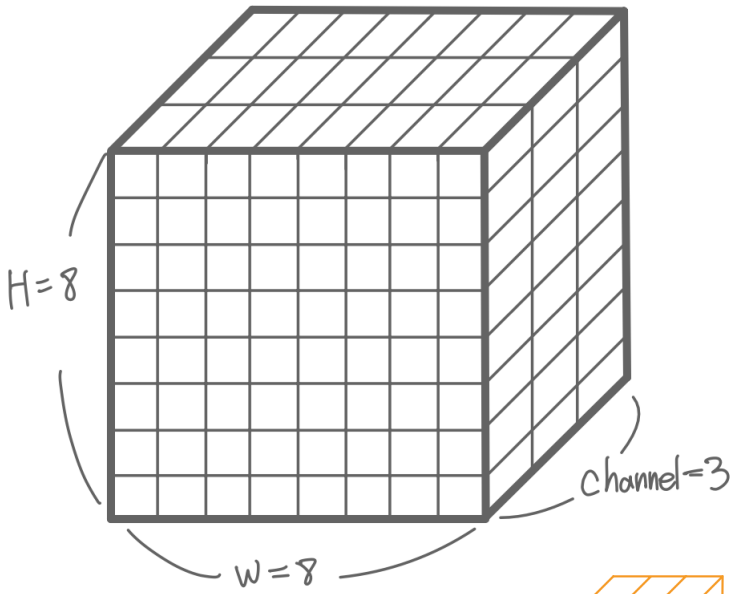
Filter 1개는 Feature map(=activation map) 1개
→ output의 channel은 filter의 개수

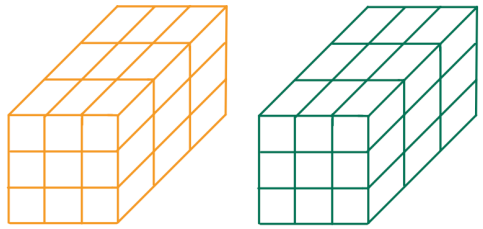
Output의 dim은 (N-F+2P)/S +1
→ N = input size
F = filter size
P = padding
S = stride

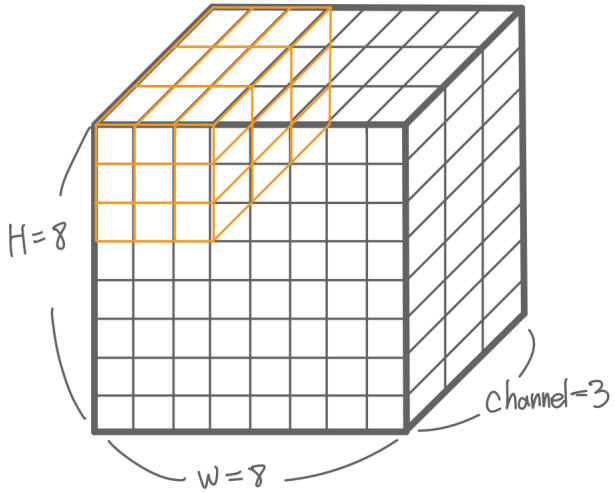
$$dim_{out} = \frac{N - F + 2P}{S} + 1$$

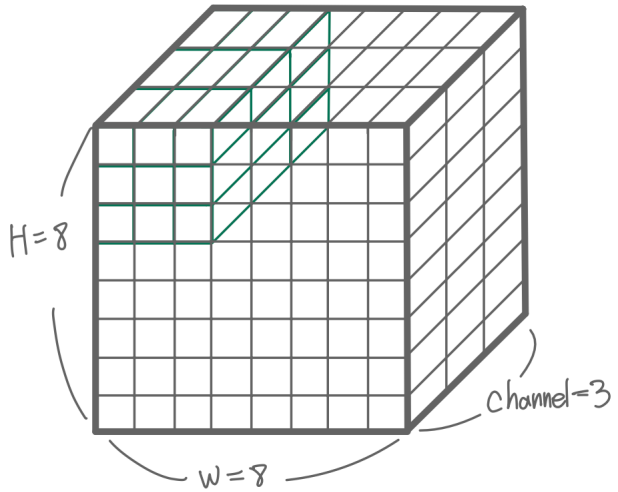
3. CNN

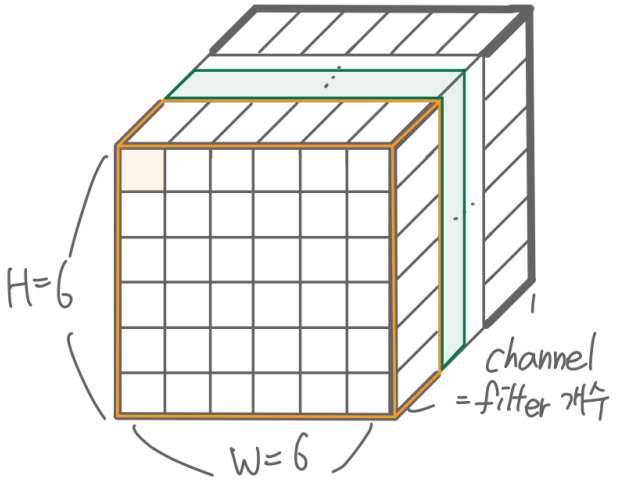
Convolutional Layer











3. CNN

Padding / Stride

Zero padding : image 주위를 0으로 둘러주는 과정

→ Convolutional layer를 거치면서 output의 dim이 작아지는 것을 방지

→ Edge pixel data를 충분히 활용

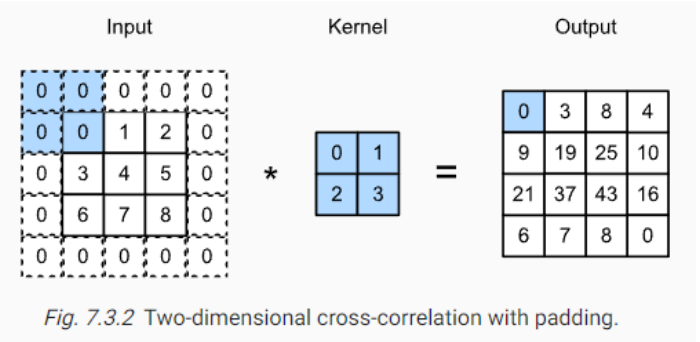
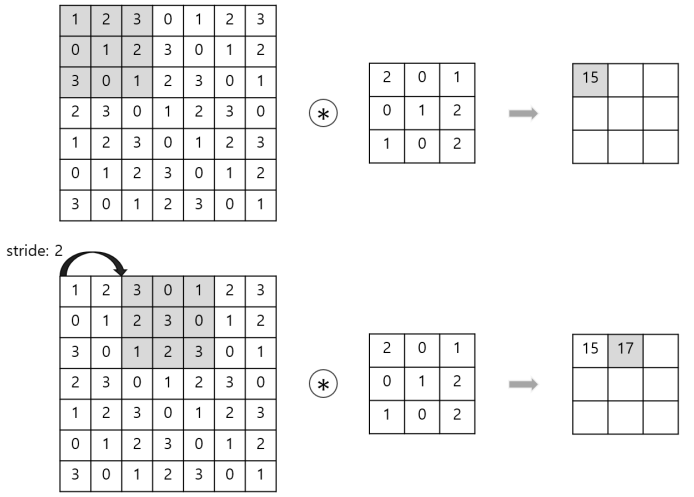


Fig. 7.3.2 Two-dimensional cross-correlation with padding.

Stride : filter가 이동하는 간격

→ Image의 dim(해상도)이 커지면서 Computation의 효율성을 위해 등장

→ stride로 나누어 정수가 아니면 해당 convolution은 성립 X



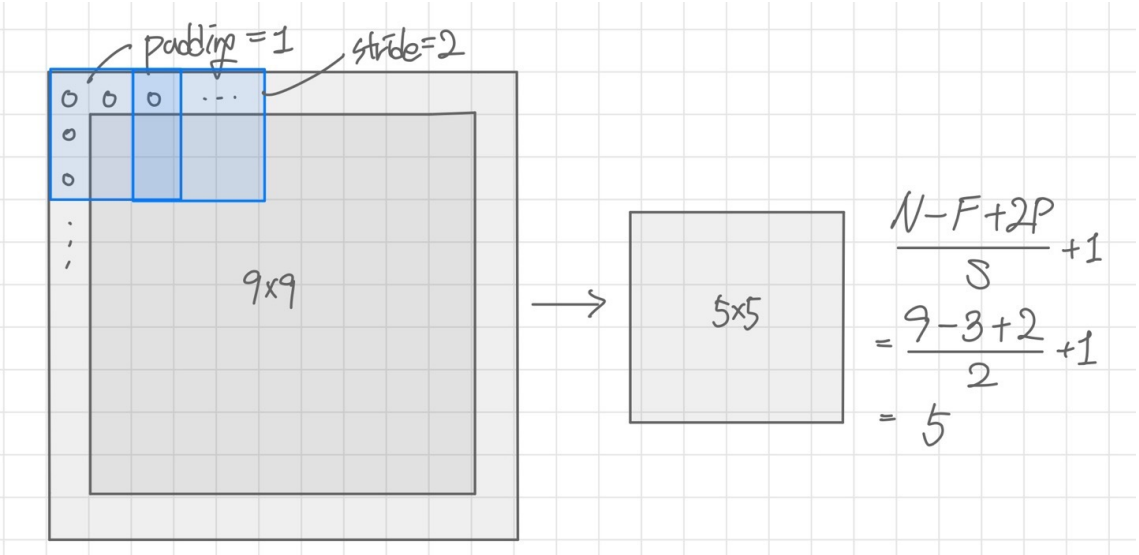
3. CNN

Padding / Stride

Input된 image는 conv layer를 거치면서
차원이 줄어드는데, (downsampling)
padding / stride를 통해 조절할 수 있다 !

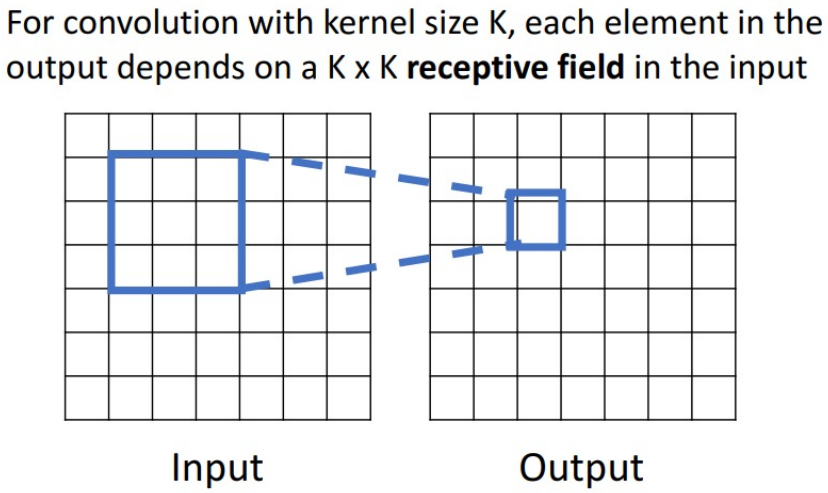
Padding은 커질수록 output dim이 덜 줄어들고
stride는 커질수록 output dim이 더 줄어든다

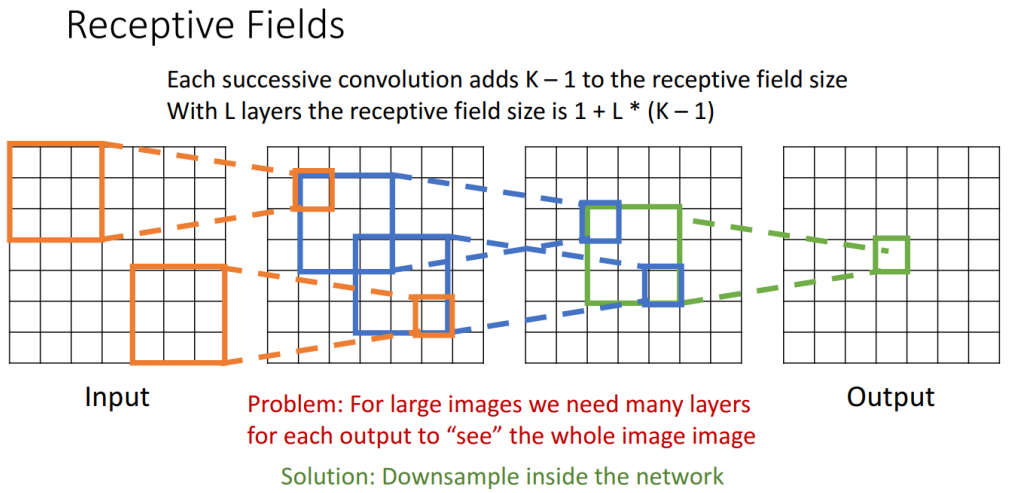
Filter를 씌워도 output dim을 유지시키기 위한
paddng size = (F-1)/2



3. CNN

Receptive Field





하나의 픽셀은 인접한 픽셀과의 correlation이 높고

output의 한 neuron은 input에서 filter size의

receptive field(neuron이 바라보는 공간)를 가짐

7*7의 receptive field를 가지기 위해서는

7*7 size의 filter 1개를 사용할 수도 있지만 (layer=1, pram=49)

3*3 size의 filter 3개를 사용할 수도 있다 (layer=3, pram=27)

3. CNN

Pooling Layer

Conv layer에 존재하는 filter(parameters)가 너무 많아짐으로 인한 overfitting을 막기 위한 layer

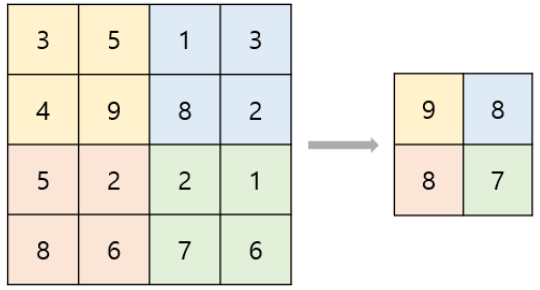
: Conv computation이 없어 parameter가 없으니 학습할 필요 X

Max Pooling

: 각 영역(지역) 내에서 가장 큰 값을 선택하여 parameter의 개수를 줄이자

: 하나의 feature map에 대해 영역을 나누어 downsampling을 수행

: (일반적으로) 각 영역이 겹치지 않도록 filter size = stride size



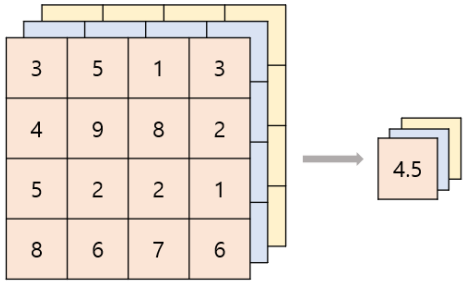
Max Pooling

Global Average Pooling

: (height, width, channel)형태의 3차원을 (channel,)형태의 1차원 벡터로

: 여러 feature map에 대해 각 feature map에서 average를 산출

: Classifier에서 FC layer를 대신하여 overfitting과 위치정보 손실을 막음



Global Average Pooling

3. CNN

Dropout

Overfitting이 발생했을 때…

Weights	Overfitting	Not-overfitting
w1	0.35	0.15
w2	100,000,412	1.2
w3	240.3	0.45
w4	-114,324,121	-1.3

모델이 x1의 값은 신경도 안쓴다…

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function　　Regularization Term

Loss function에 Regularization term을 추가

Loss function 수정 없이 “특정 노드에 극단적으로 의존하는 상황”을 해결할 수 없을까 ? → Dropout !

3. CNN

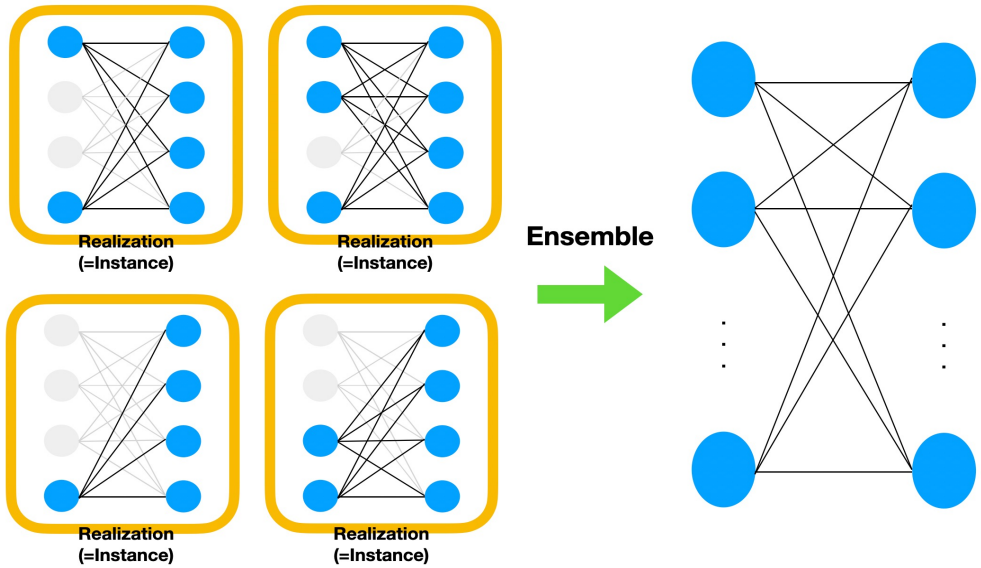
Dropout

학습할 때마다 매번 랜덤하게 뽑은 node를 제외하고 학습

→ 과도하게 의존했을 node를 빼고 학습한 경우 발생

→ Train 이후 test(inference)할 때에는 모든 node 사용

(Ensemble의 아이디어, 최신 CNN에서는 잘 안쓰임)



3. CNN

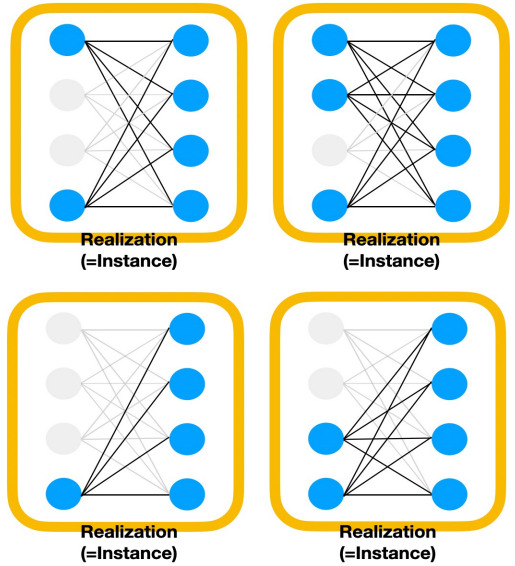
Dropout

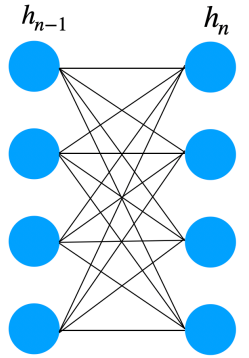
```python
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()

        ##### Layer 정의 #####
        self.layer = nn.Sequential(
            # 맨처음 RGB 채널 3개이므로 가장 처음 in_channels = 3
            # img의 가장 첫 차원이 batch_size 값은 계속해서 유지
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=6),
            nn.ReLU(),
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=17, stride=2),
            nn.ReLU(),
            nn.Dropout(0.2), # 0.2 확률로 Dropout
            nn.MaxPool2d(kernel_size=12, stride=2),
            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=15, padding=2),
            nn.ReLU(),
            nn.Conv2d(in_chan
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.MaxPool2d(kern
            nn.Conv2d(in_chan
            nn.ReLU(),
            nn.Conv2d(in_chan
        )
        self.fc_layer = nn.Se
            nn.Linear(32*7*7,
            nn.ReLU(),
            nn.Linear(100,10)
        )
```

```python
# test 함수 정의
def test(epoch):
    print(f'₩n[ Test epoch: {epoch+1} ]')

    mymodel.eval() # eval은 항상 이걸 지정하고 시작! - Dropout, Batch Normalization 등의 효과를 적용하지 않기 위함!
                   # ex. evaluation 할때는 Dropout 없이 지금까지 학습한 모든 node를 활용해서 진행해야됨

    loss = 0
    batch_losses = []
    correct = 0
    total = 0

    with torch.no_grad(): # gradient update 안함 - eval과 torch.no_grad는 하나의 세트
        for batch_idx, (inputs, targets) in enumerate(testloader_animal10):
            inputs, targets = inputs.to(device), targets.to(device)

            outputs = mymodel(inputs)
            loss = criterion(outputs, targets)
            batch_losses.append(loss.item())

            total += targets.size(0)
            _, predicted = outputs.max(1)
            correct += (predicted == targets).sum().item()

    avg_loss = sum(batch_losses) / len(batch_losses)
    seq_test_loss.append(avg_loss)
```

model.train()이면 dropout 적용

model.eval()이면 dropout 적용 X





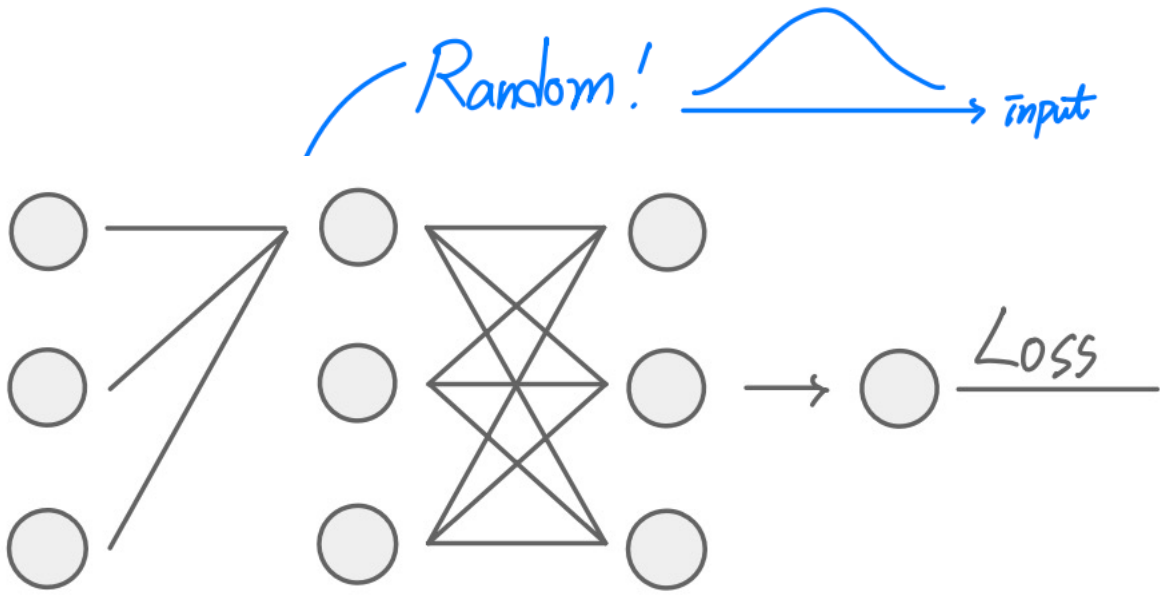
$$h_n = a(\alpha W_n h_{n-1} + b_n)$$

3. CNN

Batch Normalization

Dropout 귀찮은데…

Regularization도 되면서, Vanishing gradient 방지되면서, 빠르고 간편하게 Loss 줄이는 방법 없나 ?



$$\text{평균} = \mu_B$$

$$\text{분산} = \sigma_B^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B}$$

$$y_i = \gamma \hat{x}_i + \beta$$

training!

Initial = (1,0)

Test 할 때에는…

$$\text{평균} = E[\mu_B]$$

$$\text{분산} = E[\sigma_B^2] \times \frac{m}{m-1}$$

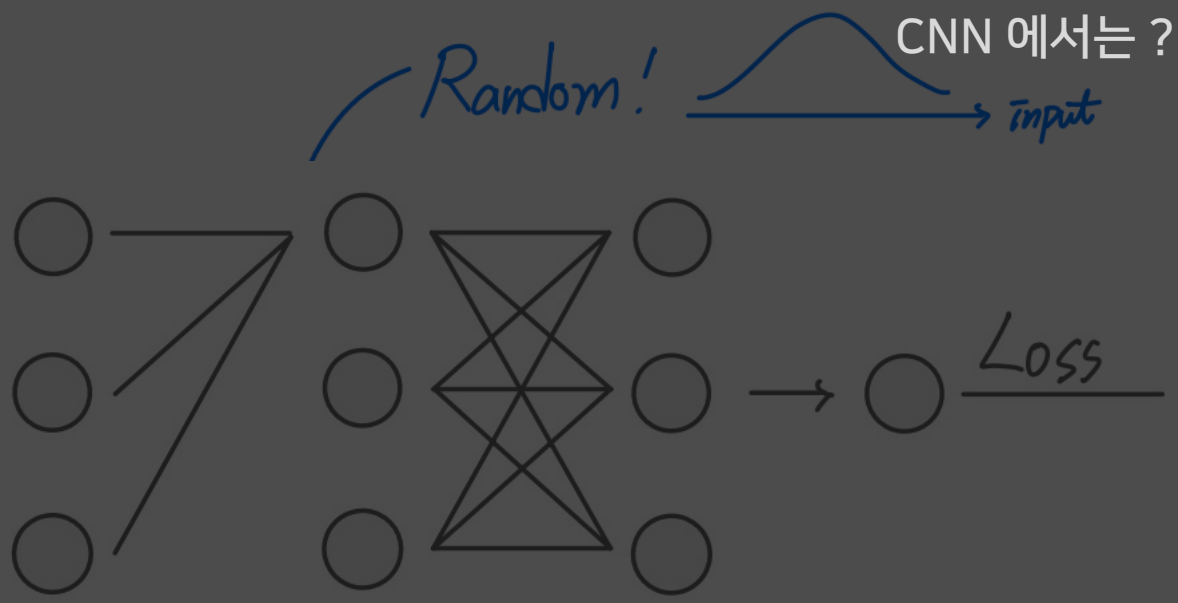
Exponential moving average

3. CNN

Batch Normalization

Dropout 귀찮은데…

Regularization도 되면서, Vanishing gradient 방지되면서, 빠르고 간편하게 Loss 줄이는 방법 없나 ?



평균 $= \mu_B$

분산 $= \sigma_B^2$

$\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B}$

$y_i = \gamma \hat{x}_i + \beta$

training!

Initial = (1,0)

Test 할 때에는…

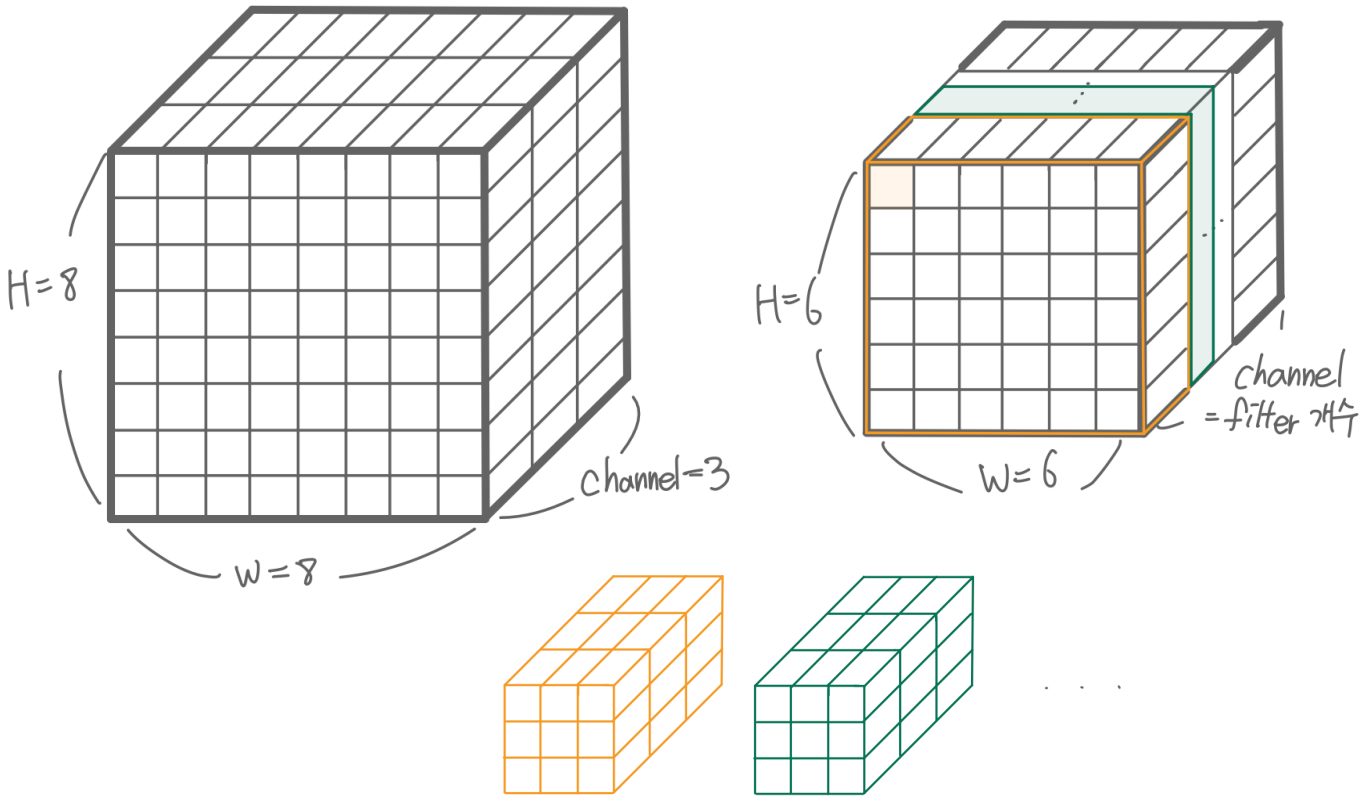
평균 $= E[\mu_B]$

분산 $= E[\sigma_B^2] \times \frac{m}{m-1}$

Exponential moving average

3. CNN

Batch Normalization



필터당 γ, β는 하나씩 !

Ex. filter가 1개, Batch size = 32이면,

출력의 dim은 6*6*1이고

μ_B 와 σ_B^2 를 구한 sample의 수는 32*6*6

Filter의 개수가 많아지면

학습 parameters인 γ, β의 pair가 많아짐

Vanishing gradient와는 무슨 상관? →

3. CNN

Batch Normalization

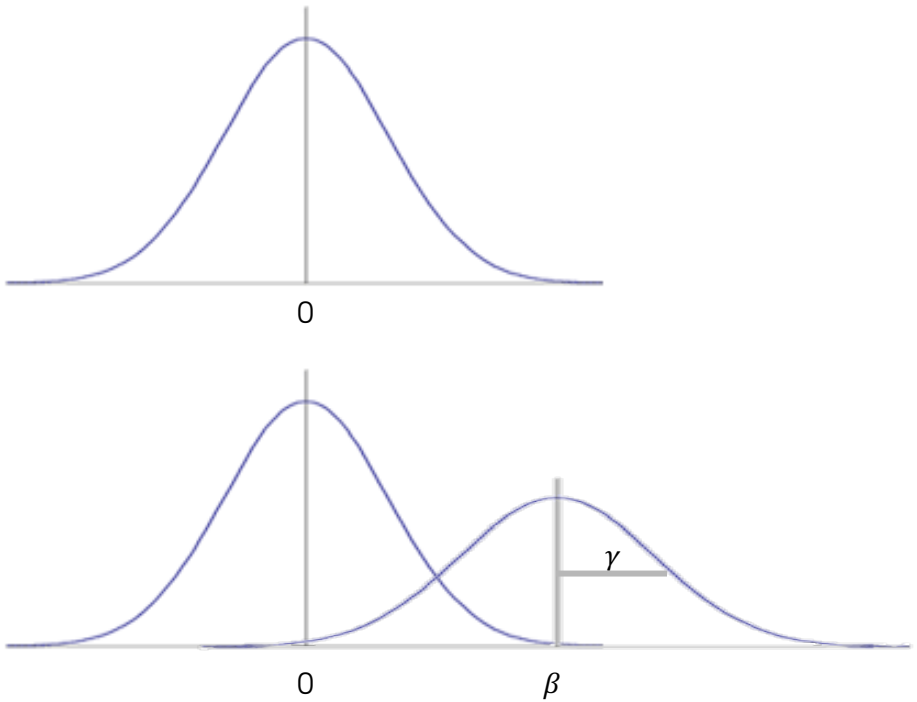
Vanishing gradient랑은 무슨 상관?

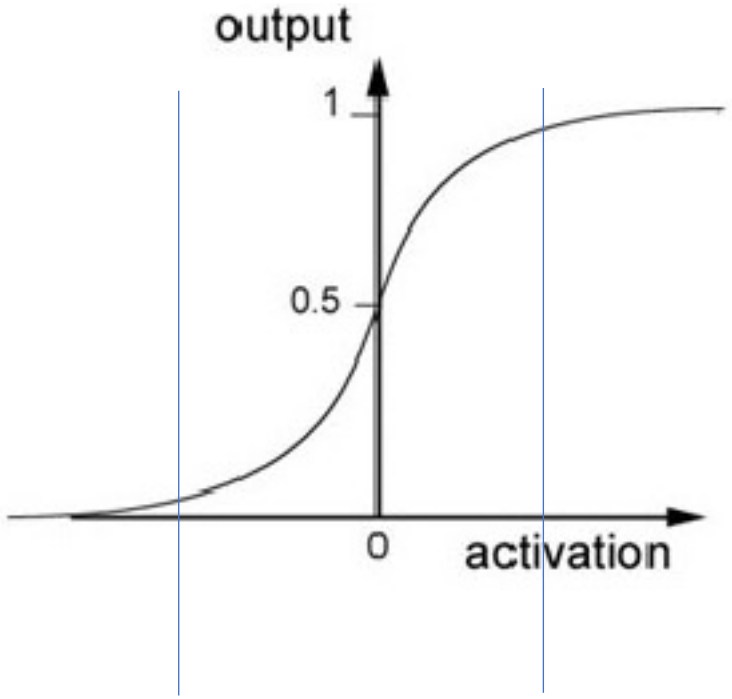
평균 $= \mu_B$

분산 $= \sigma_B^2$

$\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B}$

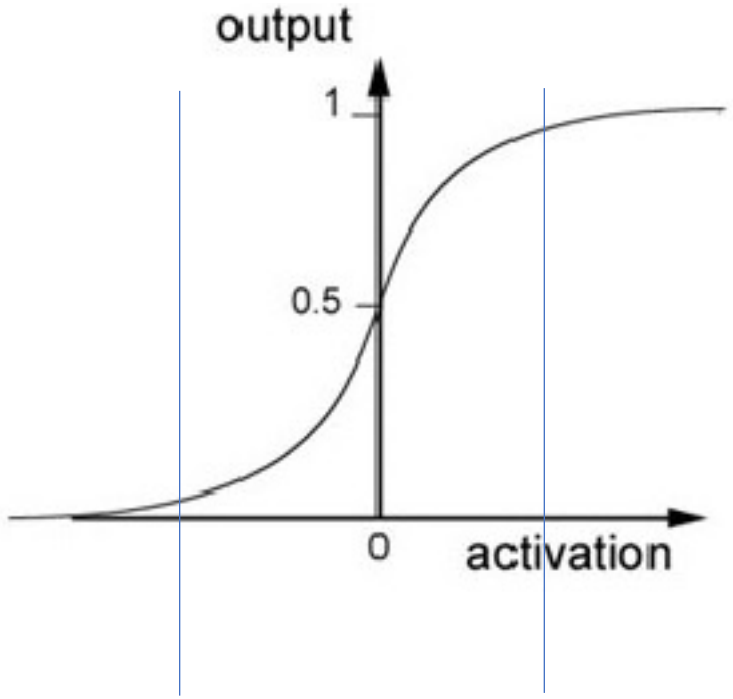
$y_i = \gamma \hat{x}_i + \beta$ training!





3. CNN

Batch Normalization



Gradient가 가파르고 & 비선형성이 있는 구간에 sample들을 뿌려준다

→ Vanishing gradient를 완화할 수 있다 !

→ Output의 분포가 좁아지므로 learning rate를 크게 설정하여 학습 속도를 향상시킬 수 있다 !

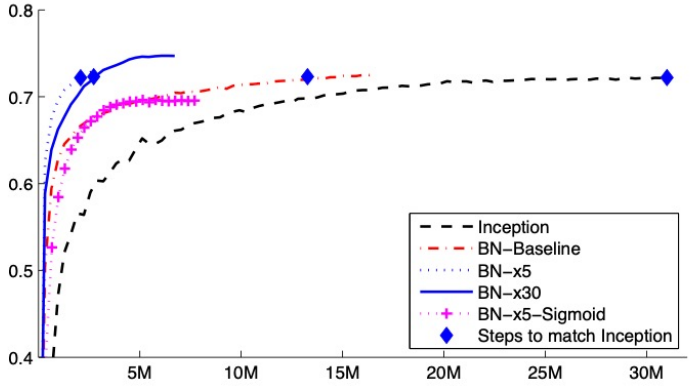


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

3. CNN

Summary

Feature Extractor는 아래 과정의 반복

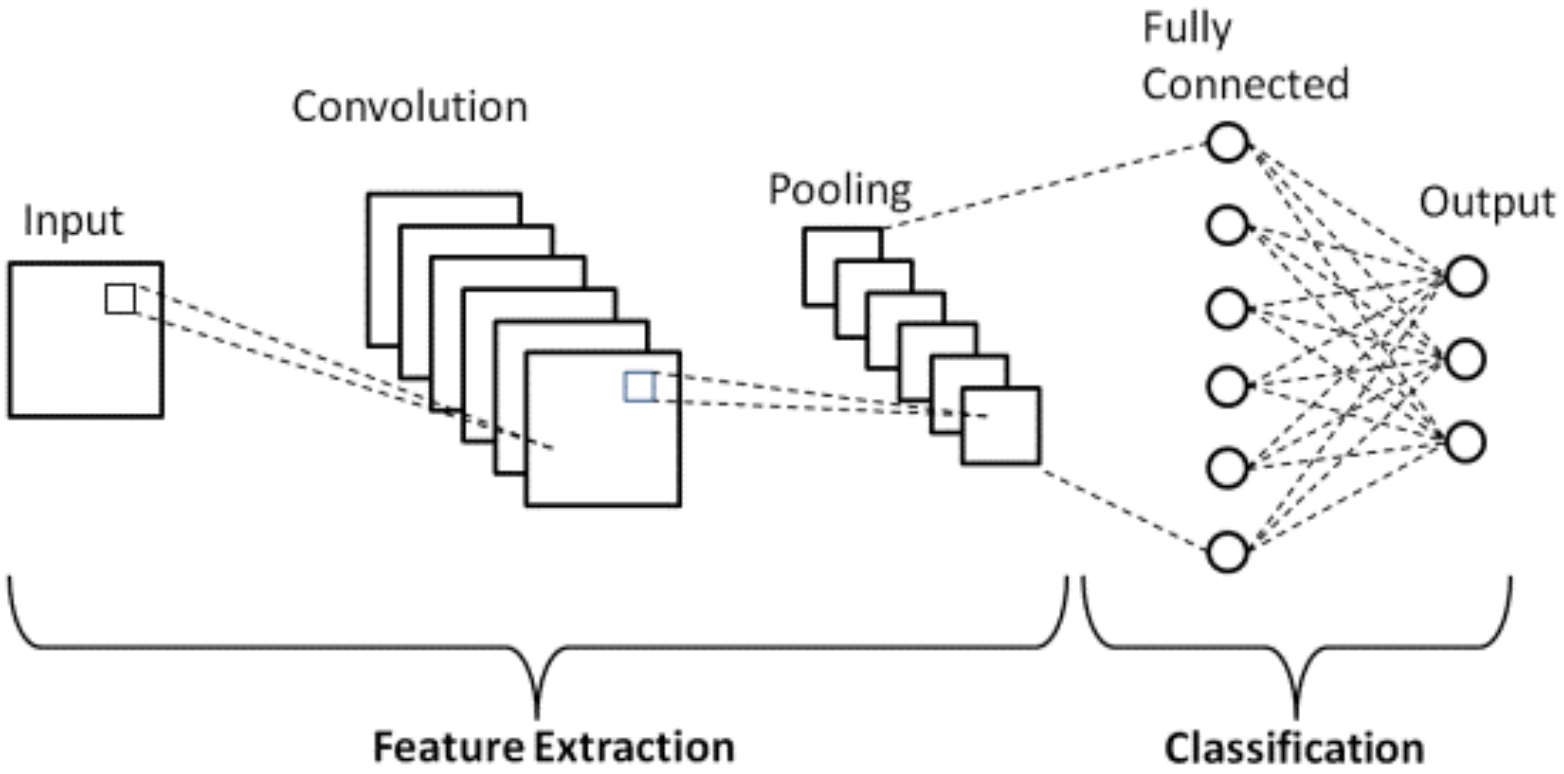
Conv Layer → Batch Normalization → Activation func (ReLU, Sigmoid, ⋯) → (Dropout) → (Pooling)

Classifier는 Feature Extractor 과정의 반복이 끝나면

Global Average Pooling → FC Layer → Softmax → Classification

3. CNN

Summary



4. Image Dataset

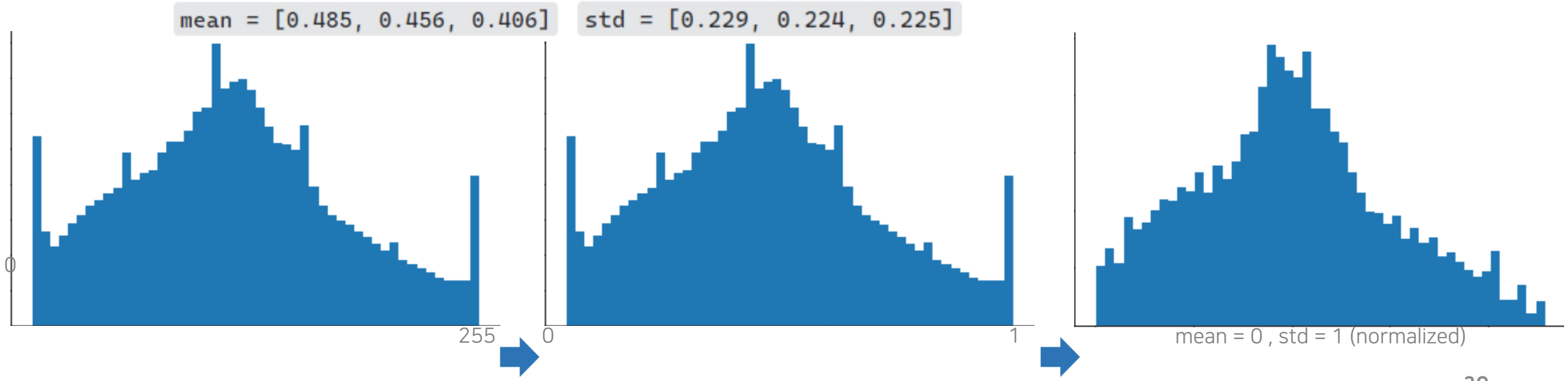
Data Preprocessing

더 좋은 train을 위해서, Image를 그대로 쓰기보다는

pixel (0~255 정수)를 channel-wise normalization (0~1 실수)

→ Train data에서 구한 mean, std로 test data도 normalize !

(train image data가 너무 많다면 ImageNet의 mean, std을 사용하기도 함)



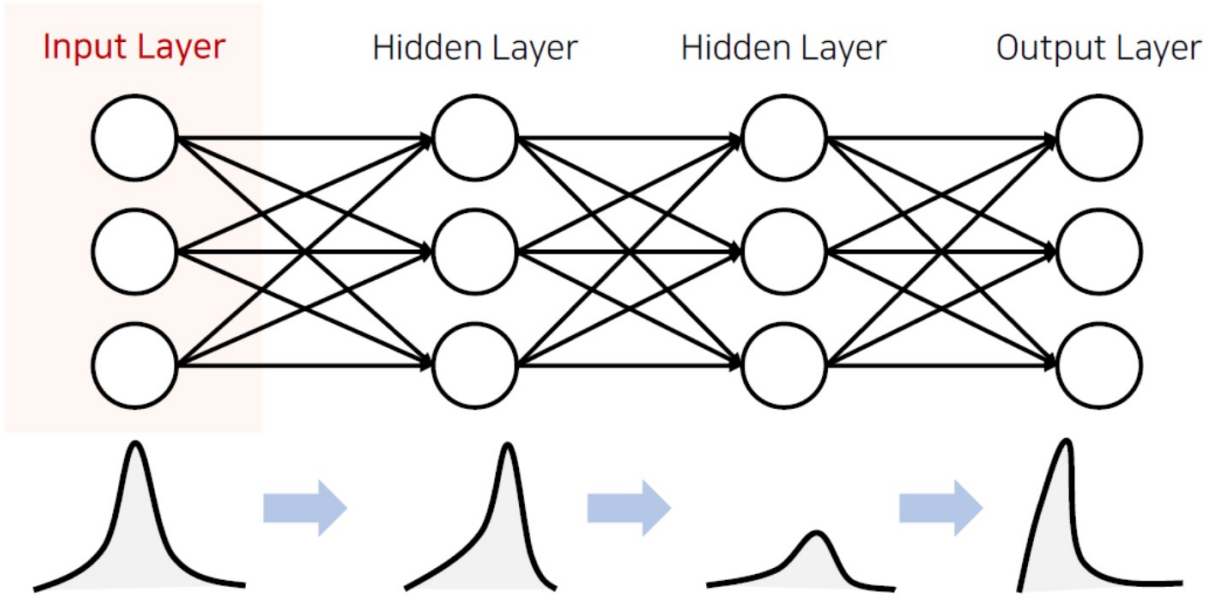
4. Image Dataset

Data Preprocessing

Input data normalize 해도 layer를 거치면서 분포가 달라질 수 있음 (Vanishing gradient 유발)

→ Layer를 거치는 과정에 있는 data들도 normalize 해야 할 필요성

= Batch Normalization (BN)

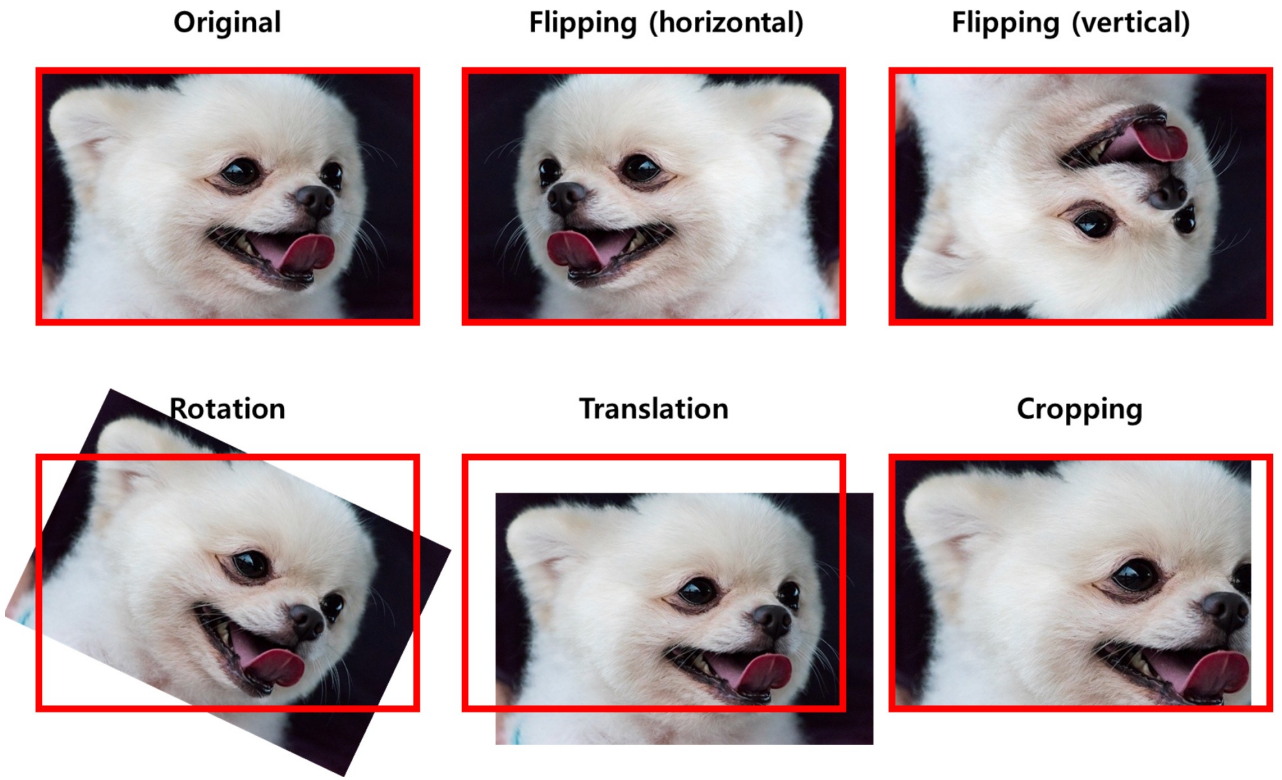


4. Image Dataset

Image Augmentation

하나의 image data로 여러 개의 data를 만들어내는 데이터 증강 기법

- Image Data Augmentation
  - Basic Image Manipulation
    - Geometric Transformations
    - Color Space Transformations
    - Kernel Filters
    - Random Erasing
    - Mixing Images
  - Deep Learning Approaches
    - Feature Space Augmentation
    - Adversarial Training
    - GAN Data Augmentation
    - Neural Style Transfer
  - Meta Learning
    - Neural Augmentation
    - AutoAugment
    - SmartAugmentation



4. Image Dataset

Image Augmentation

4. Image Dataset

Popular Dataset

새로운 model이나 method가 등장할 때, 기존의 다른 모델과 비교/평가 하기 위한

기준이 되는 popular (common) dataset이 필요

ex. MNIST, CIFAR-10, ImageNET, …

Dataset	Modality	Size	Model	Test Set Errors				
				CL guessed	MTurk checked	validated	estimated	% error
MNIST	image	10,000	2-conv CNN	100	100 (100%)	15	-	0.15
CIFAR-10	image	10,000	VGG	275	275 (100%)	54	-	0.54
CIFAR-100	image	10,000	VGG	2235	2235 (100%)	585	-	5.85
Caltech-256	image	30,607	ResNet-152	4,643	400 (8.6%)	65	754	2.46
ImageNet*	image	50,000	ResNet-50	5,440	5,440 (100%)	2,916	-	5.83
QuickDraw	image	50,426,266	VGG	6,825,383	2,500 (0.04%)	1870	5,105,386	10.12
20news	text	7,532	TFIDF + SGD	93	93 (100%)	82	-	1.11
IMDB	text	25,000	FastText	1,310	1,310 (100%)	725	-	2.9
Amazon	text	9,996,437	FastText	533,249	1,000 (0.2%)	732	390,338	3.9
AudioSet	audio	20,371	VGG	307	307 (100%)	275	-	1.35

*Because the ImageNet test set labels are not publicly available, the ILSVRC 2012 validation set is used.

4. Image Dataset

Popular Dataset

[MNIST]

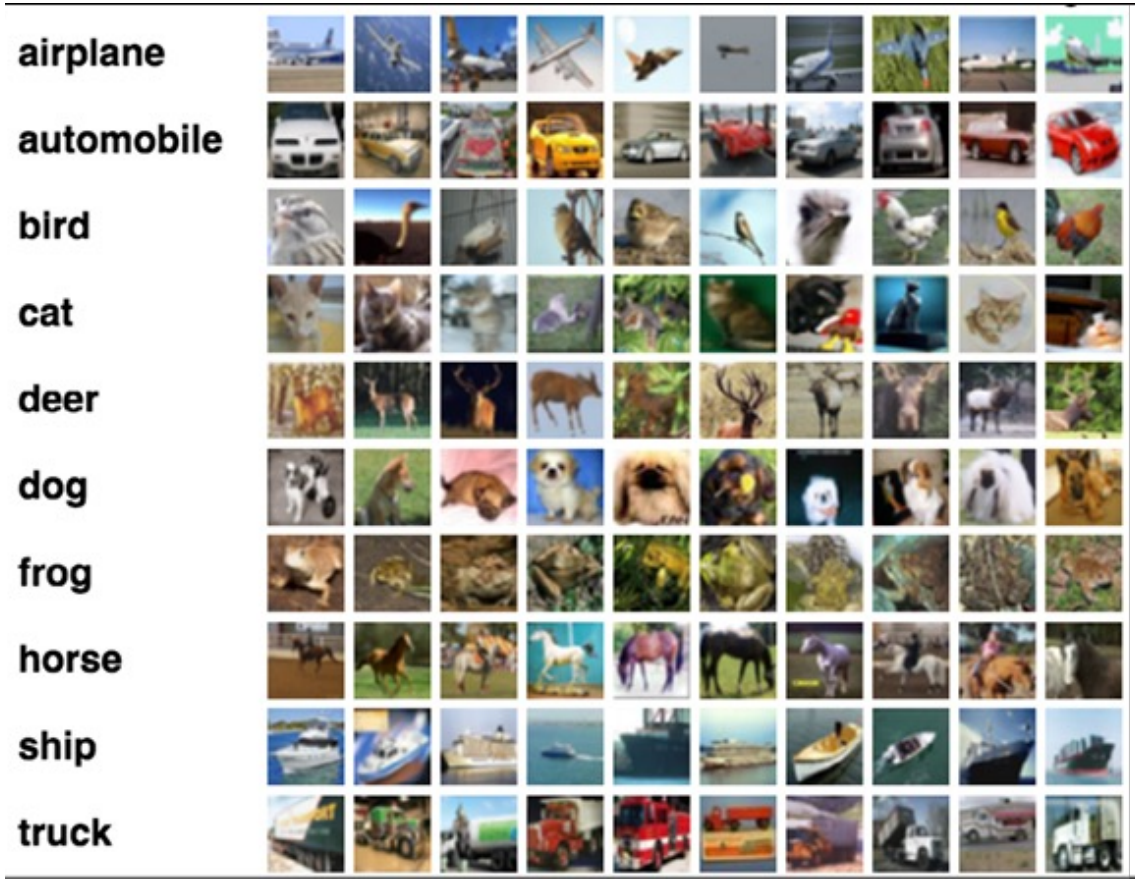
0~9 숫자 필기체 image data & label

Pixel 28*28, channel = 1 (흑백)

train 60,000개, test 10,000개

4. Image Dataset

Popular Dataset



[CIFAR-10]

10개의 class인 image data & label

Pixel 32*32, channel = 3 (컬러)

Class마다 train 6,000개, test 1,000개

(총 70,000개)

CIFAR-100 은 100개의 class

4. Image Dataset

Popular Dataset

[ImageNet]

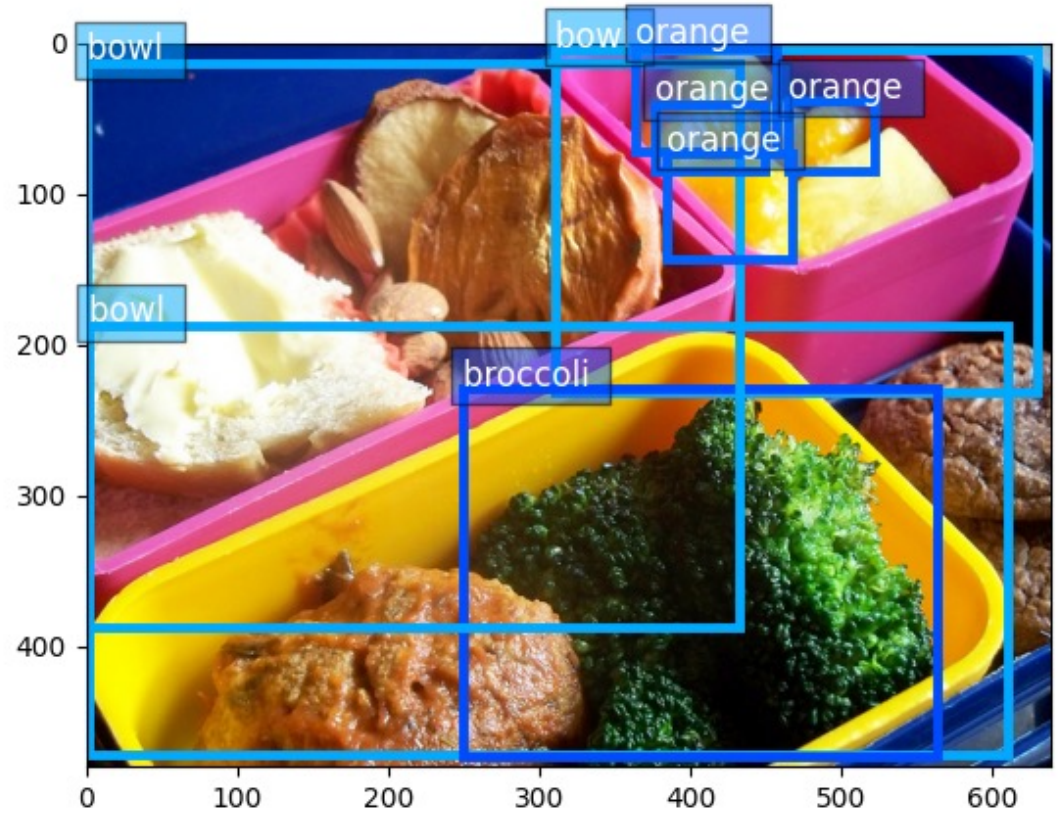
20,000개 이상의 class

14,197,122개 image data

굉장히 무거운 dataset으로, 최신 CNN에서 쓰임

4. Image Dataset

Popular Dataset

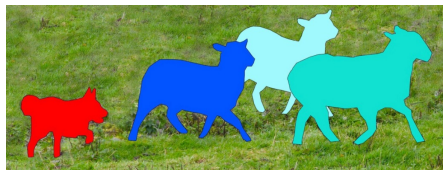


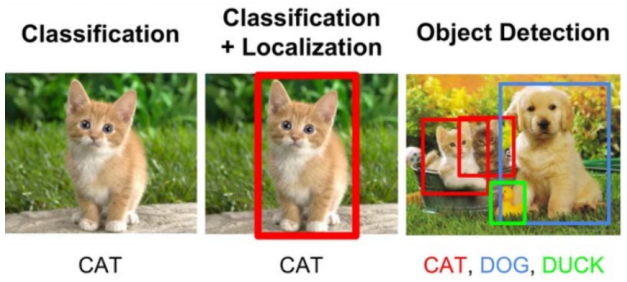
[(Microsoft) COCO]

33,000개의 labeled data

용도 : Object detection,

Instance segmentation, ···





5. History of CNN

What is 'Good' CNN model?

무작정 Feature Extractor 쌓고, Classifier 써서 CNN 모델 만들면 모든 이미지를 분류할 수 있을까?

→ NO. 유의미한 Feature를 만들어낼 수 있도록 모델을 구성해야 함

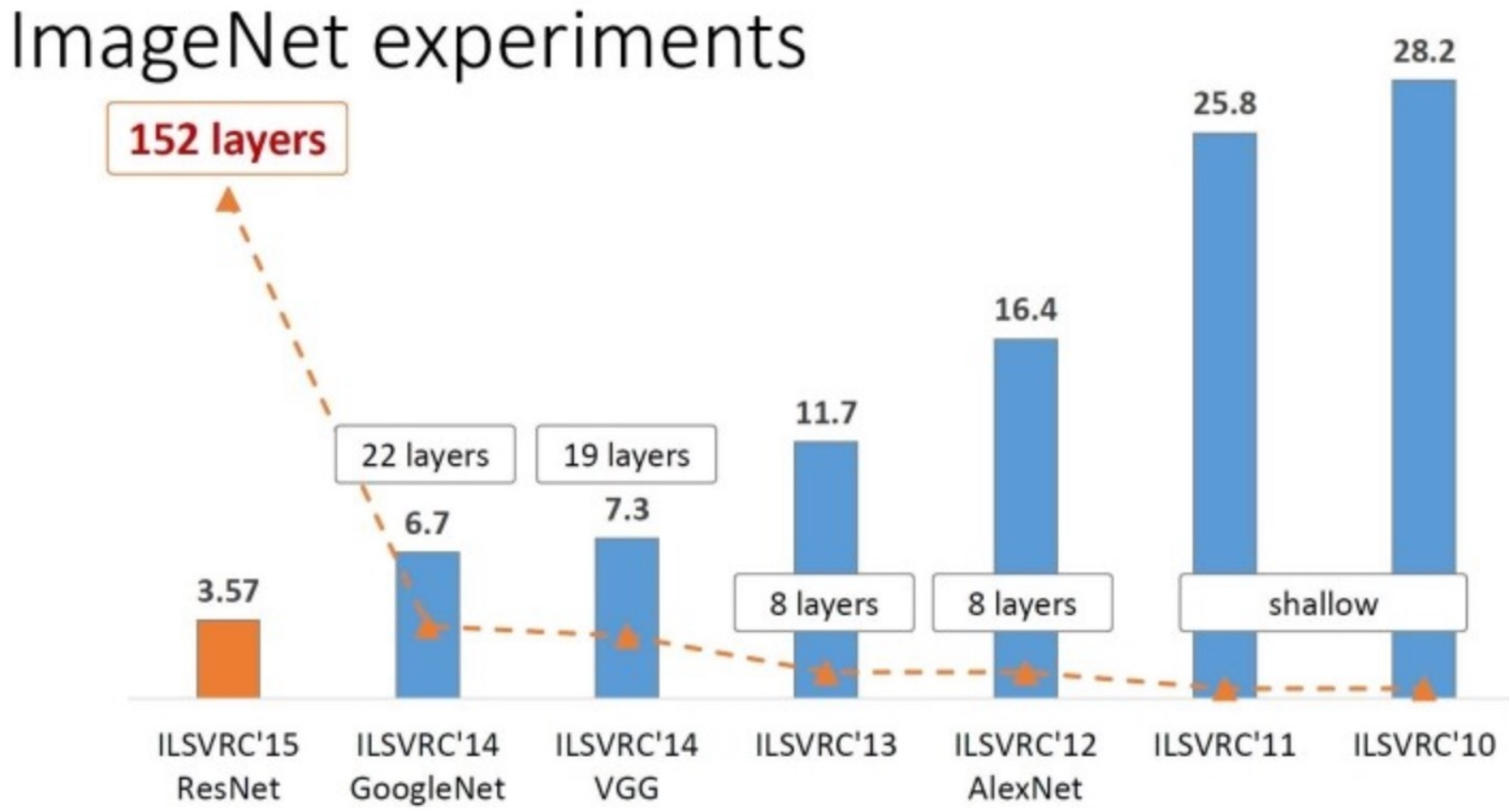
→ Filter의 크기, Network의 깊이, Activation func, 언제 어디서 Padding, Stride, Pooling 할지, …

1998년 Yann LeCun이 Convolutional Neural Network를 최초로 개발한 이후,

Image Recognition의 성능(accuracy, time complexity, …)을 높이기 위한 다양한 모델이 등장

→ 유의미하다고 평가받는 모델들을 살펴보면서, CNN 모델의 발전을 알아보고 배운 개념을 이해해보자.

5. History of CNN



5. History of CNN

AlexNet (2012)

"First CNN-based winner"

Input : 227 * 227 *3

5 Conv layer + 3 FC layer

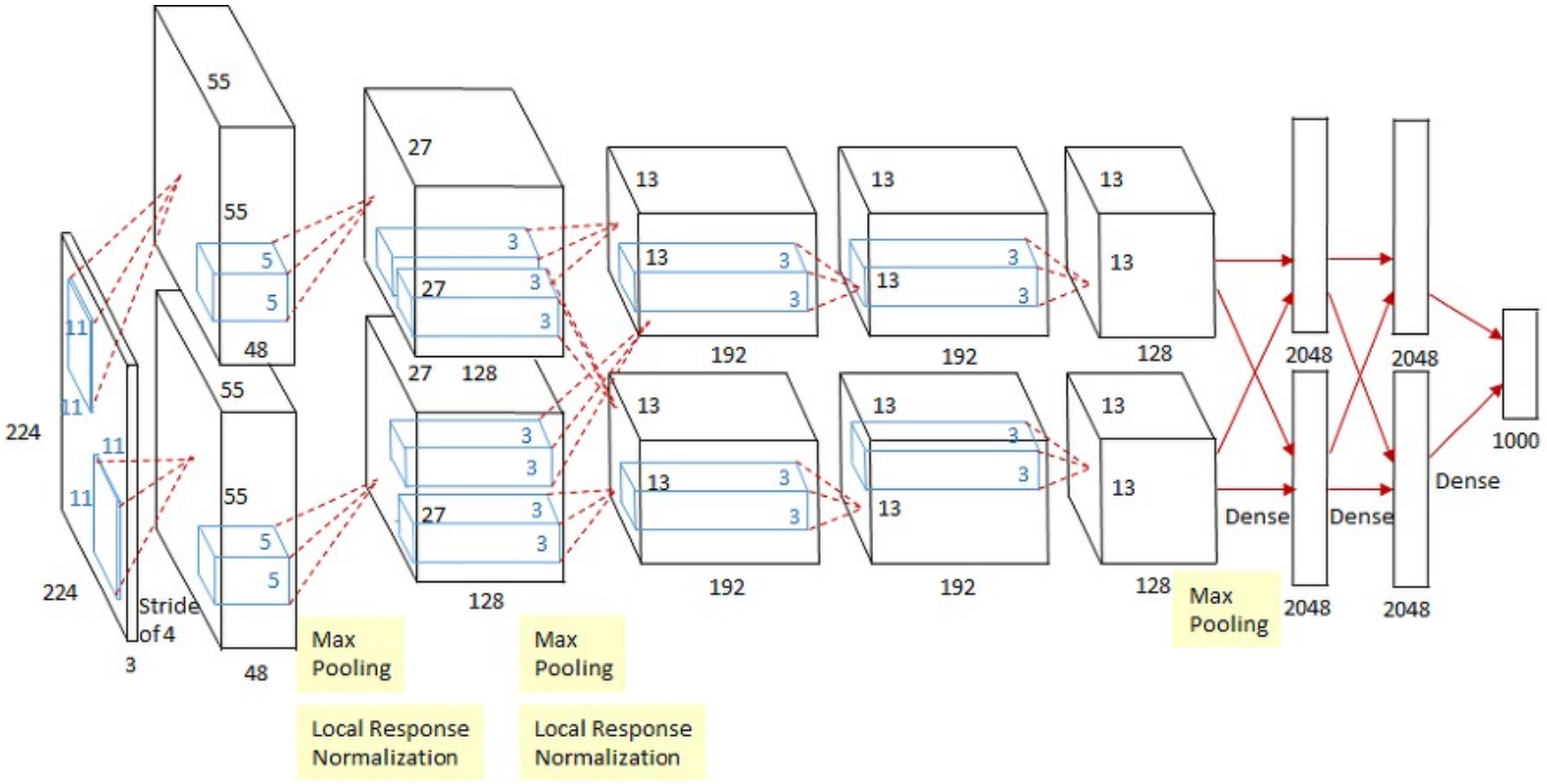
ReLU Nonlinearities

Max pooling (Overrapping)

Local Response Normalization

Data Augmentation

Dropout (in FC layer 1,2)



5. History of CNN

VGGNet (2014)

"Deeper Network"

3*3 small filter(kernel)

→ Deeper Network

→ 동일한 Receptive field에 대해

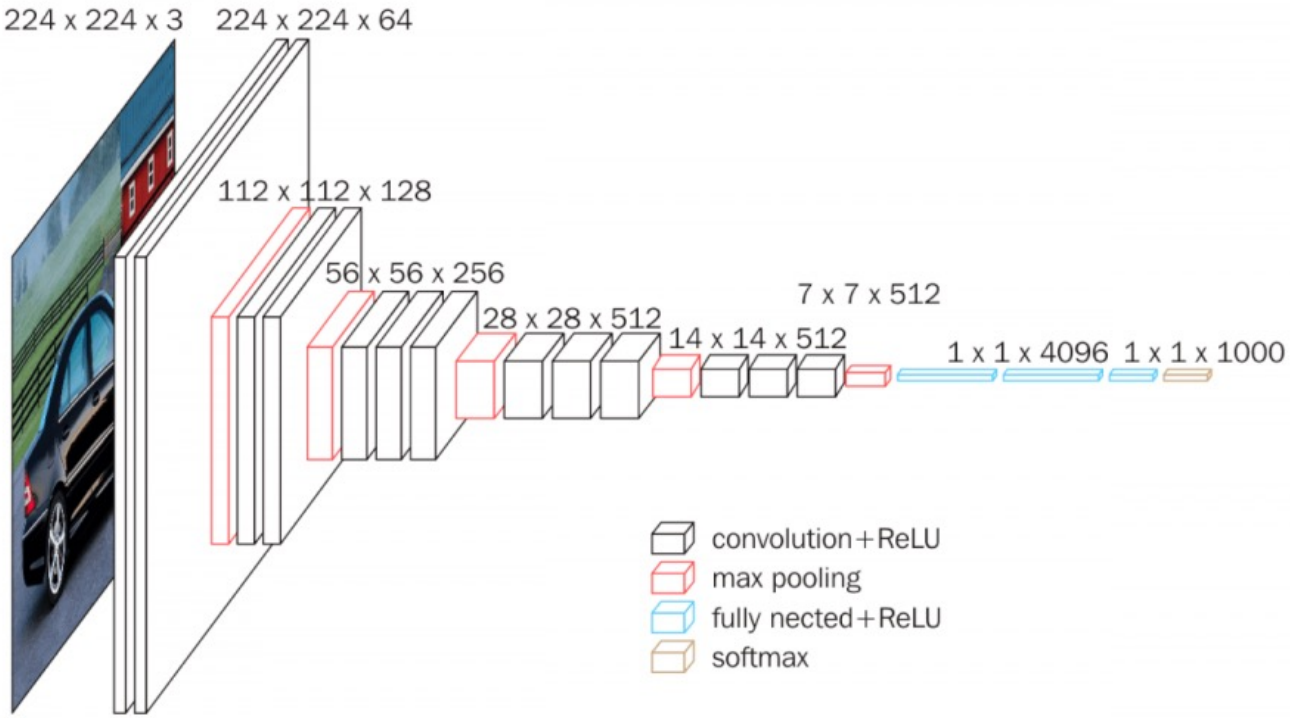
More non-linearities & Less parameters

→ 풍부한 feature를 효율적으로 뽑아낼 수 있음

But 초반 feature map이 크고, FC layer 사용

→ memory도 많이 필요하고 parameter도 많음

→ 지금 관점에서 보면 별로 효율적이지는 않음



5. History of CNN

GoogleNet (2014)

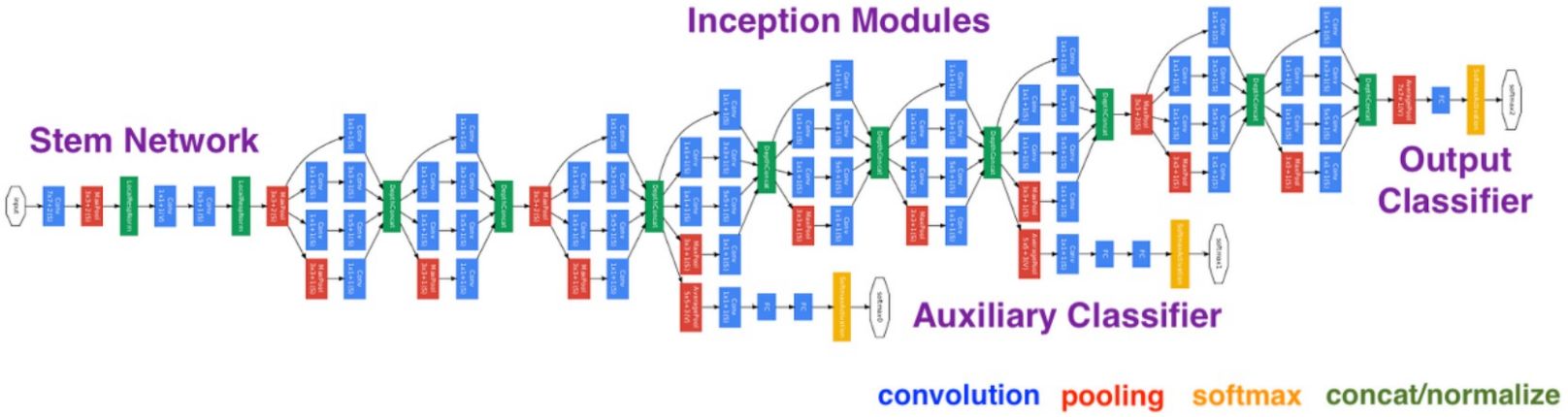
"Inception-v1"

Stem Network

Inception Modules

Auxiliary Classifier

Output Classifier



5. History of CNN

GoogleNet (2014)

Network 초반에 큰 filter 하나를 통과시켜 빠르게 downsampling !

"Inception-v1"

Stem Network

Inception Modules

Auxiliary Classifier

Output Classifier

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

5. History of CNN

GoogleNet (2014)

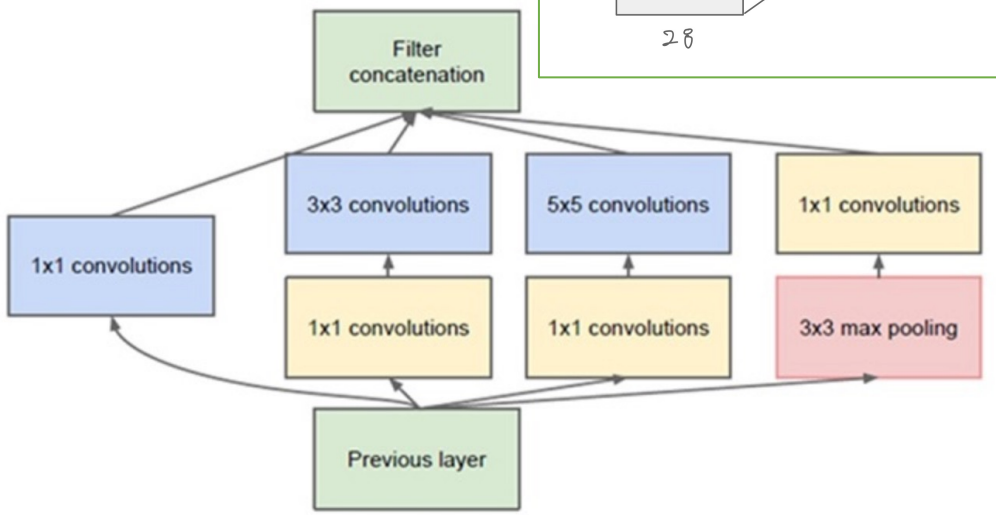
"Inception-v1"

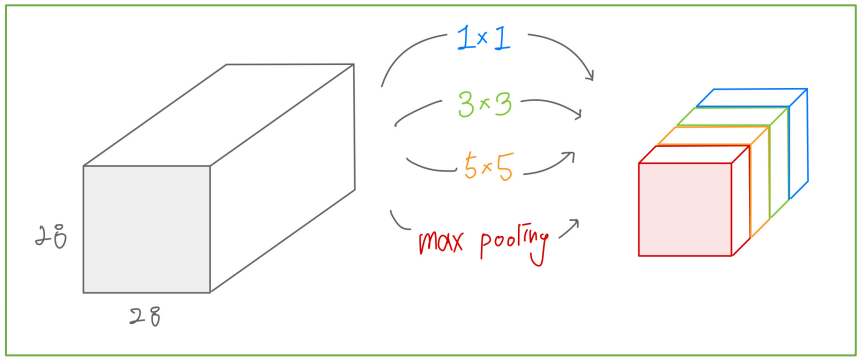
Stem Network

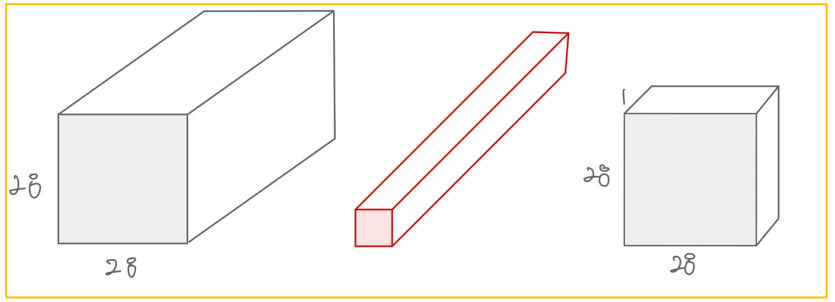
Inception Modules

Auxiliary Classifier

Output Classifier







다양한 size의 filter를 씌우려면

parameters가 너무 많아지지 않나?

→ 1*1 conv filter로 해결 !

5. History of CNN

GoogleNet (2014)

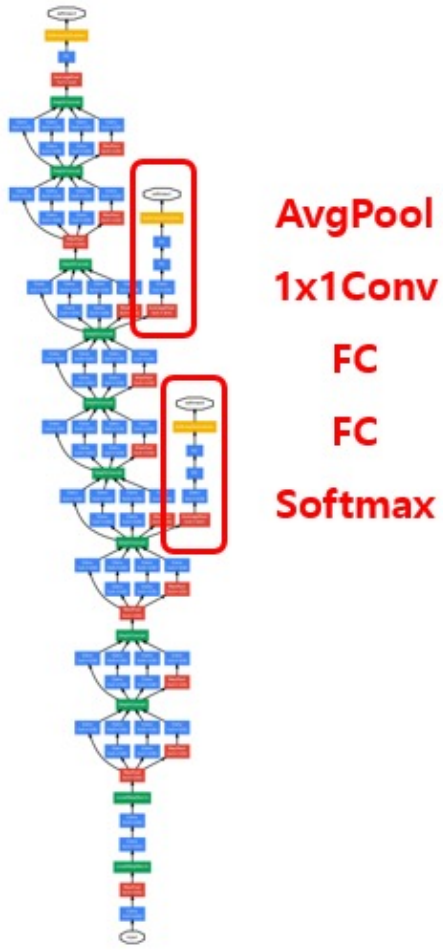
"Inception-v1"

Stem Network

Inception Modules

Auxiliary Classifier

Output Classifier



모델의 layer가 깊어지면 backpropagation에서 Input쪽에는 vanishing gradient 발생

→ 추가적인 Softmax Classifier 붙이자 !

5. History of CNN

GoogleNet (2014)

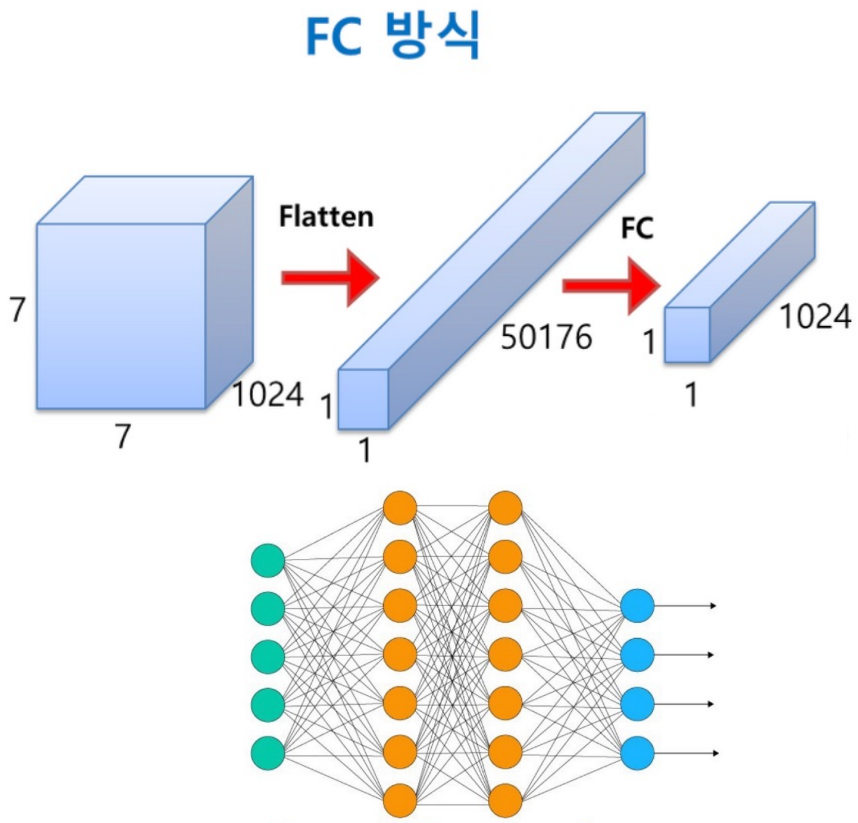
"Inception-v1"

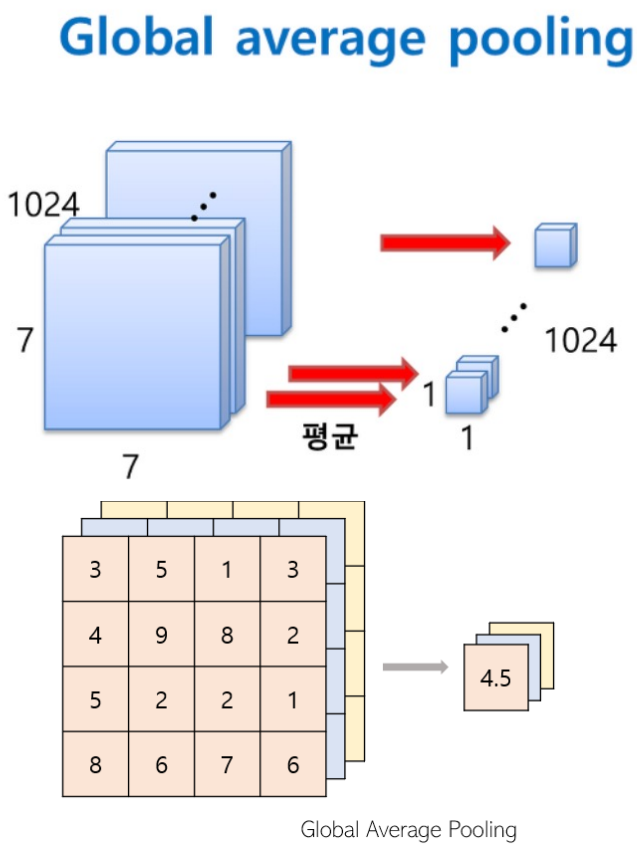
Stem Network

Inception Modules

Auxiliary Classifier

Output Classifier





6. Summary

Summary

1. What is CNN process ?

   Input Image – Extract Features – Concatenate into a vector – Classification

   Feature Extractor는 아래 과정의 반복

   Conv Layer ➡ Batch Normalization ➡ Activation func (ReLU, Sigmoid, ···) ➡ (Dropout) ➡ (Pooling)

   Classifier는 Feature Extractor 과정의 반복이 끝나면

   Global Average Pooling ➡ FC Layer ➡ Softmax ➡ Classification

2. 하나의 Filter(=Kernel)을 씌우면 하나의 Edge(=Feature)가 나온다 !

   Filter의 개수 = Feature의 개수 = output의 채널

3. $dim_{out} = \frac{N-F+2P}{S} + 1$

6. Summary

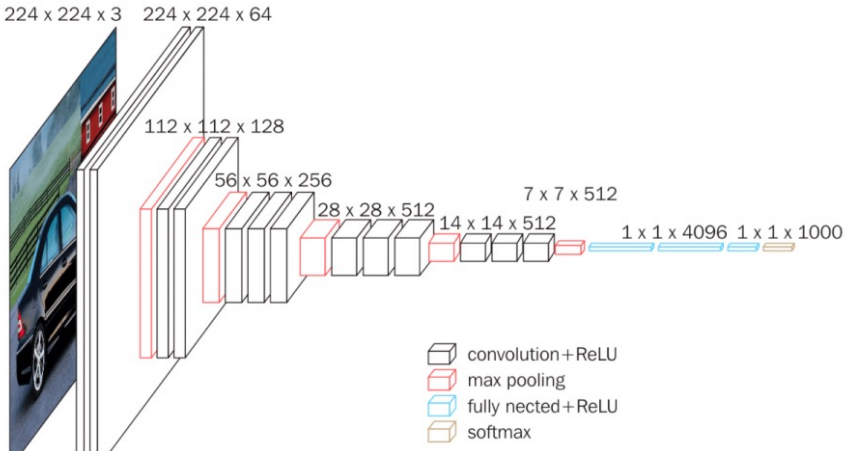
Next Session

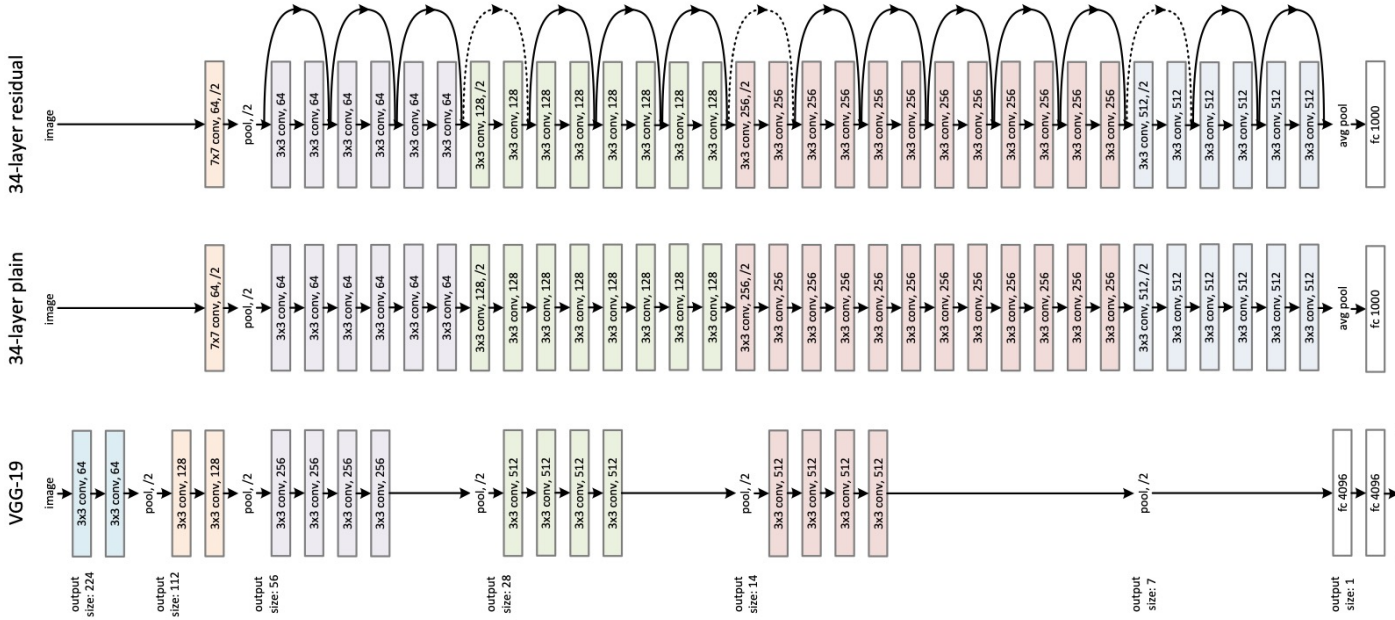
CNN Architecture

Ex. VGGNet, GoogleNet, ResNet, ···

Comparing Model Complexity

(Accuracy, # of parameters, G-ops)





6. Summary

Reference

- 22-2 정규세션 : 7기 전재현님 <CNN>
- 23-1 정규세션 : 8기 백민준님 <Linear Regression & SVM>
- Youtube <혁펜하임의 "꽂히는" 딥러닝 (Deep Learning)>
- IMaR Technology Gateway, Institute of Technology Tralee, Tralee, Ireland (2020) *Deep Learning vs. Traditional Computer Vision*
- Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner (1998) *Gradient-Based Learning Applied to Document Recognition*
- Alex Krizhevsky , Ilya Sutskever, and Geoffrey E. Hinton (2017) *ImageNet Classification with Deep Convolutional Neural Networks*
- Visual Geometry Group, Department of Engineering Science, University of Oxford (2015) *VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION*
- Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich (2015) *Going deeper with convolutions*

DATA SCIENCE LAB

발표자 : 정건우 010-6473-3938
E-mail : wjdrjsdn3938@naver.com