CNN (Convolutional Neural Network)

Multiple Levels of Abstraction

Pixel 정보 인식

↓

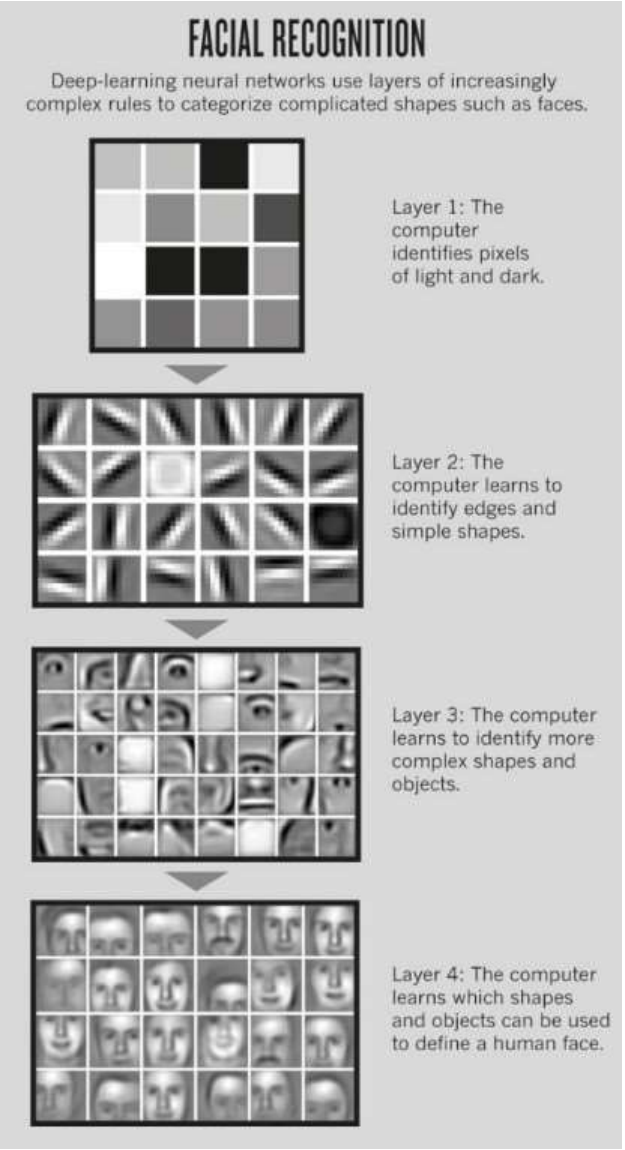
Edge/Simple Shape 특성 학습

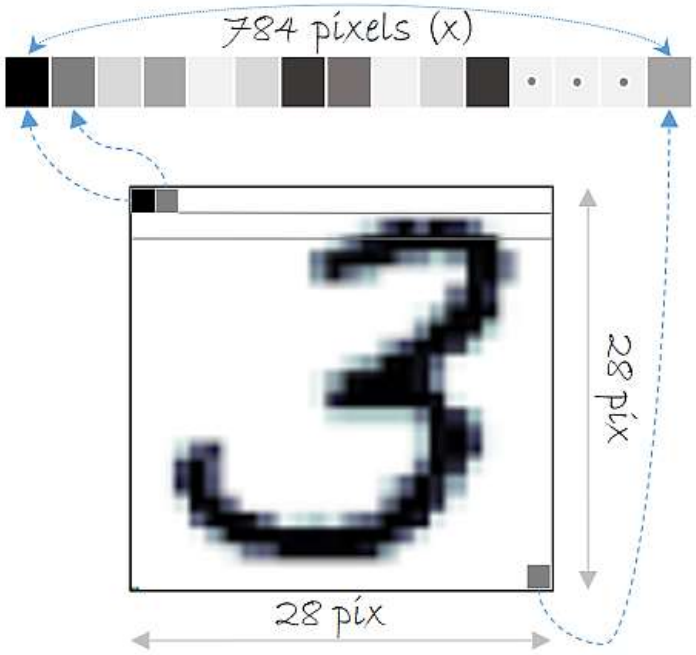
↓

Complex Shape 특성 학습

↓

얼굴인식에 필요한 특성 학습





이전 실습 모델(Dense Layer) 에서의
input image 처리

60000 x **28 X 28** → reshape(60000, **784**)

Image Data 의
공간적, 지역적 특성 상실

If 1 mega pixel → 1,000 x 1,000 x 3 = 3 million features !!!

→ **300 만 차원 x Layer 수 x 각 Layer 의 Neuron 수**

→ **계산량 급증**

→ Image Data 를 처리하기 위한 특별한 구조의 Neural Network 필요

CNN 의 특별한 Layers

- **Convolutional Layer (합성곱층)**

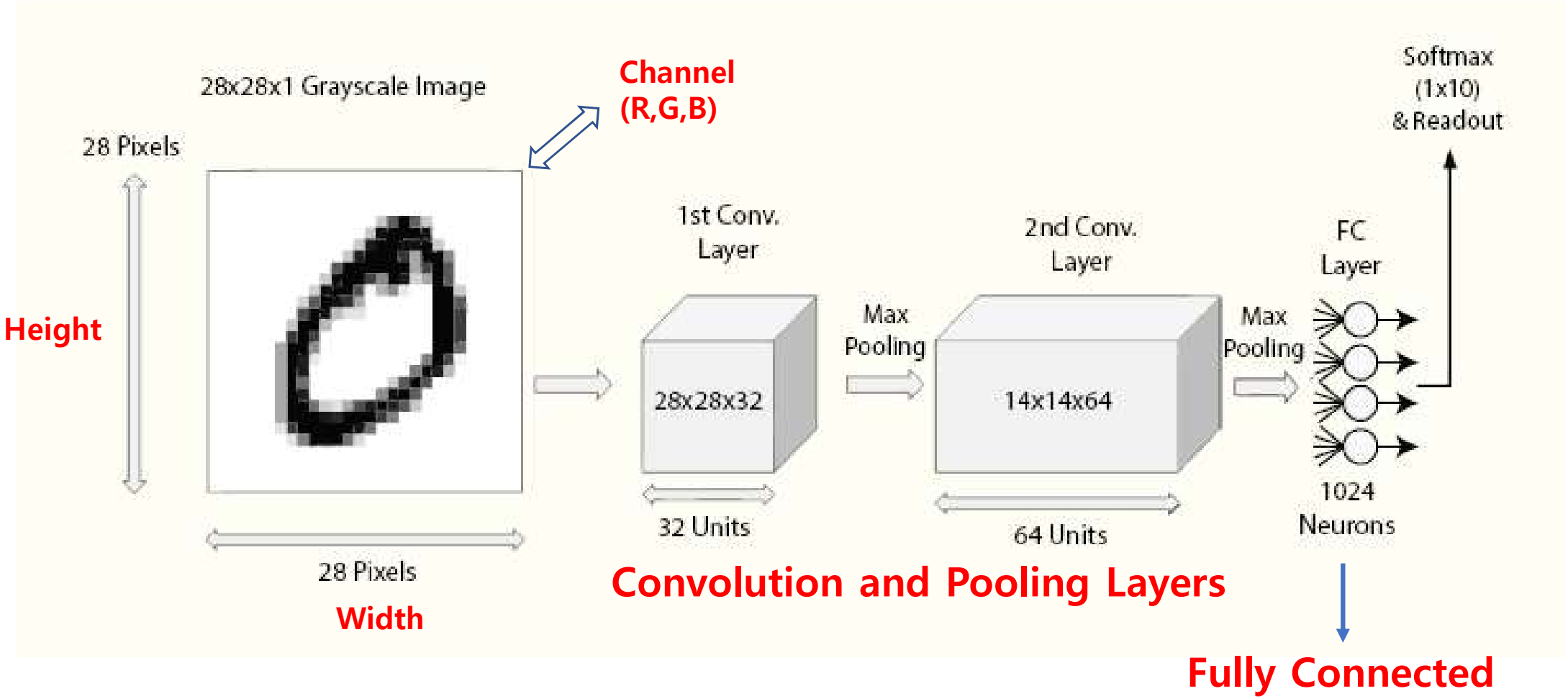
  - Image 정보의 공간적 지역 특성 보존
  - Filter (Kernel) 을 이용한 이미지 특성 추출

- **Pooling Layer (풀링층)**

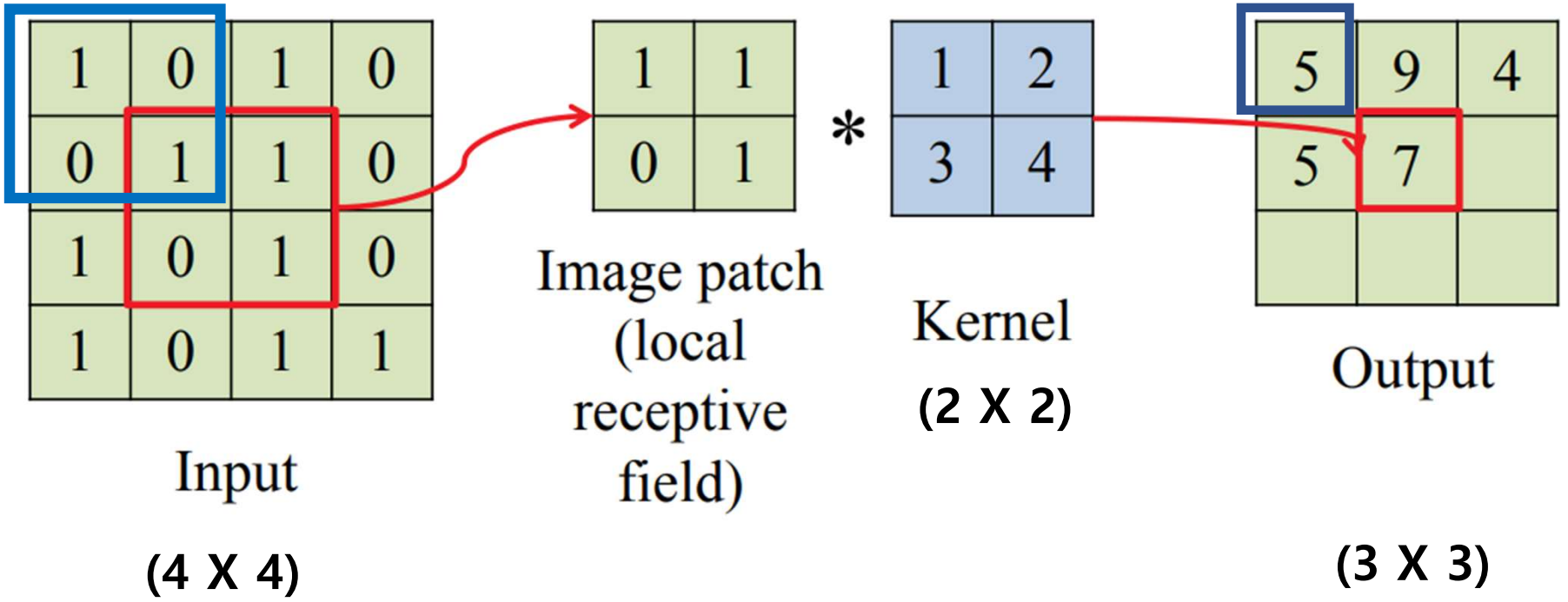
  - Image data 의 정보 손실 없는 압축

  ➔ 계산량 및 메모리 사용량 축소, 파라미터의 수 감소 (과적합 방지)

Dimensions of Layers



How Convolution works ?

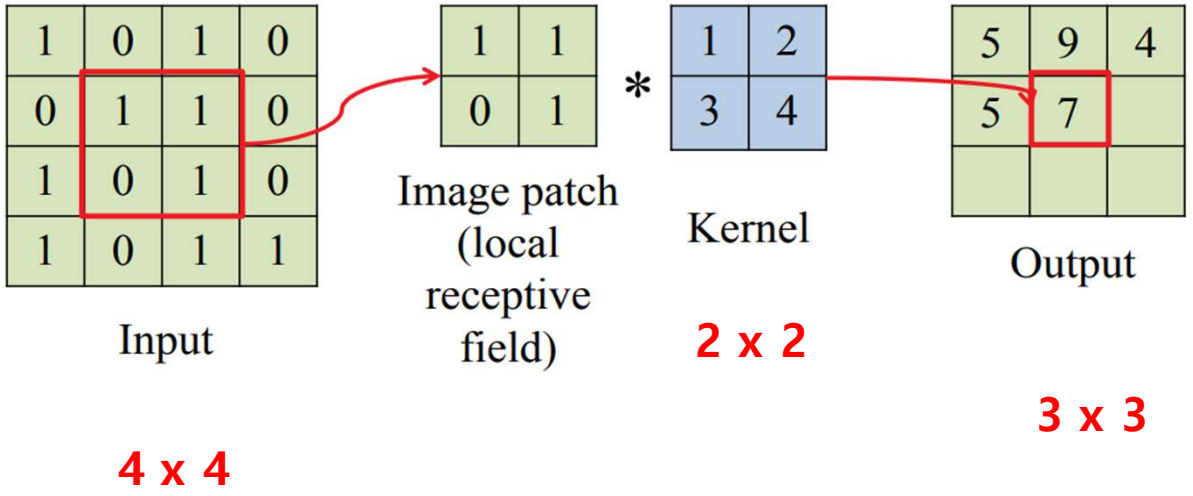


Kernel_size=(2, 2), stride=(1, 1), No padding

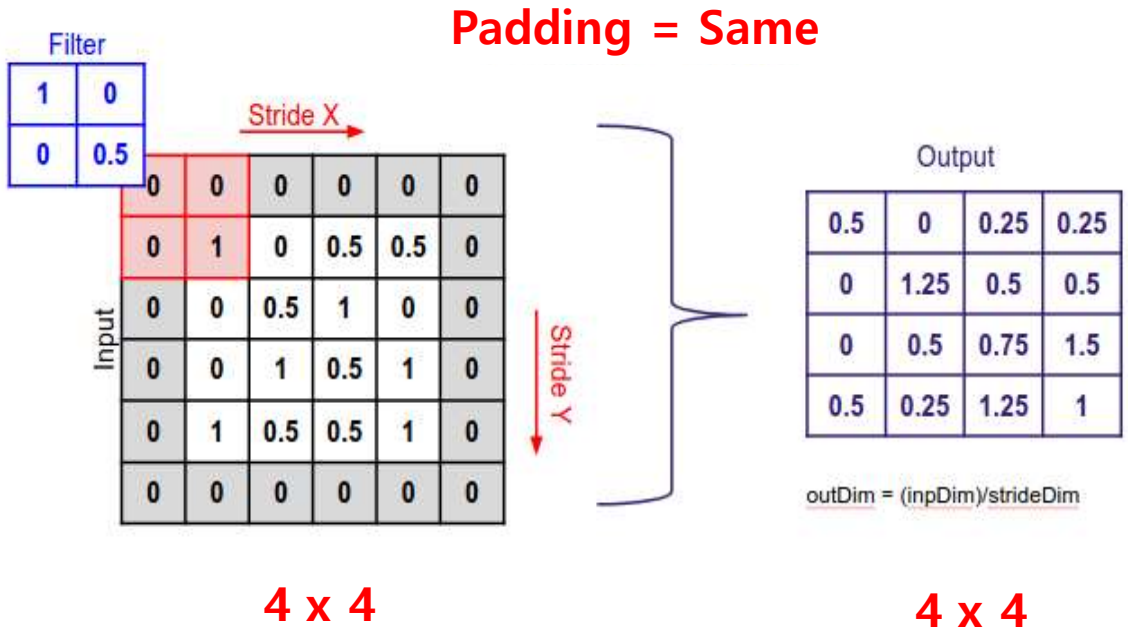
Padding

- Convolution 때 마다 image 의 edge 정보가 소실

Padding = **Valid (No Padding)**

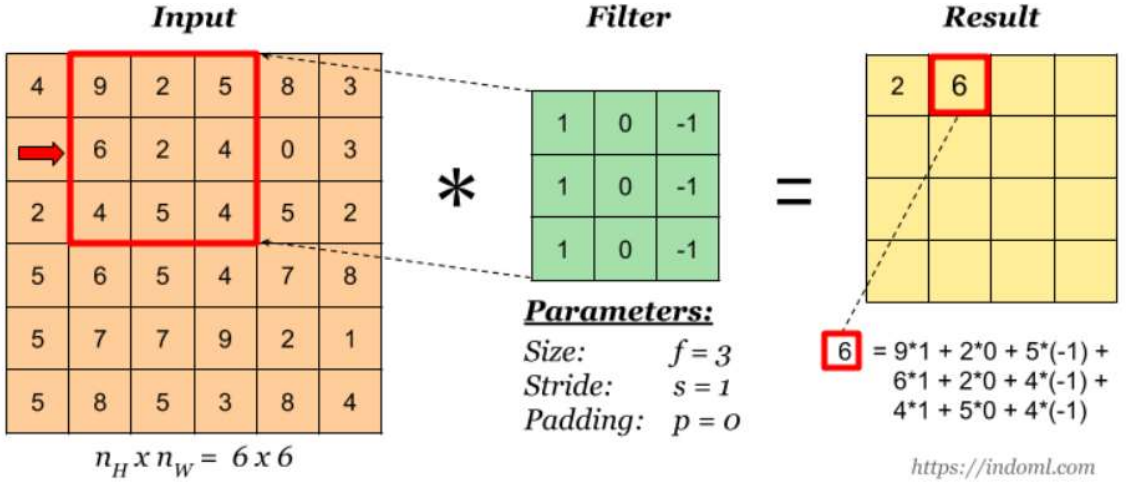


- Input size = output size (➔ Input 의 주위에 0 pixel padding)
- Padding = “same” convolution



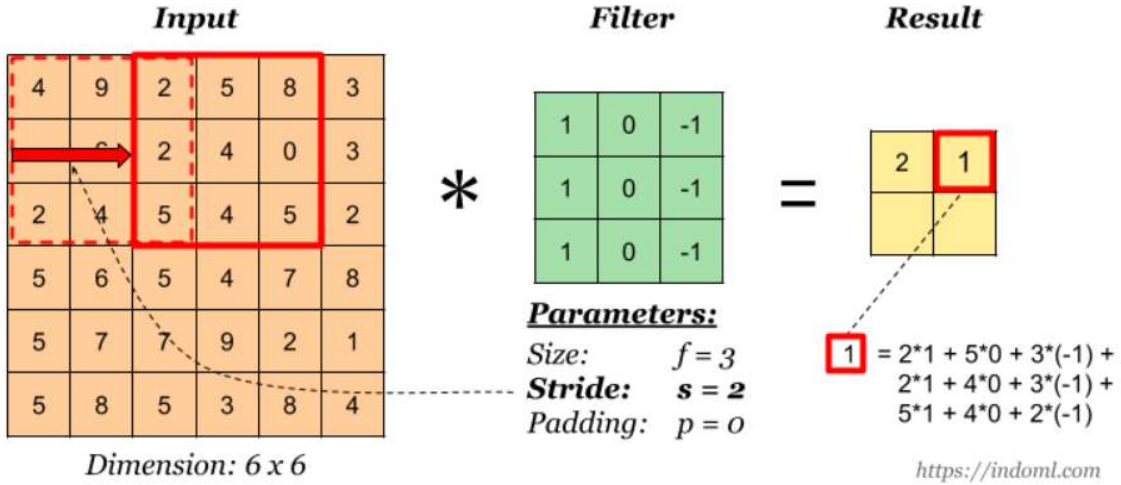
Striding

Stride = 1



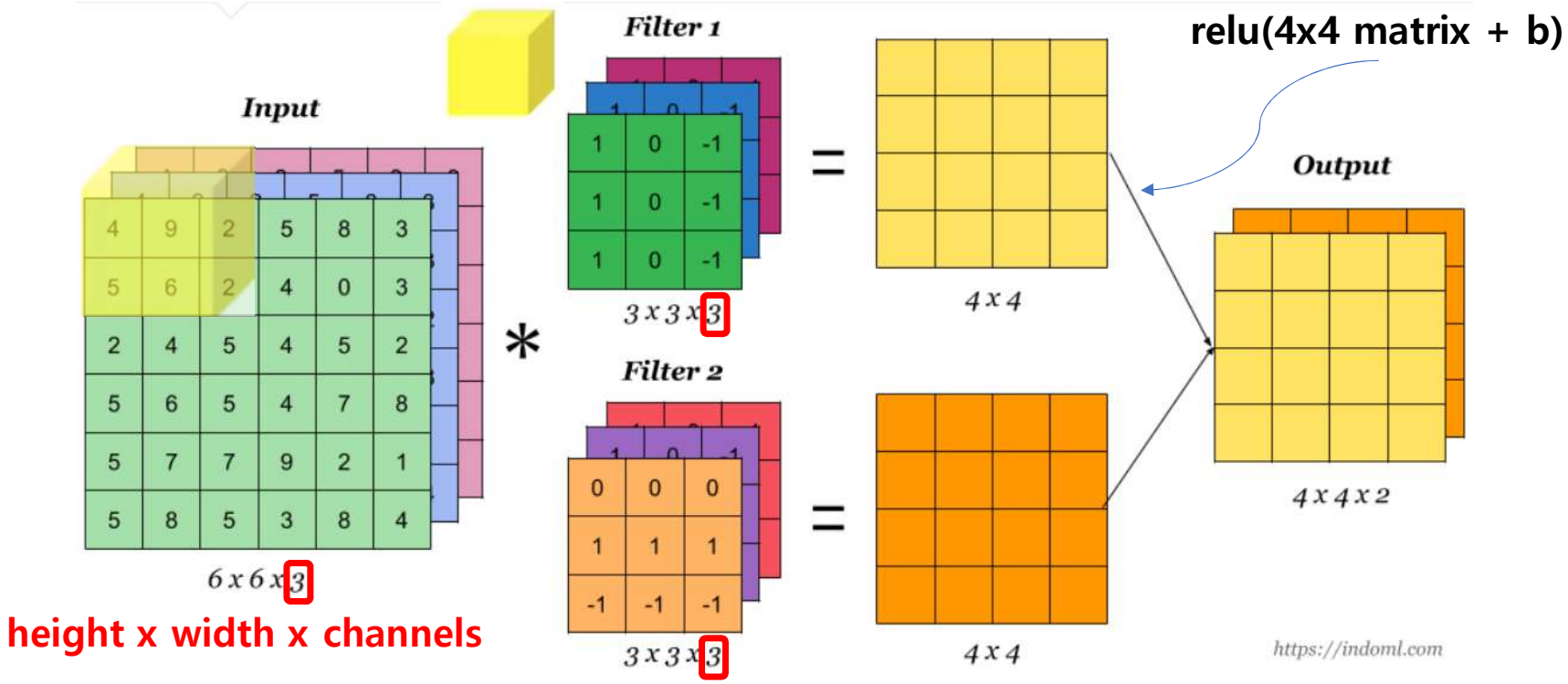
4 x 4

Stride = 2



2 x 2

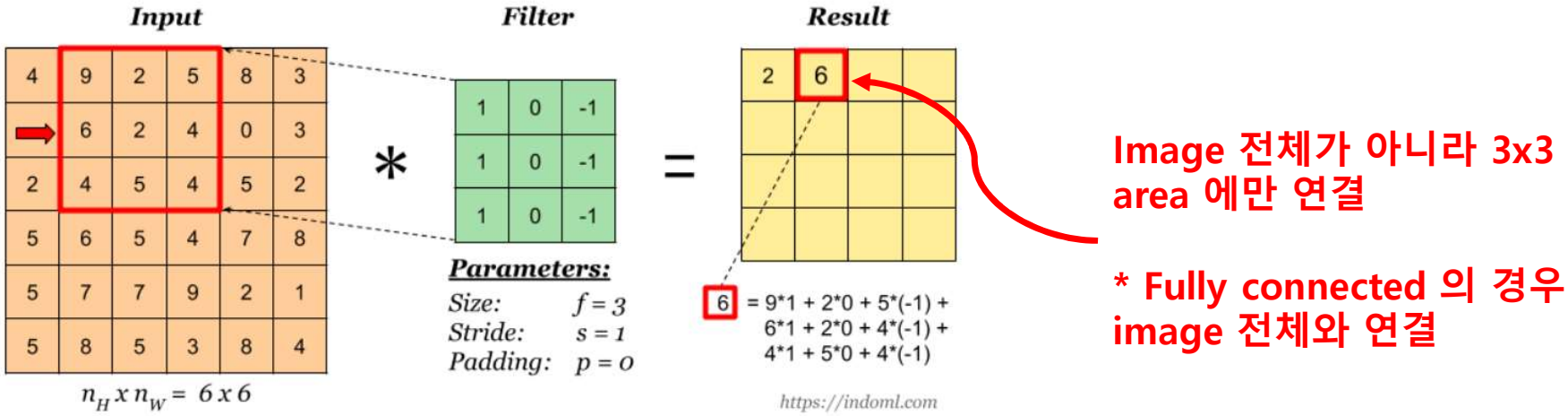
Convolutions over Volumes (RGB images)



Convolution Layer 의 2 가지 특성

1. **Locality**

   - kernel size 만큼의 작은 구역 (patch) 의 인접한 pixel 들에 대한 correlation 관계를 비선형 필터를 적용하여 추출
   - 이러한 필터를 여러 개 적용하면 다양한 local 특징을 추출 가능



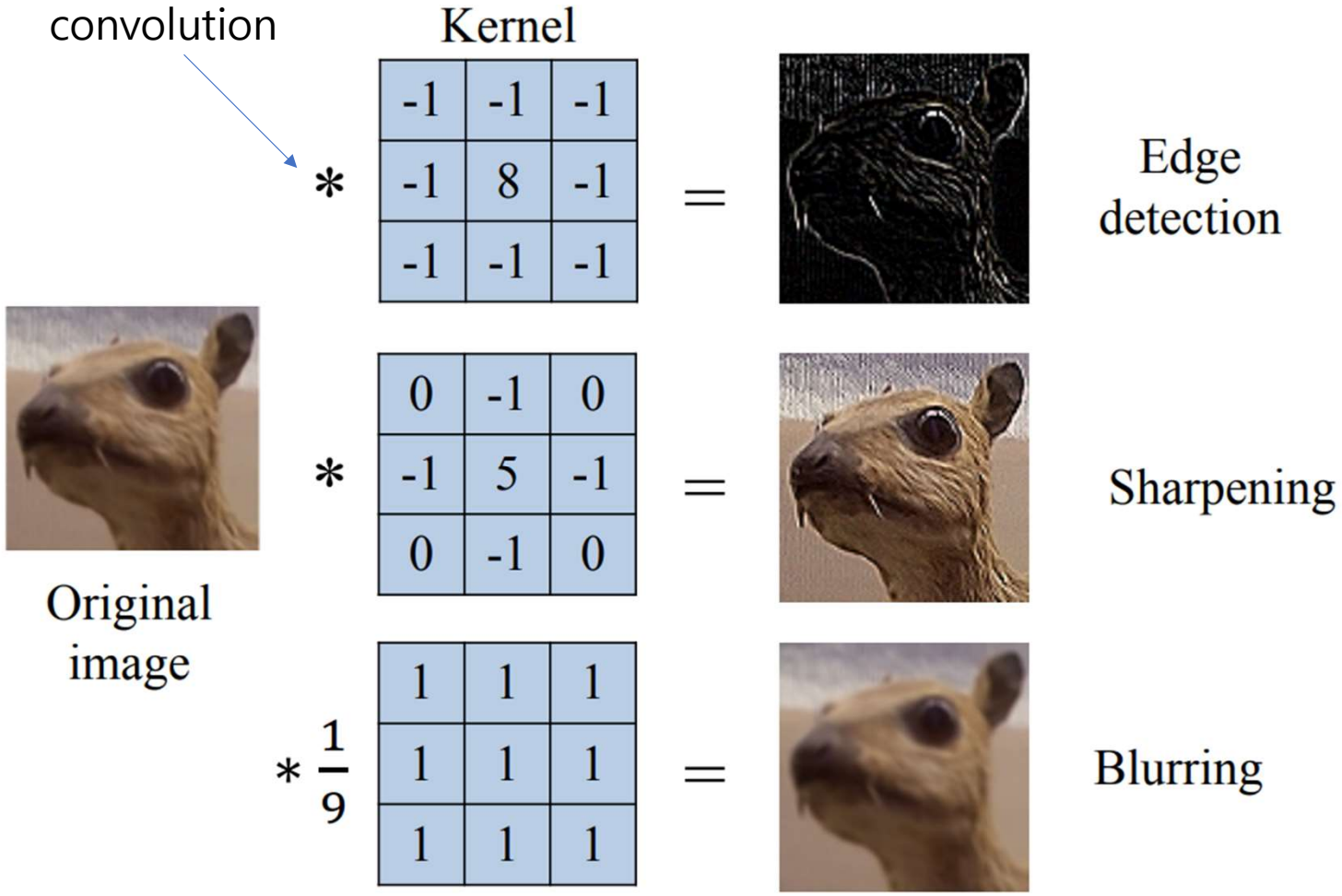
2. **Parameter Sharing**

- input 상의 모든 patch 들은 동일한 kernel 을 적용하여 next layer 의 output 을 출력한다.

  ex) vertical edge detector – image 전체에 동일한 kernel 적용

- Fully connected layer 를 image data 에 사용할 경우에 비해 parameter 의 수를 획기적으로 줄임

Kernel(Filter) 의 특성 추출 예

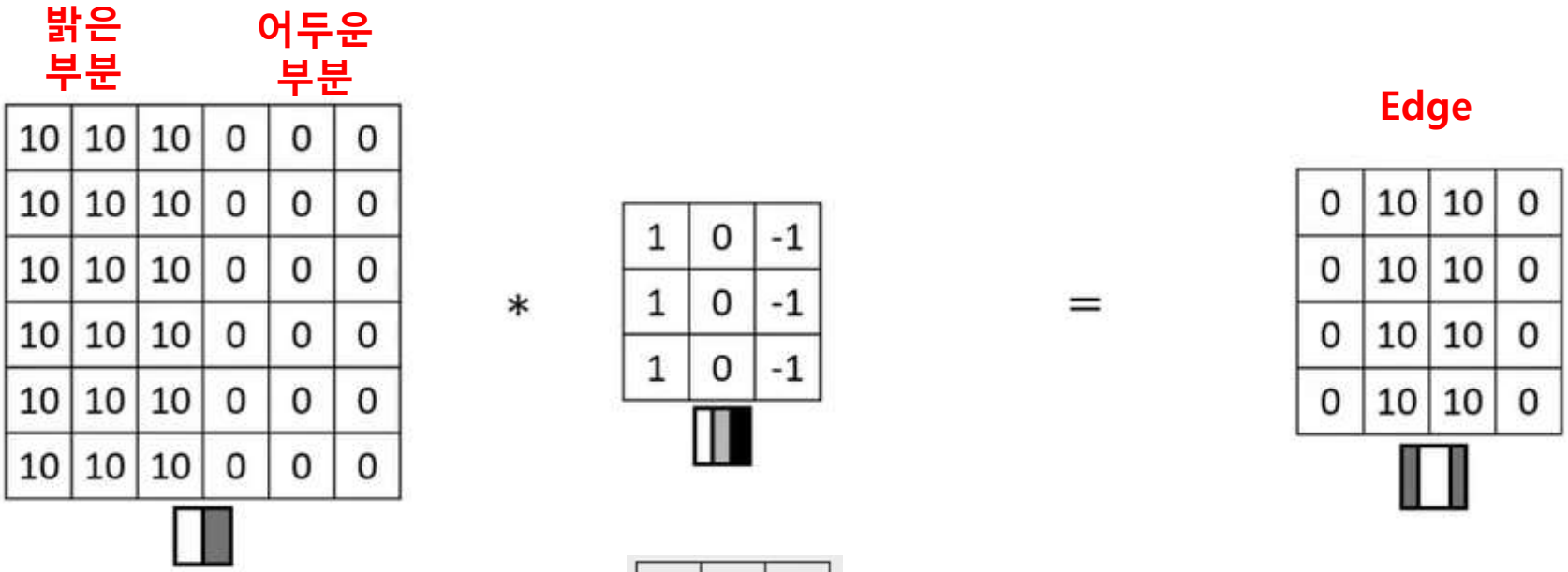


- CNN 이전에는 edge detection filter 를 computer vision 전문 가들이 모두 manually 만들어 줌

  ex) 수직 / 수평 / 45 도 / 명암 구분 filter 등

  ➔ Neural Network 은 훨씬 더 다양한 특성의 filter 들을 back-propagation 을 이용하여 자동으로 학습하고 스스로 만들어 냄

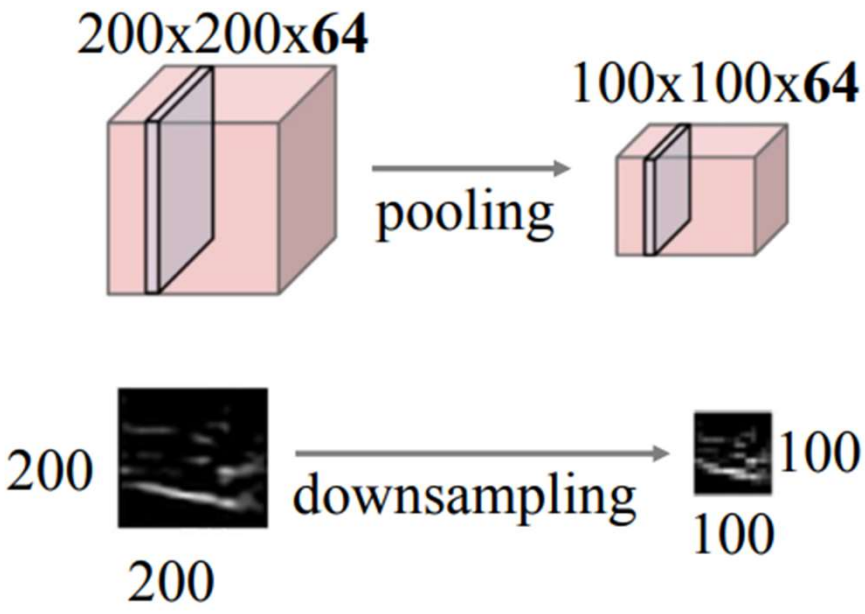
Ex) Edge Detector

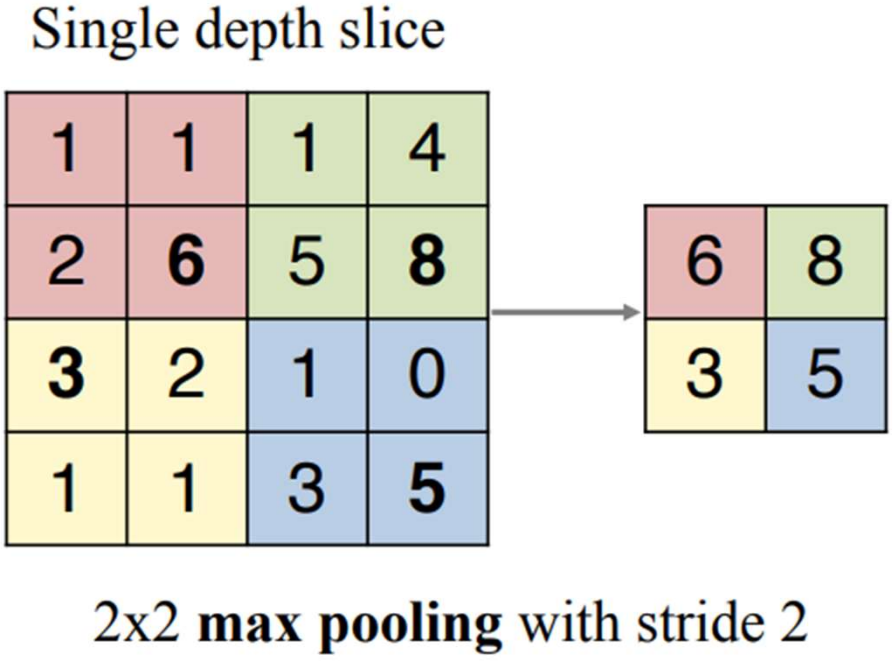


Backpropagation 에 의한 자동 학습

How Pooling works ?

- Pooling 의 뉴런은 가중치가 없음
- 최대, 평균을 이용한 이미지 subsampling (downsampling)





Pooling Layer 의 2 가지 특성

1. Positional Invariance

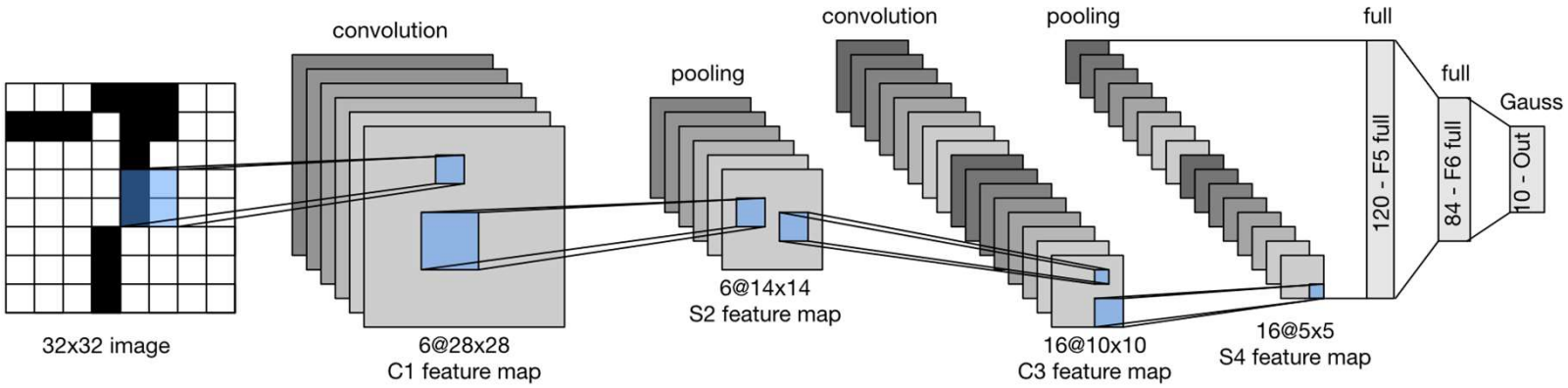
   - 특정 pixel 의 정확한 position 에 덜 민감함
   - 여러 번의 pooling 을 거치면 넓은 영역에 걸쳐 같은 효과 발생
     (See Wider !)

2. Size 축소

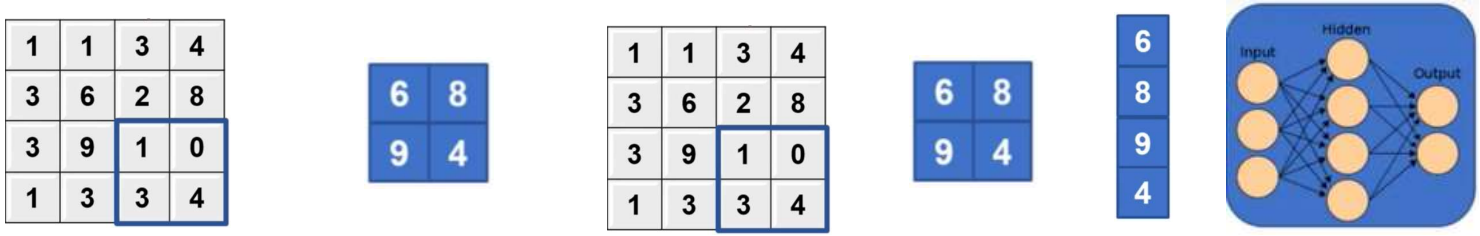
   - 계산량을 크게 줄임
   - 과적합 방지

CNN (Convolutional Neural Network, 합성곱 신경망)

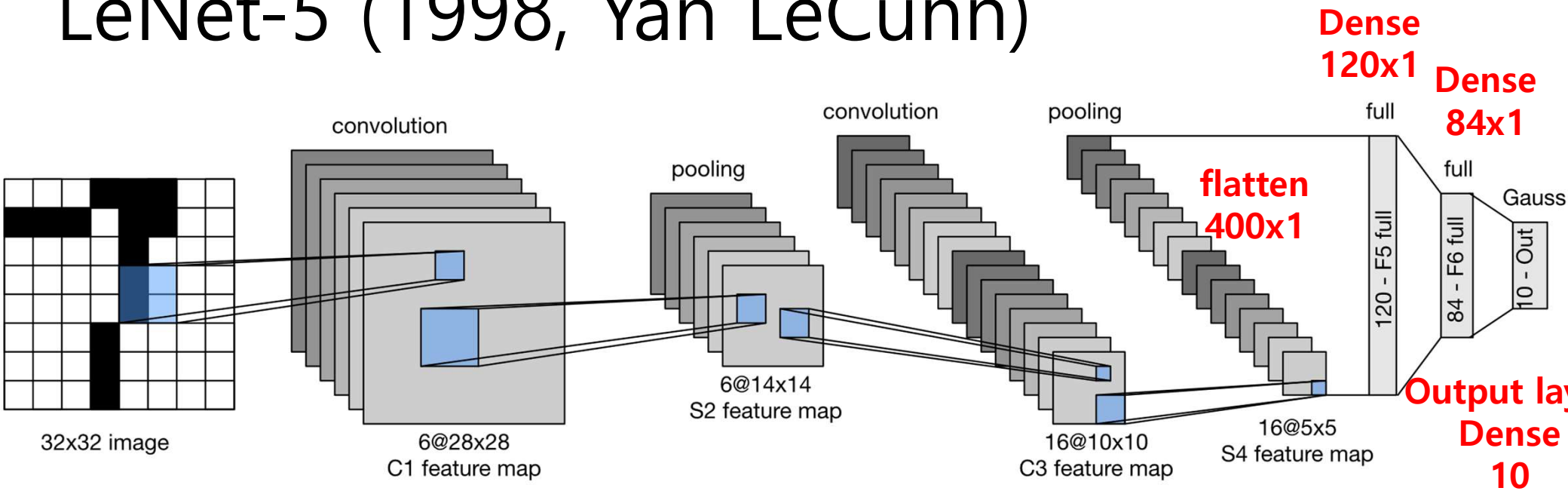
- LeNet : 5 개층 Yan Le Cunn - 1998



입력 **합성곱** **풀링** **합성곱** **풀링** **완전연결**



LeNet-5 (1998, Yan LeCunn)



Dense 120x1

Dense 84x1

Output layer Dense 10

1st layer

2nd layer

Input layer
32 x 32
Gray Scale Image

Convolution
6 filters
(5 x 5,
stride 1)
28 x 28 x 6

Average pooling
Filter
(2 x2, stride 2)

14 x 14 x 6

Convolution
16 filters
(5 x 5,
stride 1)
10 x 10 x 16

Average pooling
Filter
(2 x2, stride 2)

5 x 5 x 16

Famous CNN models

- Alex Net – 2012 년 ILSVRC(ImageNet Large Scale Visual Recognition Competition)대회 우승
- GoogleLeNet(Inception Net) – 2014 년 ILSVRC 대회 우승
- ResNet – 2015 년 ILSVRC 대회 우승 (152 개 층)
- MobileNet – mobile device 용 pre-trained model (ImageNet 20,000 개 classes)
- VGG-16 : Keras built-in pre-trained model (2014 년, 16 layers)

실습 : Le Net 을 이용한 손글씨 인식

1. Keras 를 이용한 Le Net 구축

2. 구축한 Le Net model 을 이용하여 Mnist 손글씨 분류

실습 : Deeper CNN 을 이용한 CIFAR-10 분류

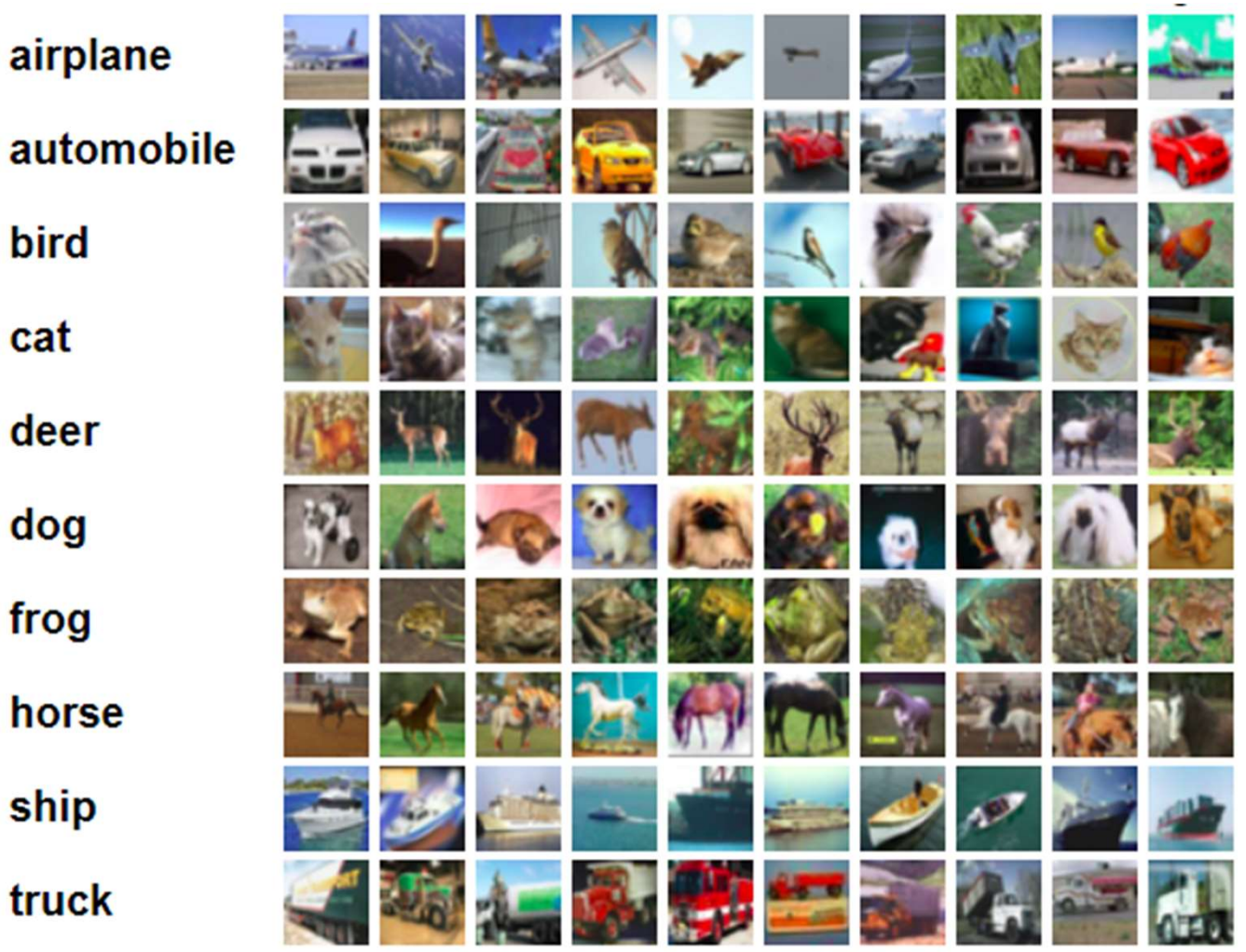
1. CIFAR-10 dataset 은 32x32 color image 를 가진 10개의 class (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck)

2. 각 class 별 6,000 개씩 total 60,000 개 image

3. Image 가 blur 하여 난이도 높음
   (최근 성적 : https://en.wikipedia.org/wiki/CIFAR-10)

4. Google Colab GPU 환경 이용

CIFAR-10 image dataset



Neural Network Architecture

Input layer
32 x 32 x 3

Conv2D
16 filters
(3, 3)
Padding=same
Activation : relu

MaxPooling2D
(2, 2)
Padding=same

Dropout : 0.2

Conv2D
32 filters
(3, 3)
Padding=same
Activation : relu

MaxPooling2D
(2, 2)
Padding=same

Dropout : 0.2

Conv2D
64 filters
(3, 3)
Padding=same
Activation : relu

MaxPooling2D
(2, 2)
Padding=same

Dropout : 0.2

Flatten

Dense
256
Activation : relu

Dropout : 0.5

Output

Dense
10
Activation : softmax

What is JSON / YAML ?

- JSON (**J**ava**S**cript **O**bject **N**otation)

```
{
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
    ]
}
```

- YAML (YAML Ain't Markup Language)

```
invoice: 34843
  date : 2001-01-23
  bill-to: &id001
  given : Chris
  family : Dumars
  address: lines: | 458 Walkman Dr. Suite #292
  city : Royal Oak
  state : MI
  postal : 48046
```

실습 : 구축한 model 의 weight 와 architecture 저장 및 loading

1. JSON / YAML file 로 model architecture 저장

2. JSON / YAML file 로 부터 model 복원

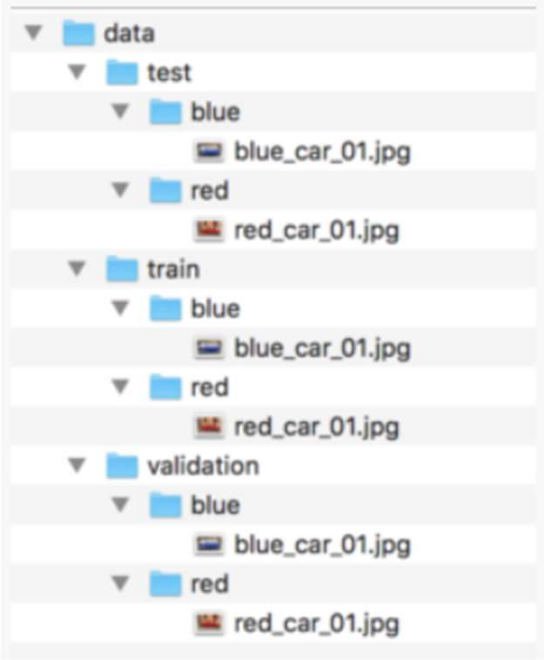
3. HDF5 file 로 model 의 parameter 저장 및 복원

ImageDataGenerator methods

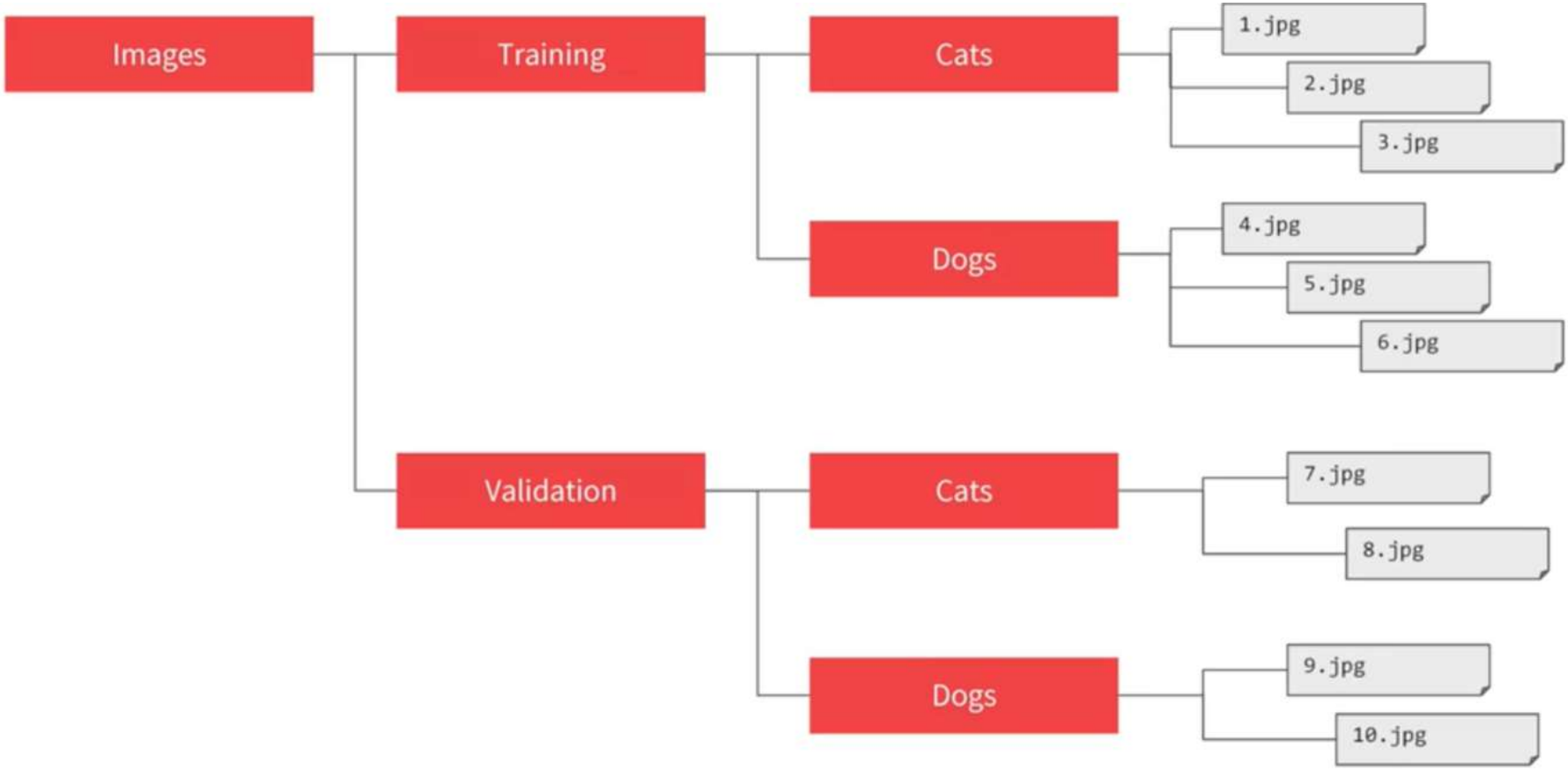
- .flow (X_train, y_train, batch_size) : augmented data 반환

- .flow_from_directory
  - 대용량 data 를 directory 에서 직접 load (with data augmentation)
  - directory 구조에 의해 자동으로 label 인식

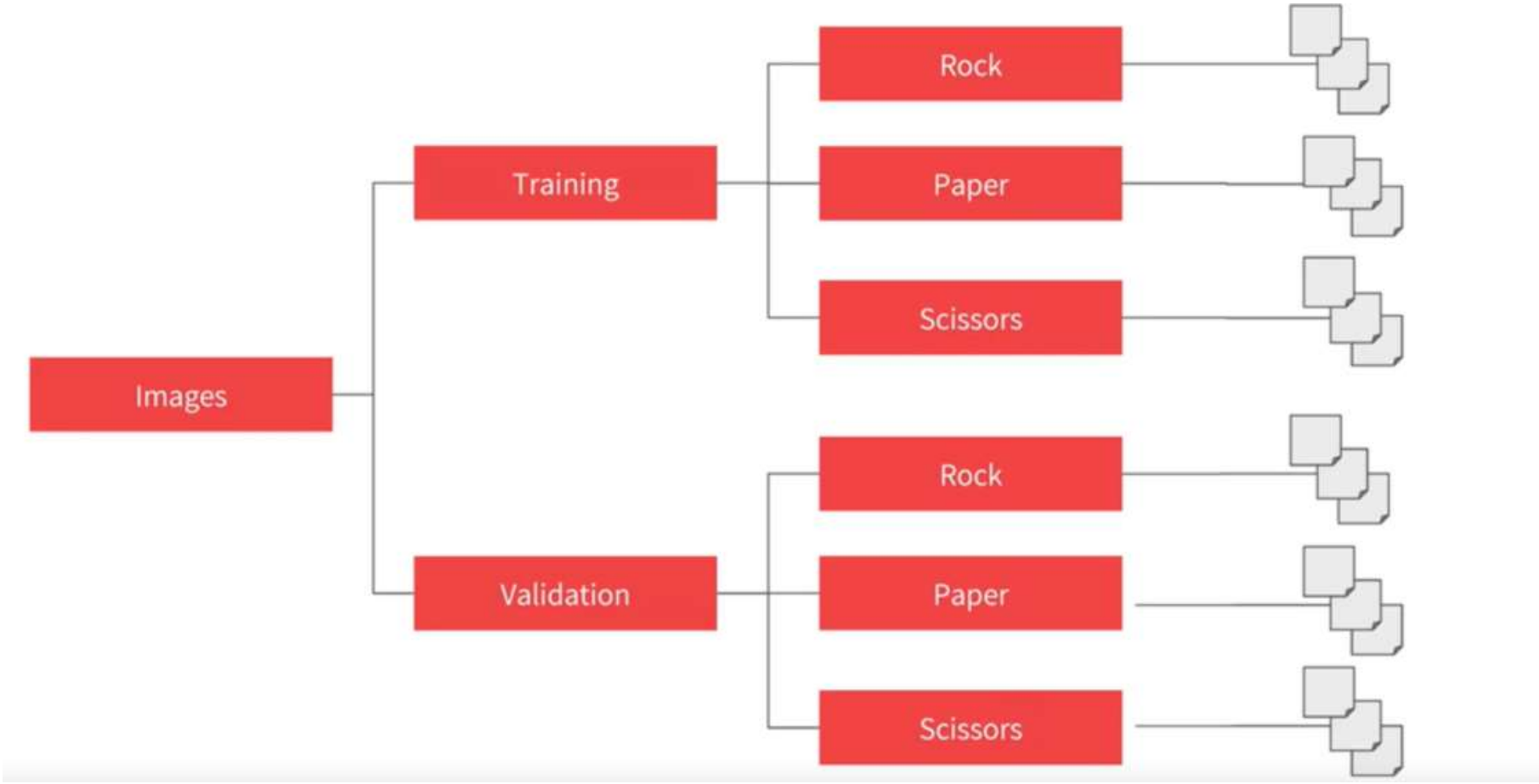
- .flow_from_dataframe
  - 모든 image 가 하나의 folder 에 있는 경우



ImageDataGenerator folder structure



ImageDataGenerator folder structure



flow_from_directory 사용법

- from tensorflow.keras.preprocessing.image import ImageDataGenerator

- Instance 생성 :

  train_data_gen = ImageDataGenerator(rescale=1/255.)

- flow_from_directory method 호출 :

  train_generator = train_data_gen.flow_from_directory(
  train_dir,
  target_size(150, 150), ➔ image size 통일
  batch_size=20,
  class_mode='binary' or 'categorical)

flow_from_directory 사용법

- history = model.fit_generator(
  train_generator,
  step_per_epoch=100, ➔ total image 개수 / batch_size
  epochs=15,
  validation_data=validation_generator,
  validation_steps=50,
  verbose=2)

Data Augmentation

- 목적
    1. 부족한 training data increase
    2. overfitting 방지

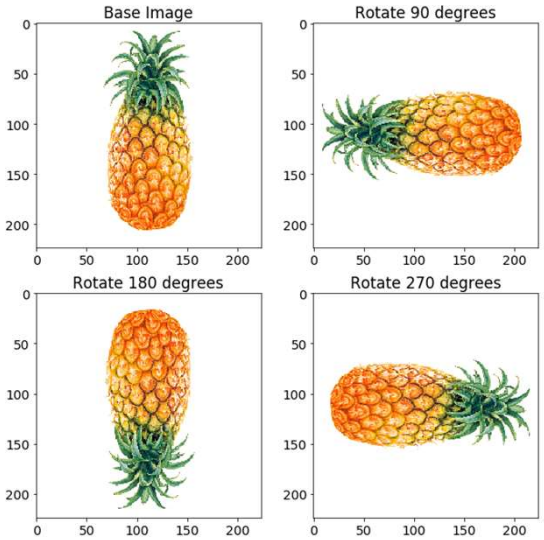
- Keras.preprocessing.image 의 ImageDataGenerator 클라스 이용

- ImageDataGenerator
  – flow, flow_from_directory, flow_from_dataframe method 사용

- model.fit_generator 를 이용하여 train

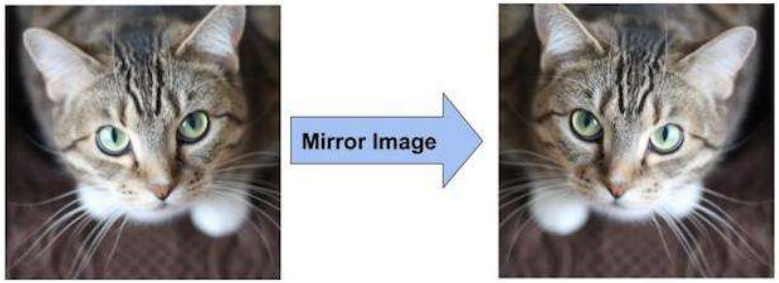
- Train, Validation dataset 에 모두 적용.

Data Augmentation



Rotation

Mirroring



Shearing

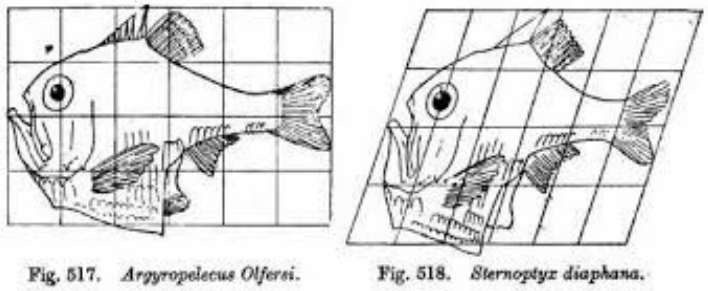
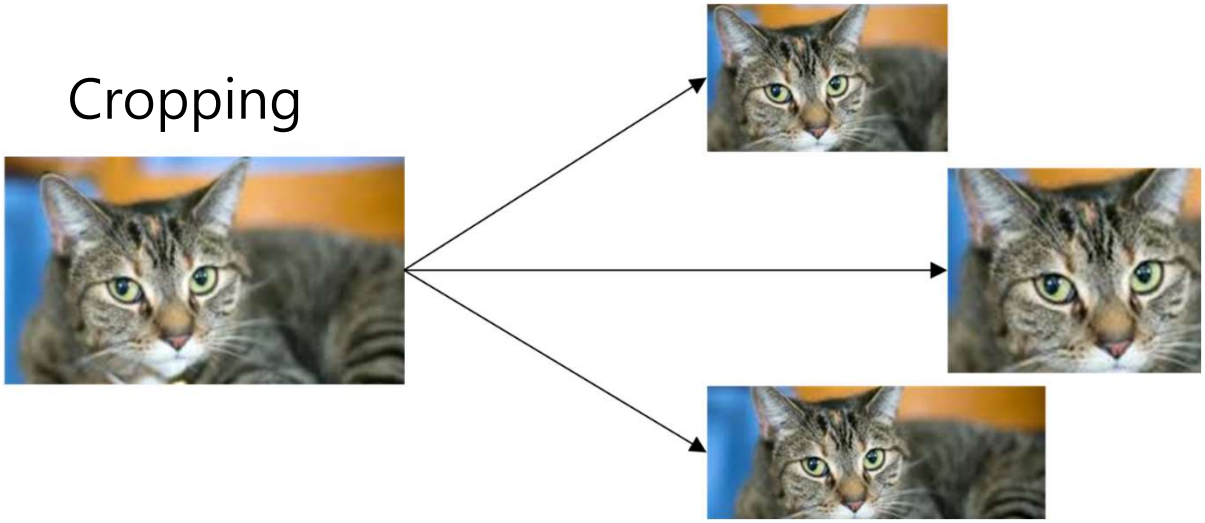


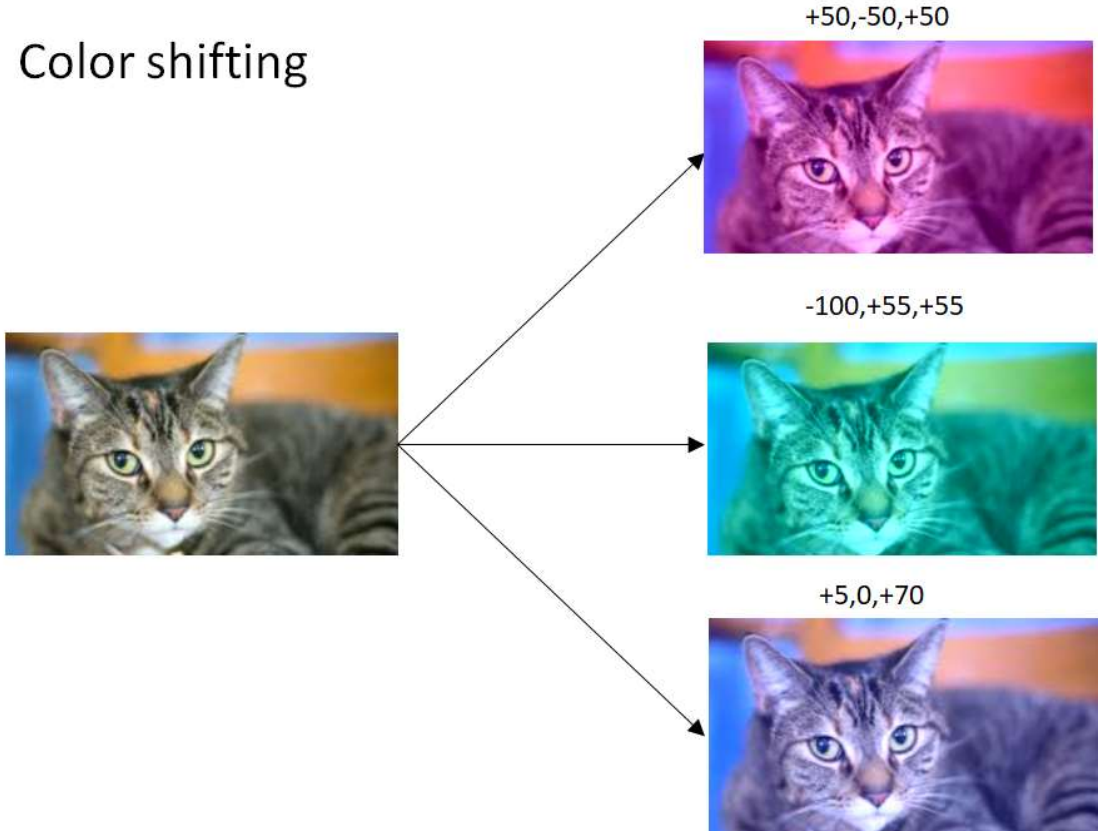
Fig. 517. *Argyropelecus Olfersi.* Fig. 518. *Sternoptyx diaphana.*

Cropping



Data Augmentation

Color shifting



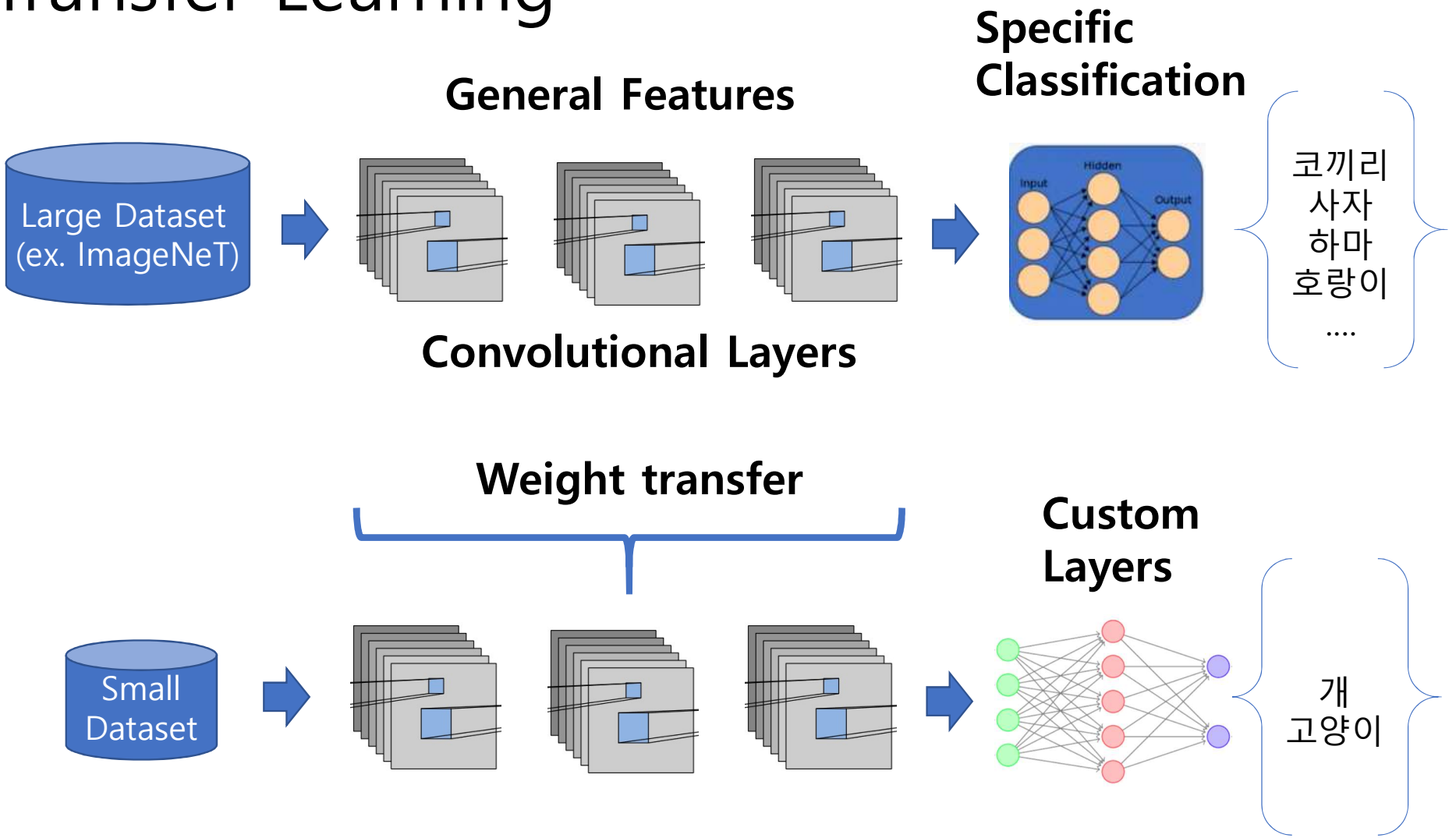
zoom

Horizontal Flip

ImageDataGenerator parameters

- train_dataget = ImageDataGenerator(
  rescale=1./255,
  rotation_range=40, ➔ 0~40 도 사이에서 random 회전
  width_shift_range=0.2, ➔ 사진 중심 20% 이동
  height_shift_range=0.2,
  shear_range=0.2 ➔ 20% shear
  zoom_range=0.2 ➔ 20% 까지 randomly zoom
  horizontal_flip=True,
  fill_mode='nearest') ➔ 생성된 공간은 인접 pixel 로 채움

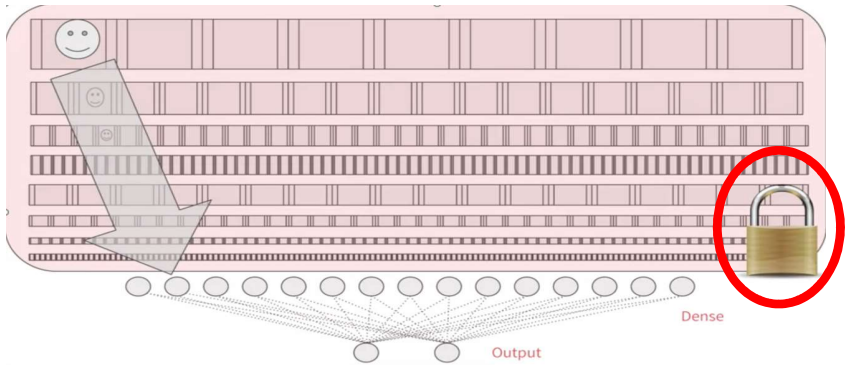
Transfer Learning



Transfer Learning Strategy

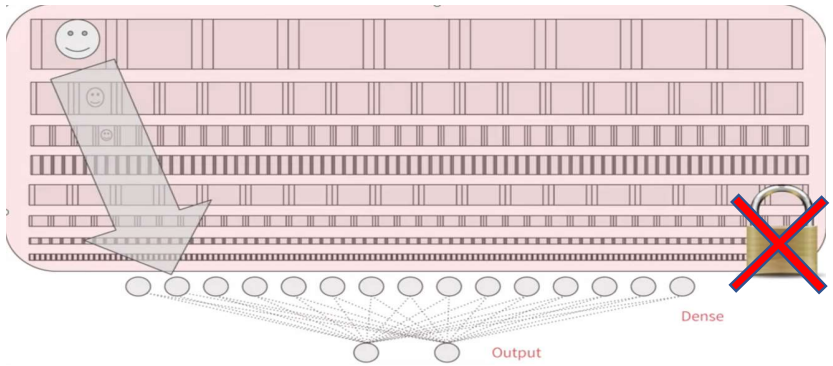
- Strategy 1

CNN layer 는 freeze 하고,
추가한 완전연결층만 새로이 train



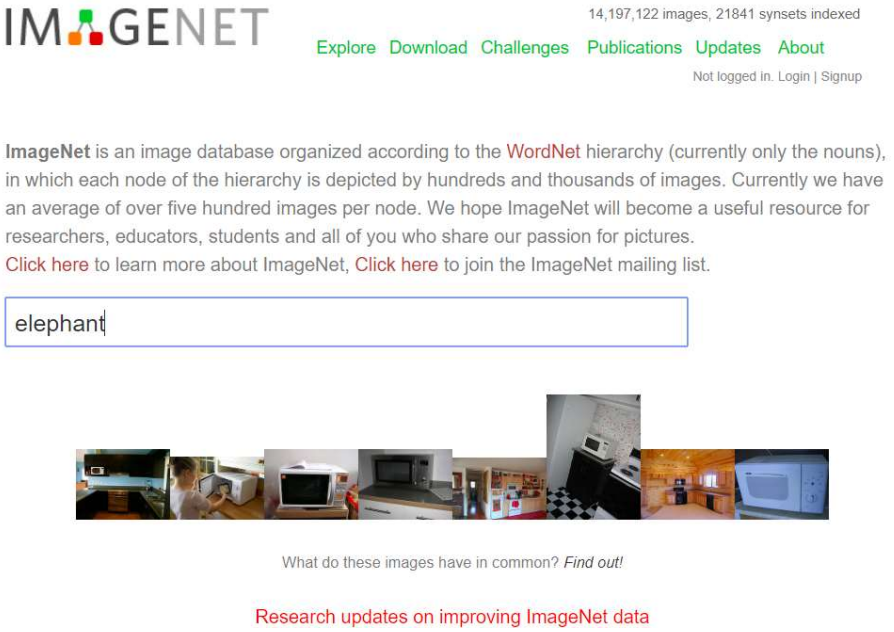
- Strategy 2

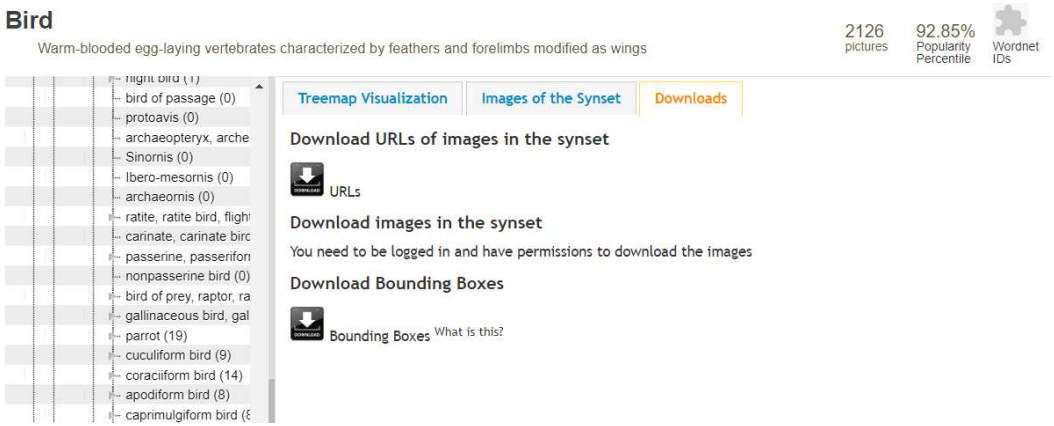
전체 Layer 를 매우 작은 learning rate 로 re-train



ImageNet

- Open Source Image Repository http://www.image-net.org/
- 1000 classes over 1.5 MM images





Transfer Learning 고려사항

- **목적에 맞는 dataset 선택**

  ex) Cat & Dog 구분 → ImageNet 에 포함

  Cancer cell 구분 → ImageNet 에 없음

- **보유 Data 의 Volume 고려**

  1. 모든 weight 새로이 training (Large Data 보유)
  2. Weight 의 일부만 training
  3. 마지막 layer 만 Fine-tuning (Small Data 보유)

실습 : Pre-trained model 을 이용한 image 분류

1. Keras 에 내장된 ResNet50 pre-trained model 을 이용

```
base_model = tf.keras.applications.ResNet50(weights = 'imagenet', include_top = False)
```

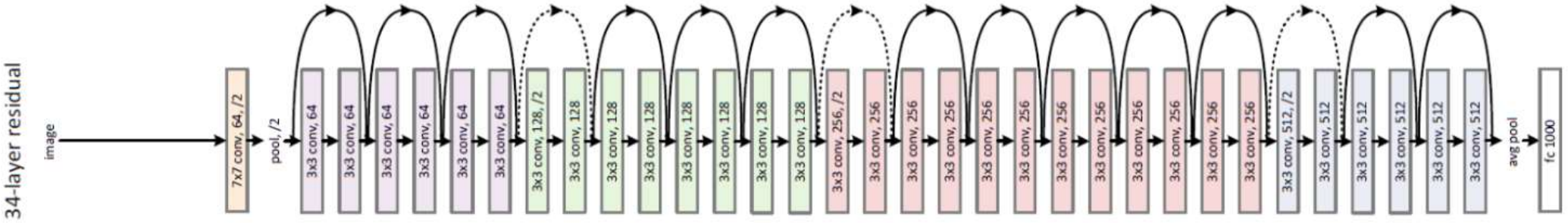
2. 임의의 image 를 ResNet50 의 입력 spec 에 맞도록 resize

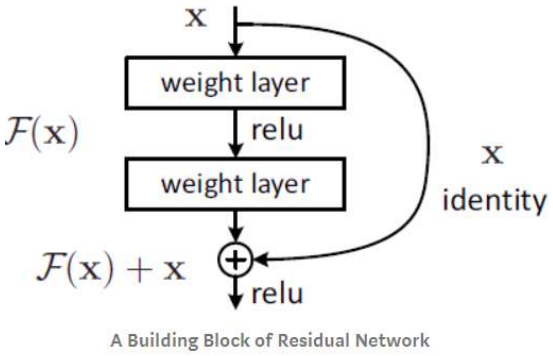
```
tf.keras.applications.resnet50.preprocess_input(Sample_Image)
```

3. decode_predictions 를 이용한 결과값 비교

```
tf.keras.applications.resnet50.decode_predictions(predictions, top = 5)[0]
```

ResNet Structure





A Building Block of Residual Network

- vanishing/exploding gradients 문제 해결을 위해 skip / shortcut connection 을 추가
- input *x* 를 몇 개 layer 의 output 에 추가

Deep Learning

Sequence Model

What is sequence data ?

- Speech recognition : 파동의 연속 → 단어의 연속으로 변환
- Music generation : 연속된 음표 출력
- Sentiment classification : Text → 평점, 부정/긍정 판단
- DNA 분석 : 염기서열 → 질병유무, 단백질 종류 등
- 자동 번역 : 한국어 → 영어
- Video activity recognition : 연속된 장면 → 행동 판단
- Financial Data : 시계열자료 → 주가, 환율 예측 등

Problem of Standard Neural Network

- 입력의 길이와 출력의 길이가 고정되어 있음

  → 따라서 standard NN 에서는 maximum input length 정하고
  초과하면 truncate, 모자라면 padding 을 해 주어야 한다.

- 입력 데이터의 순서를 무시 (모든 observation 은 독립적임)

RNN (Recurrent Neural Network)

RNN (Recurrent Neural Network)

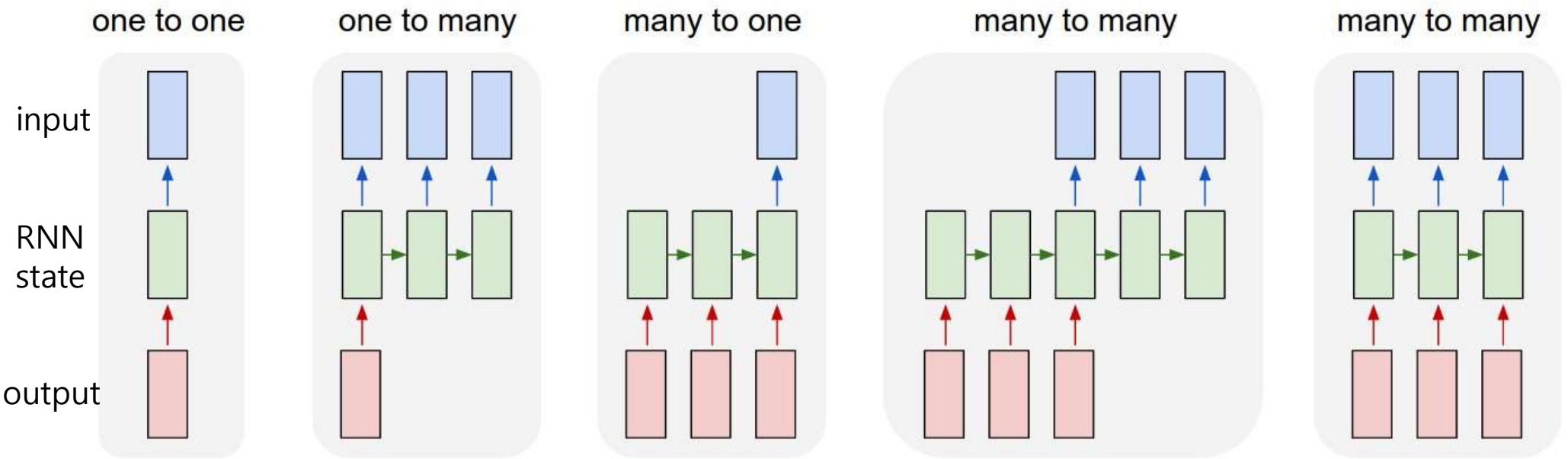
- 시퀀스 데이터에 특화

- '기억' 능력을 갖고 있음
  * 네트워크의 기억 - 지금까지의 입력 데이터를 요약한 정보
  (새로운 입력이 들어올때 마다 네트워크는 자신의 기억을 조금씩 수정)

- 입력을 모두 처리하고 난 후 네트워크에게 남겨진 기억은 시퀀스 전체를 요약하는 정보
  (사람의 시퀀스 정보 처리 방식과 비슷, 기억을 바탕으로 새로운 단어 이해)

- 이 과정은 새로운 단어마다 계속해서 반복 → Recurrent (순환적)

Different Types of RNN



예) 이미지 분류

- 이미지로 부터 문장 생성 (Image Captioning)
- 작곡, 작시

- 감성 분석 (sentiment Analysis → positive/negative)
- Spam detection

- 기계 번역 (영어 → 한국어 문장)
- Chatbot
- Question Answering

- video 각 frame 에 label 생성
- 품사 tagging
- 개체명 인식 등

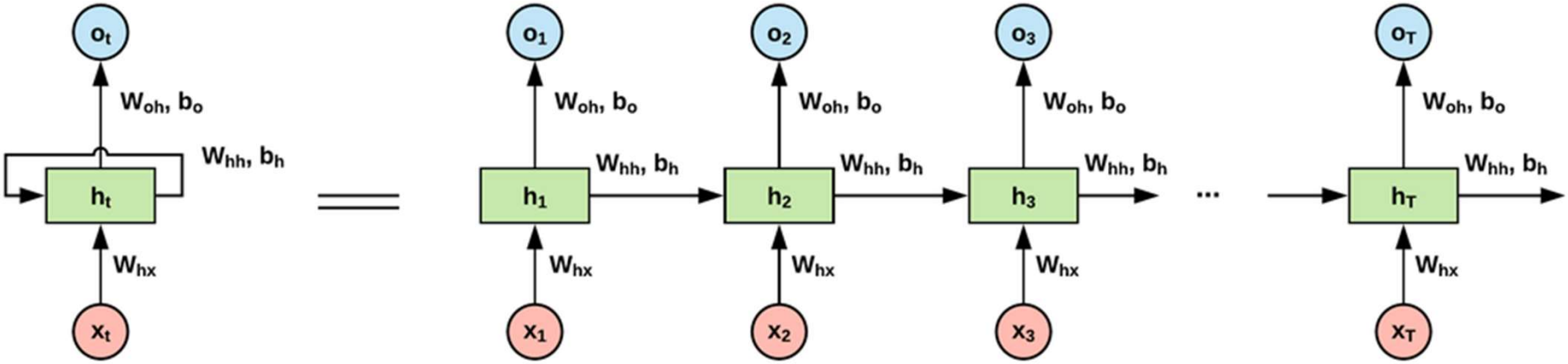
RNN (Unfold 표시)

Internal State :

$$\boldsymbol{h_t = \tanh(W_h h_{t-1} + W_x x_t)}$$

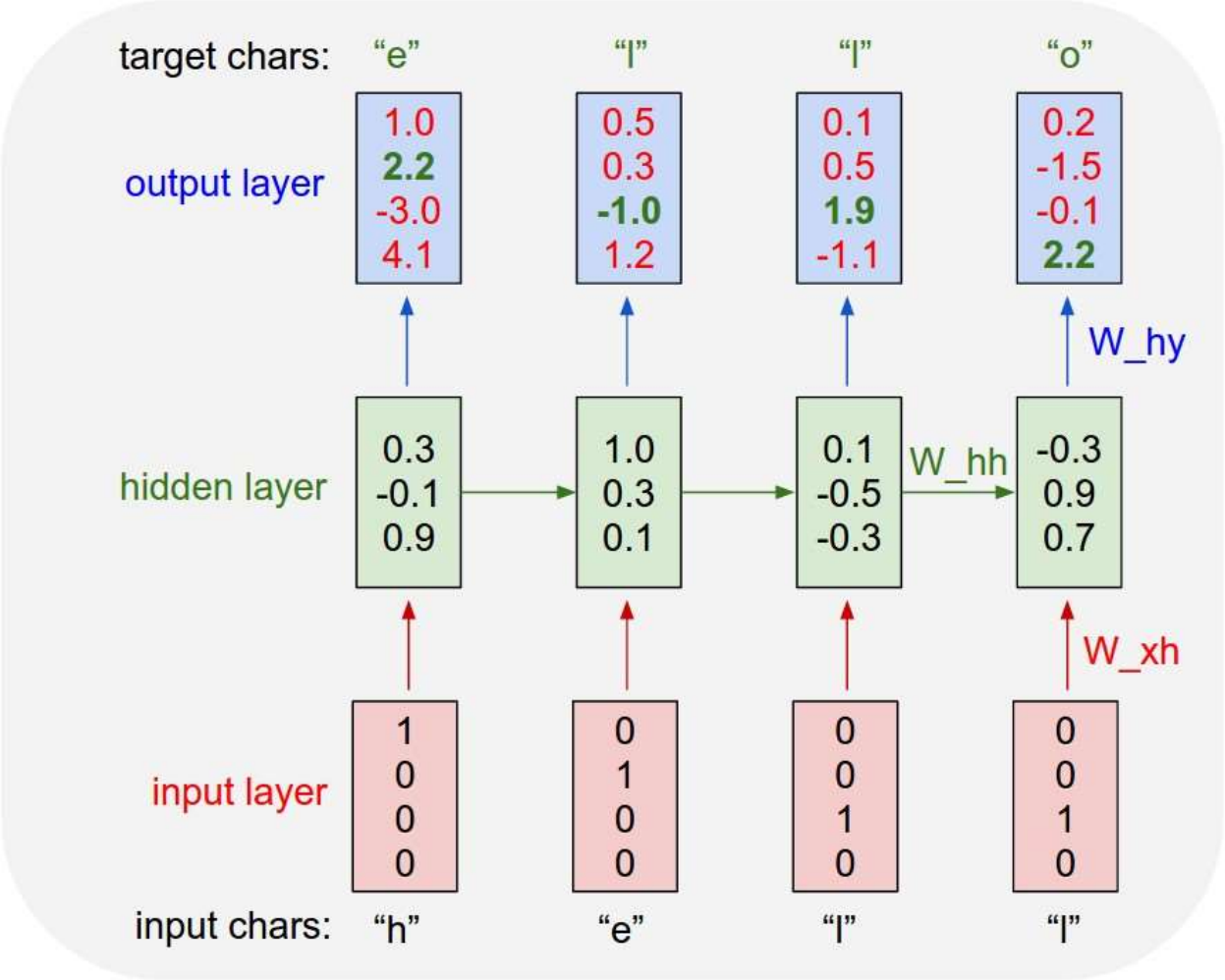
Output :

$$\boldsymbol{O_t = softmax(W_o h_t)}$$



- RNN 을 순서대로 펼쳐 놓으면 weight 를 공유하는 매우 deep 한 neural network 이 된다.
- BPTT (Backpropagation Through Time) 으로 parameter 학습

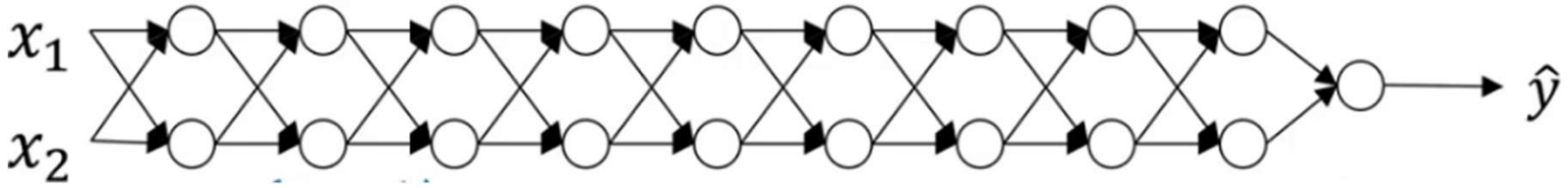
How RNN is trained ?



- "h", "e", "l", "o" 4 개 문자만 있다고 가정
- 각 문자를 One-hot encoding
- "h" 를 시작 문자로 주면 "hello" 가 출력 되도록 훈련
- 각 time step 의 target character 는 "hello" 내의 next character
- Gradient descent 와 backpropagation 을 통해 output layer 의 target score 가 증가되도록 함 (green color)
- "l" 다음의 character 는 현재의 "l" 만으로 판단할 수 없음 (history 필요)

Vanishing & Exploding Gradient

- 매우 deep 한 network 을 훈련시킬 경우 앞부분 Layer weight 의 미분값 크기가 매우 작아지거나 커지는 현상



$$\hat{y} = w^{[L]}\, w^{[L-1]}\, w^{[L-2]} \ldots\ldots\ldots\ldots\ldots\ldots\ w^{[3]}\, w^{[2]}\, w^{[1]} \mathrm{X}_1$$

if $w^{[1]} = 1.5 \rightarrow 1.5^L$ (Explode)
if $w^{[1]} = 0.5 \rightarrow 0.5^L$ (Vanish)

- 문제점 – Exploding – 훈련 자체가 안됨
  Vanishing – 경사하강법이 매우 느리게 진행

Solutions of Vanishing Gradient

- Relu 사용

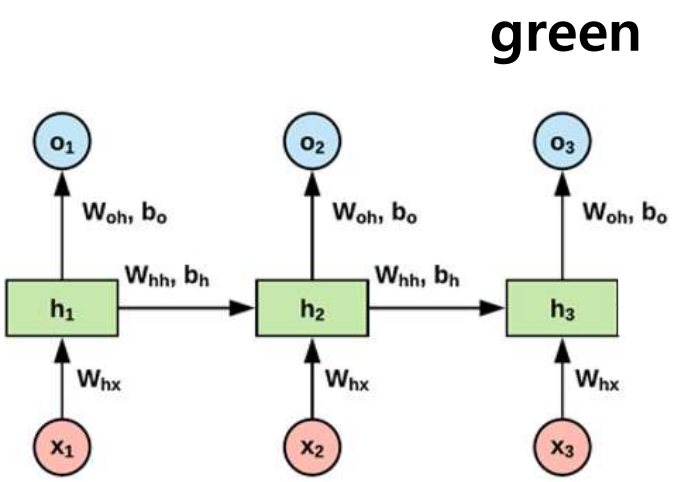
- Weight 의 신중한 초기화

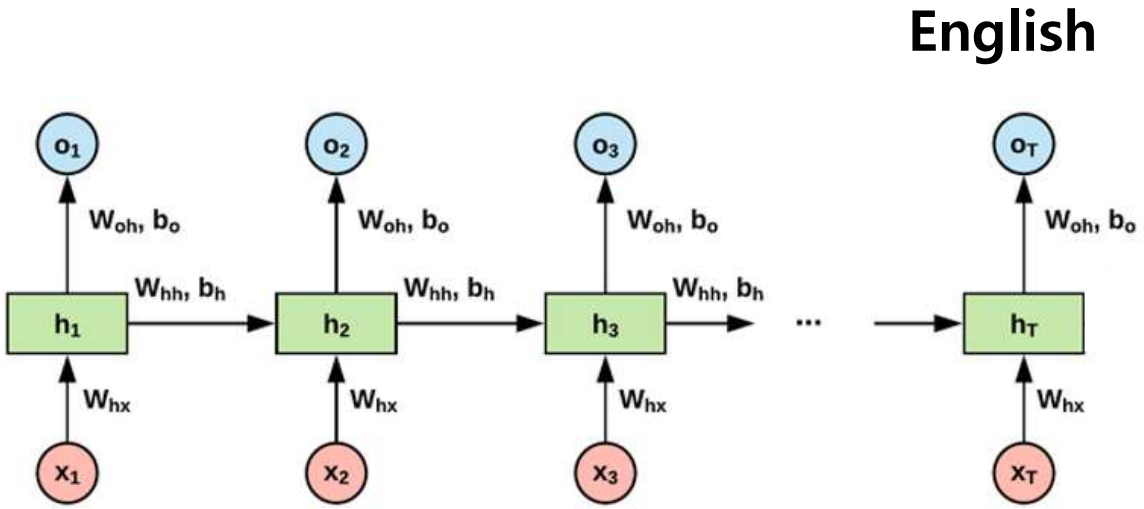
  Ex) Xavier (Glorot) Initializer with Tanh, He Initializer with Relu

- LSTM (Long Short Term Memory) /
  GRU (Gated Recurrent Unit) 사용

  ➔ Vanishing Gradient + memory

LSTM (Long Short-Term Memory)

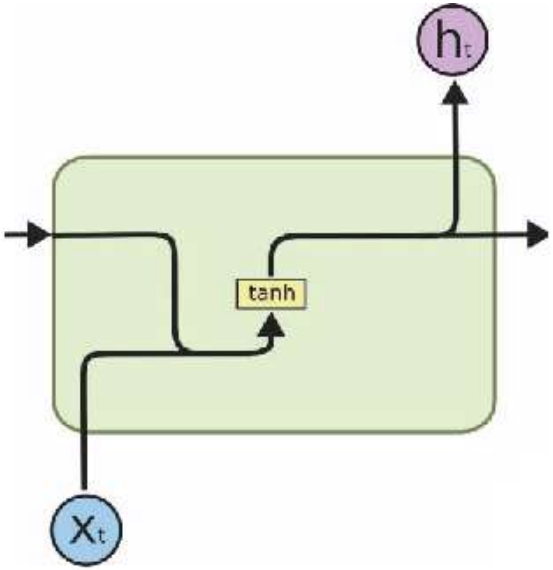




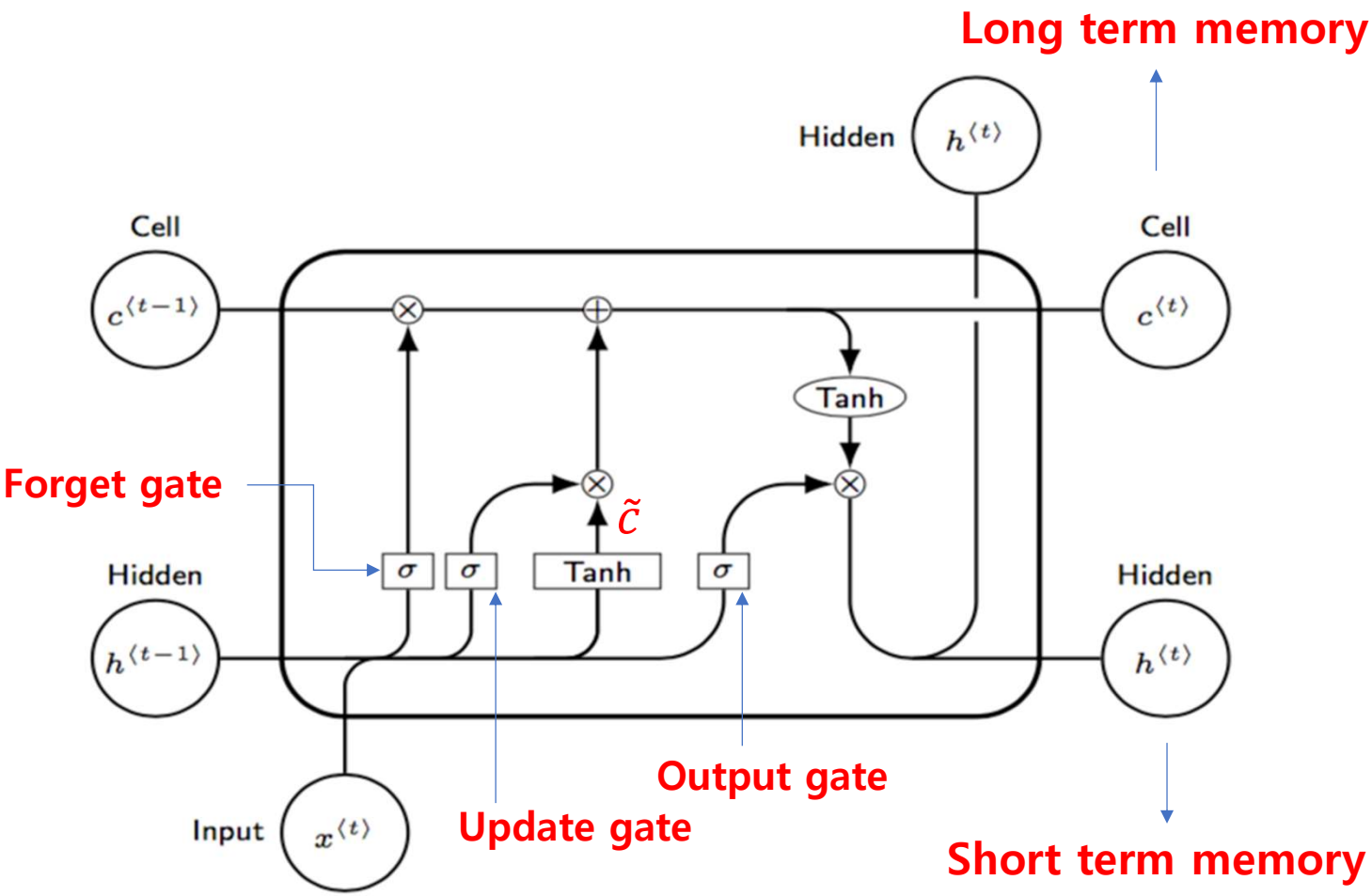
He was born in **England** and moved to Asia and his mother language is

Long-Term Memory 필요

SimpleRNN



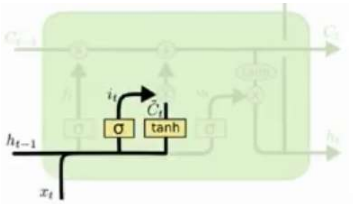
LSTM (Long Short-Term Memory)



LSTM 내부 구조

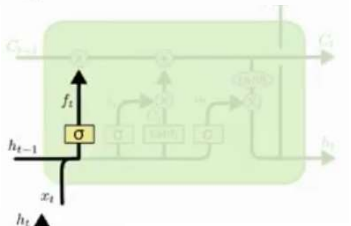
- Input – 이전 step 의 output + new data

  $\tilde{C}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$ ➔ 새로운 cell status 후보



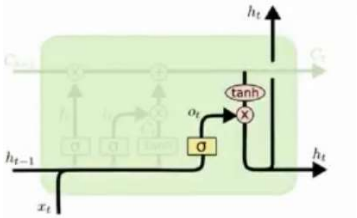
- Update gate – 새로운 input 을 어느정도 받아들일지 결정 (0-무시, 1-전체)

  $\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$



- Forget gate – 내부 state 를 어느정도 기억할지 결정 (0-forget, 1-전체 기억)

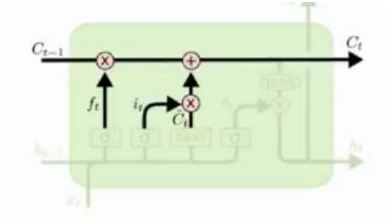
  $\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$



- Output gate – cell state 의 어느 부분을 output 으로 보낼지 결정

  $\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$

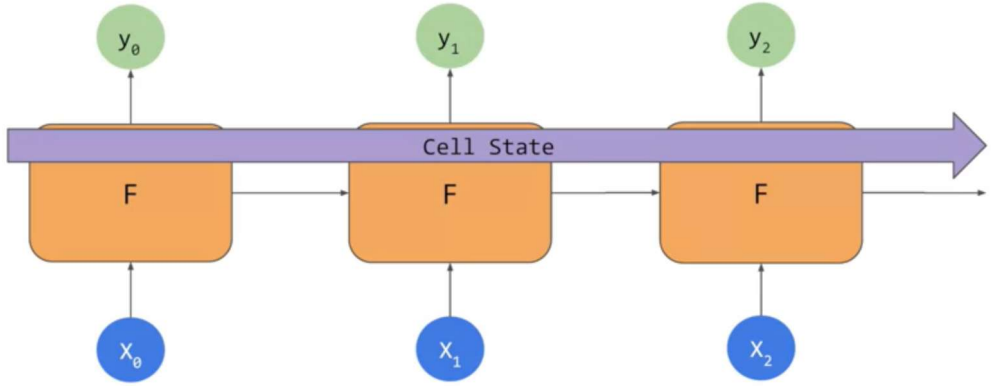
- $C^{<t>} = \Gamma_u * \tilde{C}^{<t>} + \Gamma_f * C^{<t-1>}$



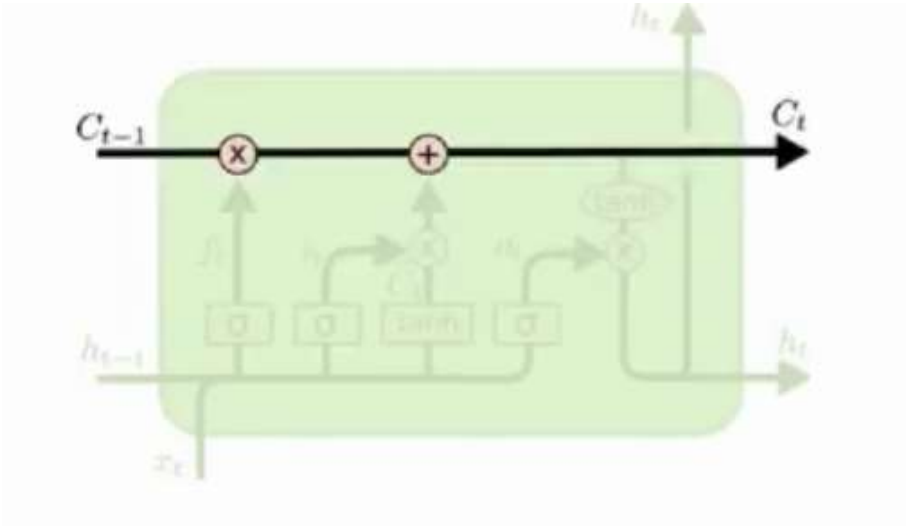
Past cell status * forget gate + Current cell status candidate * update gate

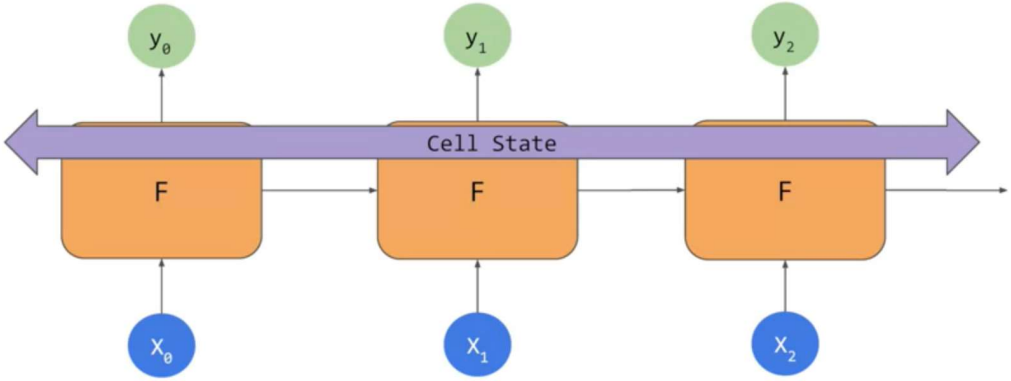
- $a^{<t>} = \Gamma_o * \tanh C^{<t>}$

LSTM Cell State



Uni-directional

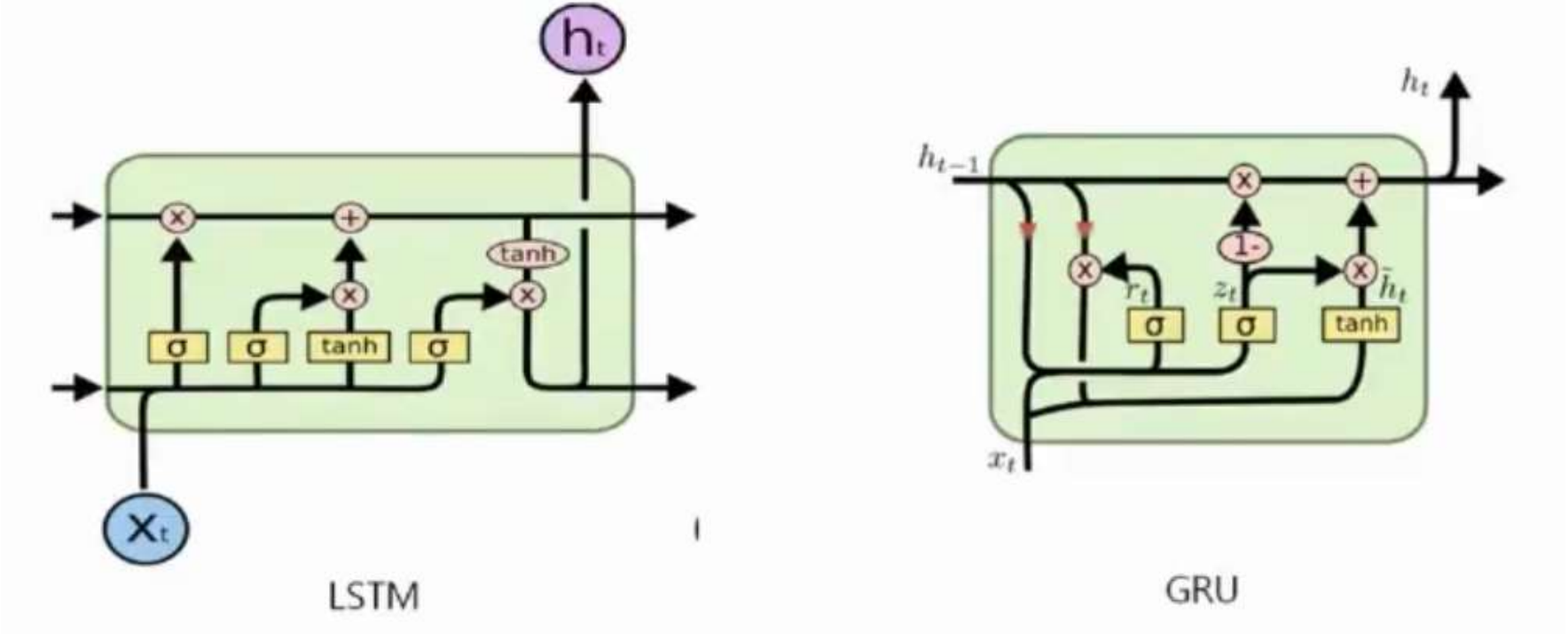




Bi-directional

GRU (Gated Recurrent Unit)

- LSTM 의 장점을 유지하면서 일부 Gate 를 생략하여 계산의 복잡성을 낮춤



자연어 처리(NLP)의 RNN 적용

단어(word) 와 문장(sentence)의 표현 방법

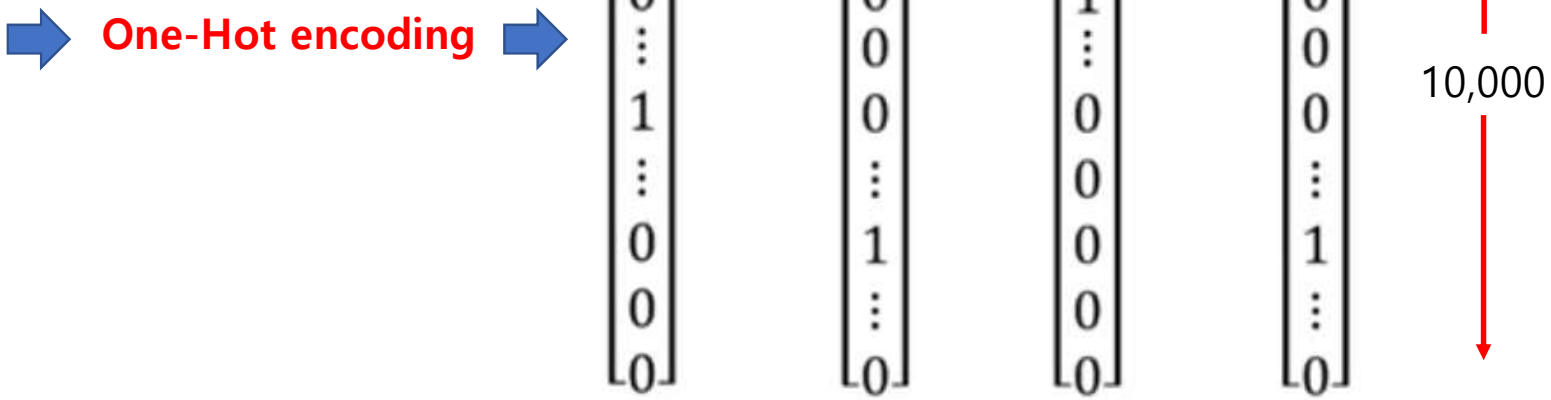
- Text 전 처리 – 특수문자 제거, 불용어 제거 등
- Token 분리 – 단어(형태소) 단위 분리
- One-hot-encoding
- Word Embedding

단어의 Vector 표현

- One-Hot-Encoding

Vocaburary 사전

a -1
aaron – 2
.
apple – 456
.
king – 4914
.
orange – 6257
.
queen – 7157
.
zebra – 9,999
<UNK> - 10,000



Word Embedding

- 단순 One-Hot-Encoding 의 문제점 – 단어 간의 유사도가 표시되지 않음
- 수자화된 단어의 나열 → sentiment 추출
- 연관성 있는 단어들을 군집화하여 multi-dimension 공간에 vector 로 표시
- 예를 들어, 호감(positive), 비호감(negative) 두가지 label 에 따라 관련 단어들을 두개의 category 로 군집화

  ex) IMDB : boring, bad, unfunny → negative
  funny, good, interesting → positive

Word Embedding (Feature 화 표시)

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
⋮	⋮	⋮				
size						
cost						
alive						
verb						

300

I like a glass of orange _____

I like a glass of apple _____

Man (5931) 의 300 dimension vector 표시

Embedding matrix (example)

학습된 featrues

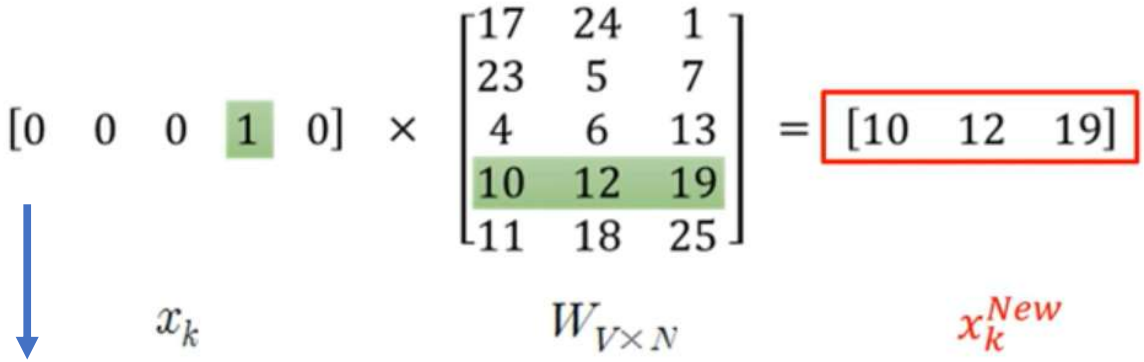
words

	0	1	2	3	4	5	6	7	8	9	...	290	291	292	
fox	-0.348680	-0.077720	0.177750	-0.094953	-0.452890	0.237790	0.209440	0.037886	0.035064	0.899010	...	-0.283050	0.270240	-0.654800	0.10
ham	-0.773320	-0.282540	0.580760	0.841480	0.258540	0.585210	-0.021890	-0.463680	0.139070	0.658720	...	0.464470	0.481400	-0.829200	0.35
brown	-0.374120	-0.076264	0.109260	0.186620	0.029943	0.182700	-0.631980	0.133060	-0.128980	0.603430	...	-0.015404	0.392890	-0.034826	-0.72
beautiful	0.171200	0.534390	-0.348540	-0.097234	0.101800	-0.170860	0.295650	-0.041816	-0.516550	2.117200	...	-0.285540	0.104670	0.126310	0.12
jumps	-0.334840	0.215990	-0.350440	-0.260020	0.411070	0.154010	-0.386110	0.206380	0.386700	1.460500	...	-0.107030	-0.279480	-0.186200	-0.54
eggs	-0.417810	-0.035192	-0.126150	-0.215930	-0.669740	0.513250	-0.797090	-0.068611	0.634660	1.256300	...	-0.232860	-0.139740	-0.681080	-0.37
beans	-0.423290	-0.264500	0.200870	0.082187	0.066944	1.027600	-0.989140	-0.259950	0.145960	0.766450	...	0.048760	0.351680	-0.786260	-0.36
sky	0.312550	-0.303080	0.019587	-0.354940	0.100180	-0.141530	-0.514270	0.886110	-0.530540	1.556600	...	-0.667050	0.279110	0.500970	-0.27
bacon	-0.430730	-0.016025	0.484620	0.101390	-0.299200	0.761820	-0.353130	-0.325290	0.156730	0.873210	...	0.304240	0.413440	-0.540730	-0.03
breakfast	0.073378	0.227670	0.208420	-0.456790	-0.078219	0.601960	-0.024494	-0.467980	0.054627	2.283700	...	0.647710	0.373820	0.019931	-0.03
toast	0.130740	-0.193730	0.253270	0.090102	-0.272580	-0.030571	0.096945	-0.115060	0.484000	0.848380	...	0.142080	0.481910	0.045167	0.05
today	-0.156570	0.594890	-0.031445	-0.077586	0.278630	-0.509210	-0.066350	-0.081890	-0.047986	2.803600	...	-0.326580	-0.413380	0.367910	-0.26
blue	0.129450	0.036518	0.032298	-0.060034	0.399840	-0.103020	-0.507880	0.076630	-0.422920	0.815730	...	-0.501280	0.169010	0.548250	-0.31
green	-0.072368	0.233200	0.137260	-0.156630	0.248440	0.349870	-0.241700	-0.091426	-0.530150	1.341300	...	-0.405170	0.243570	0.437300	-0.46
kings	0.259230	-0.854690	0.360010	-0.642000	0.568530	-0.321420	0.173250	0.133030	-0.089720	1.528600	...	-0.470090	0.063743	-0.545210	-0.19
dog	-0.057120	0.052685	0.003026	-0.048517	0.007043	0.041856	-0.024704	-0.039783	0.009614	0.308416	...	0.003257	-0.036864	-0.043878	0.00
sausages	-0.174290	-0.064869	-0.046976	0.287420	-0.128150	0.647630	0.056315	-0.240440	-0.025094	0.502220	...	0.302240	0.195470	-0.653980	-0.29
lazy	-0.353320	-0.299710	-0.176230	-0.321940	-0.385640	0.586110	0.411160	-0.418680	0.073093	1.486500	...	0.402310	-0.038554	-0.288670	-0.24
love	0.139490	0.534530	-0.252470	-0.125650	0.048748	0.152440	0.199060	-0.065970	0.128830	2.055900	...	-0.124380	0.178440	-0.099469	0.00
quick	-0.445630	0.191510	-0.249210	0.465900	0.161950	0.212780	-0.046480	0.021170	0.417660	1.686900	...	-0.329460	0.421860	-0.039543	0.15

20 rows × 300 columns

Embedding Layer 를 이용한 vector 표현

- Projection Matrix



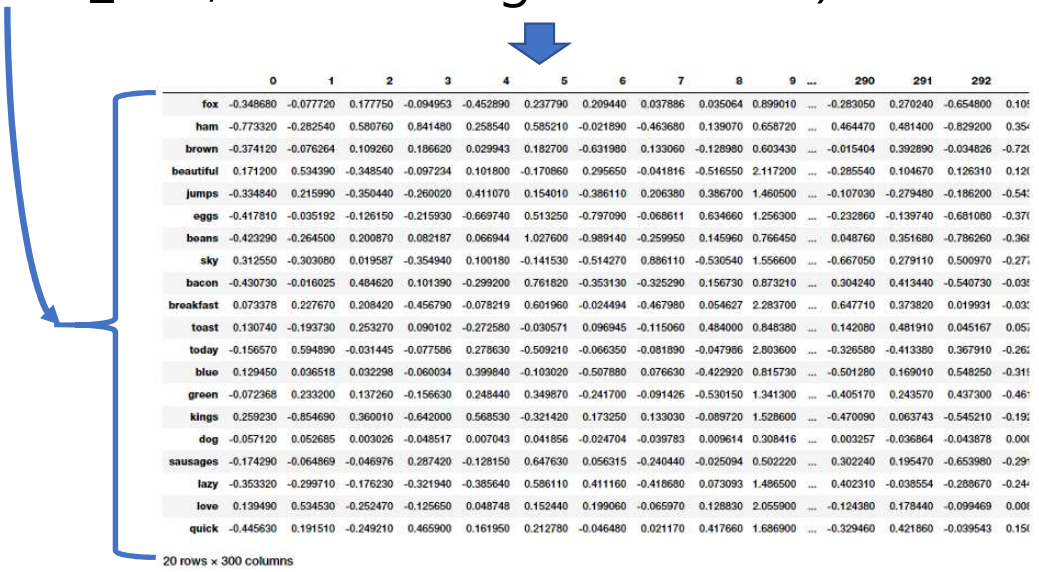
- **One-Hot-Vector 에 Projection Matrix(Embedding Layer) 를 곱해 새로운 vector 생성 ➔** 계산의 간편성

- Projection Matrix 의 k 번째 row 가 k 번째 단어에 대응하는 **weight** 임

Neural Network 의 word embedding Layer

- 입력 data 에 대해 numpy matrix 연산을 manually 하는 것은 비효율적이므로, tf.keras.layers.Embedding() 이용

- Tensorflow NLP model 의 1st layer 는 Embedding layer 로 시작
  - tf.keras.layers.Embedding(vocab_size, embedding dimension)
  - One-hot-encoding
  - indexing

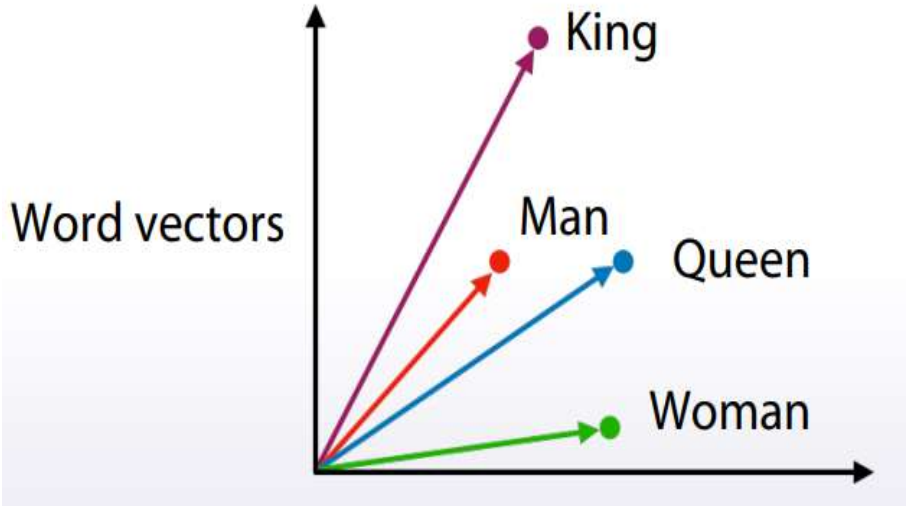


	0	1	2	3	4	5	6	7	8	9	...	290	291	292	
fox	-0.348680	-0.077720	0.177750	-0.094953	-0.452890	0.237790	0.209440	0.037886	0.035064	0.899010	...	-0.283050	0.270240	-0.654800	0.10
ham	-0.773320	-0.282540	0.580760	0.841480	0.258540	0.585210	-0.021890	-0.463680	0.139070	0.658720	...	0.464470	0.481400	-0.829200	0.35
brown	-0.374120	-0.076264	0.109260	0.186620	0.029943	0.182700	-0.631980	0.133060	-0.128980	0.603430	...	-0.015404	0.392890	-0.034826	-0.72
beautiful	0.171200	0.534390	-0.348540	-0.097234	0.101800	-0.170860	0.295650	-0.041816	-0.516550	2.117200	...	-0.285540	0.104670	0.126310	0.12
jumps	-0.334840	0.215990	-0.350440	-0.260020	0.411070	0.154010	-0.386110	0.206380	0.386700	1.460500	...	-0.107030	-0.279480	-0.186200	-0.54
eggs	-0.417810	-0.035192	-0.126150	-0.215930	-0.669740	0.513250	-0.797090	-0.068611	0.634660	1.256300	...	-0.232860	-0.139740	-0.681080	-0.37
beans	-0.423290	-0.264500	0.200870	0.082187	0.066944	1.027600	-0.989140	-0.259950	0.145960	0.766450	...	0.048760	0.351680	-0.786260	-0.36
sky	0.312550	-0.303080	0.019587	-0.354940	0.100180	-0.141530	-0.514270	0.886110	-0.530540	1.556600	...	-0.667050	0.279110	0.500970	-0.27
bacon	-0.430730	-0.016025	0.484620	0.101390	-0.299200	0.761820	-0.353130	-0.325290	0.156730	0.873210	...	0.304240	0.413440	-0.540730	-0.03
breakfast	0.073378	0.227670	0.208420	-0.456790	-0.078219	0.601960	-0.024494	-0.467980	0.054627	2.283700	...	0.647710	0.373820	0.019931	-0.03
toast	0.130740	-0.193730	0.253270	0.090102	-0.272580	-0.030571	0.096945	-0.115060	0.484000	0.848380	...	0.142080	0.481910	0.045167	0.05
today	-0.156570	0.594890	-0.031445	-0.077586	0.278630	-0.509210	-0.066350	-0.081890	-0.047986	2.803600	...	-0.326580	-0.413380	0.367910	-0.26
blue	0.129450	0.036518	0.032298	-0.060034	0.399840	-0.103020	-0.507880	0.076630	-0.422920	0.815730	...	-0.501280	0.169010	0.548250	-0.31
green	-0.072368	0.233200	0.137260	-0.156630	0.248440	0.349870	-0.241700	-0.091426	-0.530150	1.341300	...	-0.405170	0.243570	0.437300	-0.46
kings	0.259230	-0.854690	0.360010	-0.642000	0.568530	-0.321420	0.173250	0.133030	-0.089720	1.528600	...	-0.470090	0.063743	-0.545210	-0.19
dog	-0.057120	0.052685	0.003026	-0.048517	0.007043	0.041856	-0.024704	-0.039783	0.009614	0.308416	...	0.003257	-0.036864	-0.043878	0.00
sausages	-0.174290	-0.064869	-0.046976	0.287420	-0.128150	0.647630	0.056315	-0.240440	-0.025094	0.502220	...	0.302240	0.195470	-0.653980	-0.29
lazy	-0.353320	-0.299710	-0.176230	-0.321940	-0.385640	0.586110	0.411160	-0.418680	0.073093	1.486500	...	0.402310	-0.038554	-0.288670	-0.24
love	0.139490	0.534530	-0.252470	-0.125650	0.048748	0.152440	0.199060	-0.065970	0.128830	2.055900	...	-0.124380	0.178440	-0.099469	0.00
quick	-0.445630	0.191510	-0.249210	0.465900	0.161950	0.212780	-0.046480	0.021170	0.417660	1.686900	...	-0.329460	0.421860	-0.039543	0.15

20 rows × 300 columns

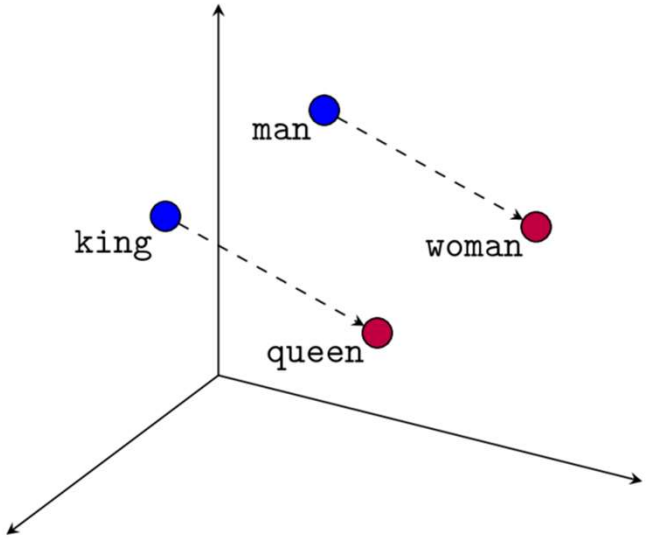
Word2Vec

- 2013년 구글에서 개발
- 단어를 vector 화 (Word Embedding) 하여 단어의 의미 표현
- 매우 큰 Corpus (ex, 10억, 100 억 단어) 에서 word embedding 자동 학습

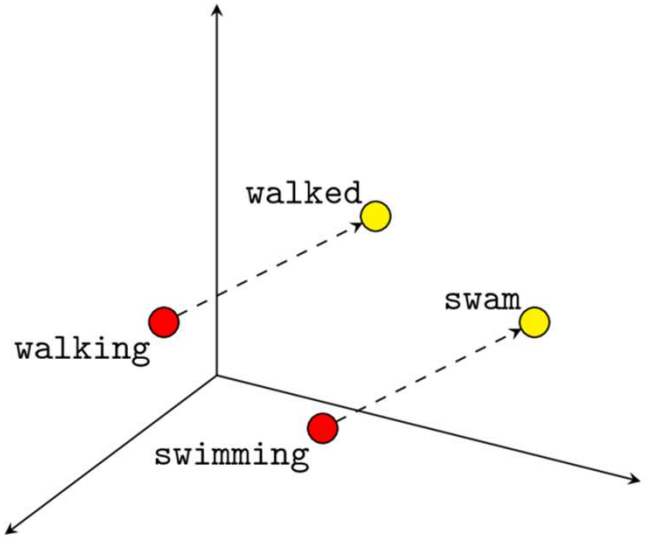


Word2Vec Vectorized (Word Embedding)

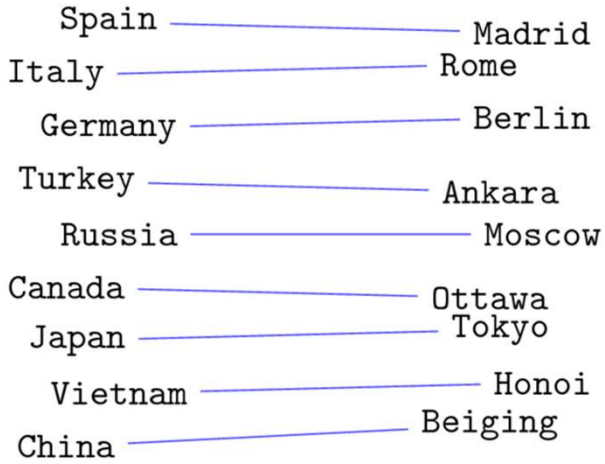
king – queen ≈ man - woman



성별



시제



국가-수도

https://ronxin.github.io/wevi/

Word2Vec

- 중심단어로 주변단어를 (skip-gram), 주변단어로 중심단어를(CBOW) 예측하는 과정에서 단어를 벡터로 임베딩하는 방법

- 임베딩된 단어의 내적(inner product)이 코사인 유사도가 되도록 함

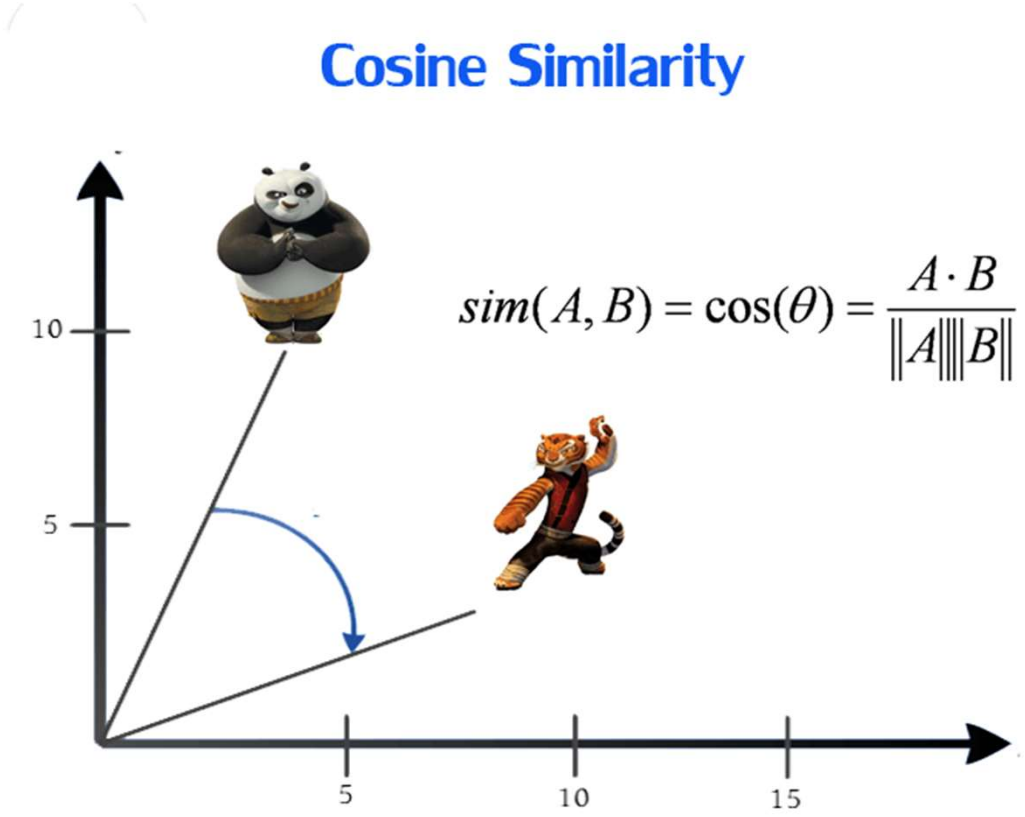
ex) I love **king** of Korea. (skip-gram : window size = 2)

input : king → [0,0,0,0,….1,..0]

target : I love → [0,0,0,0,….1,..0] [0,0,0,0,0,0,…1,….0]

of Korea → [0,0,0,….1,..0] [0,0,0,0,0,…1,….0]

유사도 측정 (Cosine Similarity)

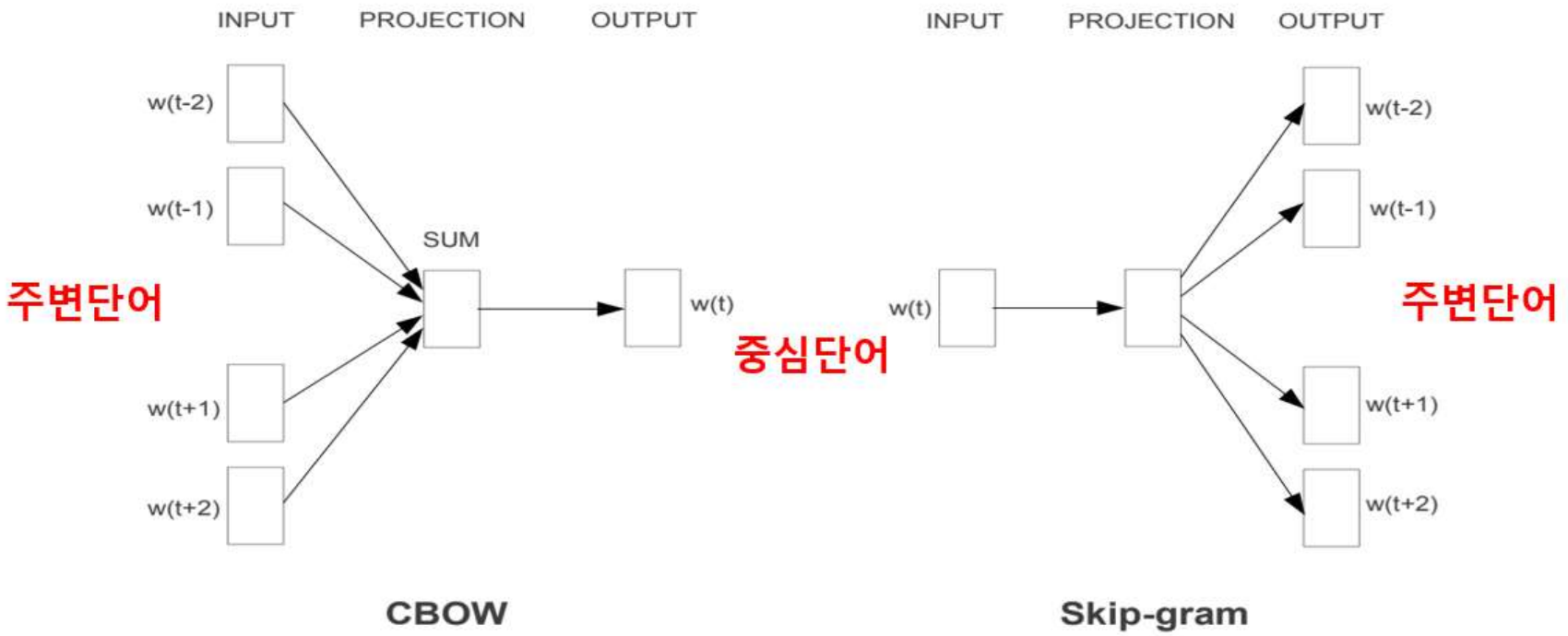


$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

A, B 유사하면 ≈ 1

관련 없으면 ≈ 0

CBOW (Continuous Bag of Words) vs. Skip-gram



- 주변단어(문맥)로 부터 중심단어를 예측
- 적은 데이터에서 성능 좋음

- 중심단어로 부터 주변단어(문맥) 예측
- 많은 데이터에서 성능 좋음
- 주로 사용되는 방법

Word encoding – keras API

```
1  import tensorflow as tf
2  from tensorflow import keras
3  from tensorflow.keras.preprocessing.text import Tokenizer
```

```
1  sentences = ['I love my dog.', 'I love my cat.', 'You love my dog!']
2
3  tokenizer = Tokenizer(num_words=100)    # take top 100 words from the sentences
4  tokenizer.fit_on_texts(sentences)
5  word_index = tokenizer.word_index
```

```
1  print(word_index)
```

```
{'love': 1, 'my': 2, 'i': 3, 'dog': 4, 'cat': 5, 'you': 6}
```

소문자 변환, 구둣점 제거

Text to sequence – 문장의 수열 표시

```
1  sentences = ['I love my dog.',
2               'I love my cat.',
3               'You love my dog!',
4               'Do you think my dog is amazing?']
5
6  tokenizer = Tokenizer(num_words=100)    # take top 100 words from the sentences
7  tokenizer.fit_on_texts(sentences)
8  word_index = tokenizer.word_index
9
10 sequences = tokenizer.texts_to_sequences(sentences)
```

Corpus

```
1  print(word_index)
2  print(sequences)
```

```
{'my': 1, 'love': 2, 'dog': 3, 'i': 4, 'you': 5, 'cat': 6, 'do': 7, 'think': 8, 'is': 9, 'amazing': 10}
[[4, 2, 1, 3], [4, 2, 1, 6], [5, 2, 1, 3], [7, 5, 8, 1, 3, 9, 10]]
```

word index 에 없는 단어를 가진 문장 처리

```
1  test_data = ['I really love my dog',
2               'my dog loves my lizard']
3
4  test_seq = tokenizer.texts_to_sequences(test_data)
5
6  print(test_seq)
7  print(word_index)
```

```
[[4, 2, 1, 3], [1, 3, 1]]
{'my': 1, 'love': 2, 'dog': 3, 'i': 4, 'you': 5, 'cat': 6, 'do': 7, 'think': 8, 'is': 9, 'amazing': 10}
```

my dog my

- Word index 작성 시 Large corpus 필요
- 없는 단어의 자리를 special token (oov_token) 으로 채움

```
sentences = ['I love my dog.',
             'I love my cat.',
             'You love my dog!',
             'Do you think my dog is amazing?']

tokenizer = Tokenizer(num_words=100,oov_token='<OOV>')    # take top 100 words from the sentences
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)
```

```
print(word_index)
print(sequences)
```

```
{'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}
[[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]
```

```
test_data = ['I really love my dog',
             'my dog loves my lizard']

test_seq = tokenizer.texts_to_sequences(test_data)

print(test_seq)
print(word_index)
```

```
[[5, 1, 3, 2, 4], [2, 4, 1, 2, 1]]
{'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}
```

padding

- 입력 text 를 동일한 길이로 맞추기

```
4 from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
1  sentences = ['I love my dog.',
2               'I love my cat.',
3               'You love my dog!',
4               'Do you think my dog is amazing?']
5
6  tokenizer = Tokenizer(num_words=100, oov_token='<OOV>')    # take top 100 words from the sentences
7  tokenizer.fit_on_texts(sentences)
8  word_index = tokenizer.word_index
9
10 sequences = tokenizer.texts_to_sequences(sentences)
11 padded = pad_sequences(sequences)
```

```
1 print(word_index)
2 print(sequences)
3 print(padded)
```

```
{'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}
[[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]
[[ 0  0  0  5  3  2  4]
 [ 0  0  0  5  3  2  7]
 [ 0  0  0  6  3  2  4]
 [ 8  6  9  2  4 10 11]]
```

Padding 위치 및 max length

```
10 sequences = tokenizer.texts_to_sequences(sentences)
11 padded = pad_sequences(sequences, padding='post', truncating='post', maxlen=5)
```

```
1 print(word_index)
2 print(sequences)
3 print(padded)
```

```
{'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}
[[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]
[[5 3 2 4 0]
 [5 3 2 7 0]
 [6 3 2 4 0]
 [8 6 9 2 4]]
```

```
tokenizer = Tokenizer(num_words=100,oov_token='<OOV>')  # 빈도수 상위 100 개로 구성

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'I was born in Korea and graduaged University in USA.'
]

tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

```
{'<OOV>': 1, 'i': 2, 'love': 3, 'my': 4, 'dog': 5, 'in': 6, 'cat': 7, 'you': 8, 'was': 9, 'born': 10, 'korea': 11, 'and': 12, 'graduaged': 13, 'unive
rsity': 14, 'usa': 15}
```

Tokenizer (word → number)

```
sequences = tokenizer.texts_to_sequences(sentences)
sequences
```

```
[[2, 3, 4, 5],
 [2, 3, 4, 7],
 [8, 3, 4, 5],
 [2, 9, 10, 6, 11, 12, 13, 14, 6, 15]]
```

```
padded = pad_sequences(sequences)
padded
```

```
array([[ 0,  0,  0,  0,  0,  0,  2,  3,  4,  5],
       [ 0,  0,  0,  0,  0,  0,  2,  3,  4,  7],
       [ 0,  0,  0,  0,  0,  0,  8,  3,  4,  5],
       [ 2,  9, 10,  6, 11, 12, 13, 14,  6, 15]])
```

Padding – make uniform input length

pad_sequences(sequences, padding="pre")

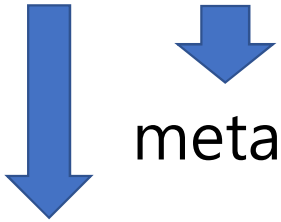
- padding='pre' or 'post'

실습 : 영화 관람평 분류 (sentiment 분석)

- Keras 의 built-in IMDB (Internet Movie Database) 이용
  - 각 25,000 개 training/testing 영화관람평
- Movie reviews sentiment classification
  - Label : positive, negative
- Preprocessing 되어 있고 모든 review 는 word indexes (integers) 로 표시 (인덱스 순서는 빈번히 나타나는 단어 순서. ex. 3 : 3 번째로 빈번히 나타나는 단어)
- LSTM 을 이용한 Many-to-One type 의 RNN 으로 구현

IMDB Dataset

```
import tensorflow_datasets as tfds
imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)
```



metadata

(input, label)

Train, Test dataset 분리

```
train_data, test_data = imdb['train'], imdb['test']
```

```
training_sentences = []
training_labels = []

testing_sentences = []
testing_labels = []

for s,l in train_data:
    training_sentences.append(str(s.numpy()))
    training_labels.append(l.numpy())

for s,l in test_data:
    testing_sentences.append(str(s.numpy()))
    testing_labels.append(l.numpy())

training_labels_final = np.array(training_labels)
testing_labels_final = np.array(testing_labels)
```

Text, Label 분리

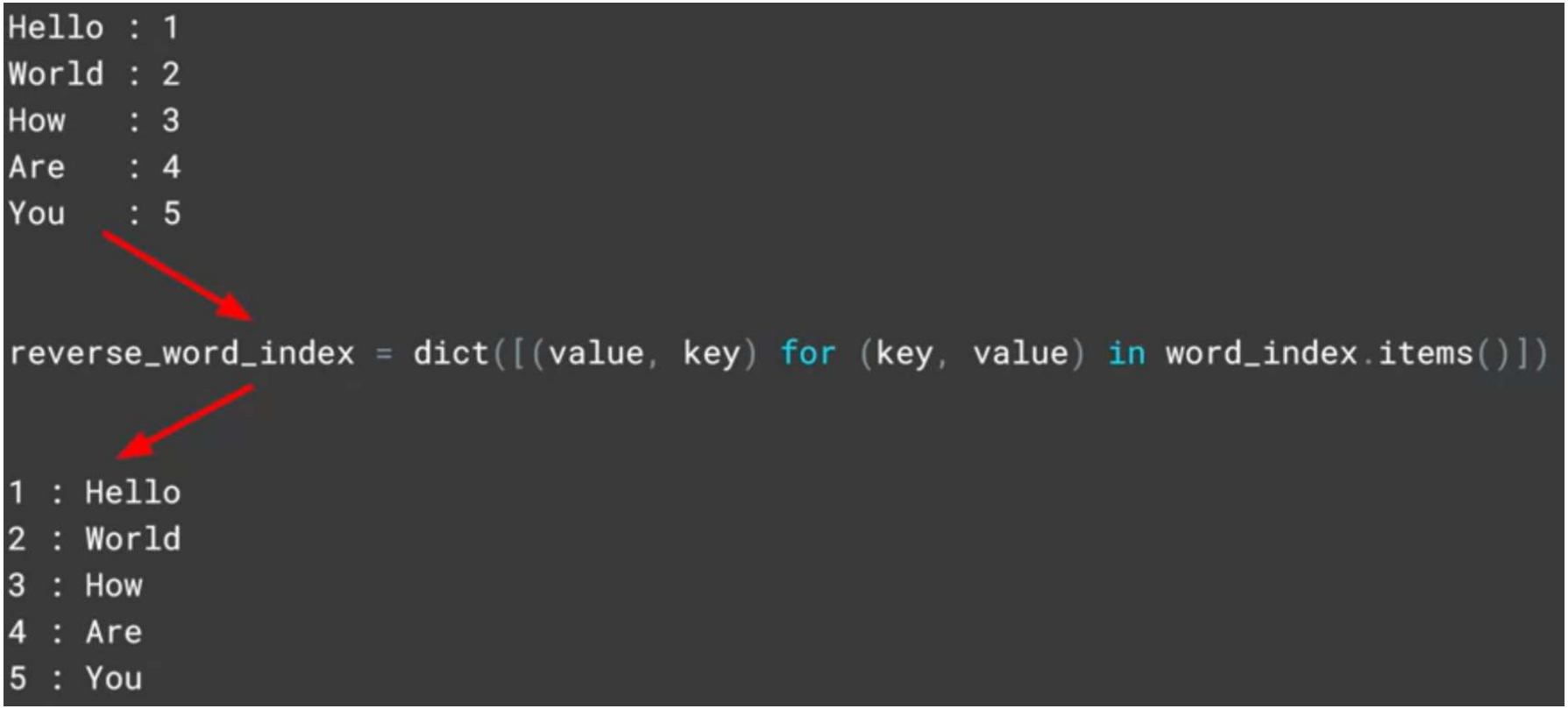
s, l : tensor

Tokenize sentences of IMDB

```
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length, truncating=trunc_type)
```

Hyper-parameters

Helper function of reverse word_index



실습: Sentence 생성 (이상한 나라의 Alice)

```
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1

# create input sequences using list of tokens
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

# create predictors and label
predictors, label = input_sequences[:,:-1],input_sequences[:,-1]

label = ku.to_categorical(label, num_classes=total_words)     # one-hot-encoding
```

```
Line:                                   Input Sequences:

[4 2 66 8 67 68 69 70]                  [4 2]

                                        [4 2 66]

                                        [4 2 66 8]

                                        [4 2 66 8 67]

                                        [4 2 66 8 67 68]

                                        [4 2 66 8 67 68 69]

                                        [4 2 66 8 67 68 69 70]
```

```
# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

# create predictors and label
predictors, label = input_sequences[:,:-1],input_sequences[:,-1]
```

```
[4 2 66 8 67 68 69 70]
```

```
Padded Input Sequences:
```

Input (X)

Label (Y)

```
[0 0 0 0 0 0 0 0 0 0 4 2]
[0 0 0 0 0 0 0 0 0 4 2 66]
[0 0 0 0 0 0 0 0 4 2 66 8]
[0 0 0 0 0 0 0 4 2 66 8 67]
[0 0 0 0 0 0 4 2 66 8 67 68]
[0 0 0 0 0 4 2 66 8 67 68 69]
[0 0 0 0 4 2 66 8 67 68 69 70]
```

실습 : 한글 어린왕자 문장 생성기

- 이상한 나라의 Alice 문장 생성기를 한글 어린왕자 문장 생성기로 응용

Keras Processing Tips

Early Stopping

- Epoch 이 반복되어도 더 이상 성능 향상이 이루어지지 않을 경우 조기에 training 종료

Callback

- Training 중에 keras 의 behavior 를 변경할 수 있는 customization 기능

Callbacks

```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('loss')<0.4):
      print("\nLoss is low so cancelling training!")
      self.model.stop_training = True
```

```
callbacks = myCallback()
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()
training_images=training_images/255.0
test_images=test_images/255.0
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
model.fit(training_images, training_labels, epochs=5, callbacks=[callbacks])
```

실습 : 자동차 연비 예측 Regression

- UCI Machine Learning Data 이용
- 의도적으로 과적합을 만들고, early stopping 기능 test
- Early stopping 전, 후의 loss plot 하여 비교
- Checkpoint 추가 및 model save / reload

GAN

GAN (Generative Adversarial Network)

- “the most interesting idea in the last 10 years in ML” – Yann LeCun
- 2014 Ian Goodfellow 가 최초 제안
- WaveNet – 2016 Google DeepMind 가 제안한 GAN model
- Computer 가 이미지, 인간의 목소리, 악기소리, 소설, 기사 등을 실제와 같이 생성
- 위조를 담당하는 Generator(위조범) 와 위조를 판별하는 Discriminator (경관)의 두개 Deep Neural Network 으로 구성

Generative vs. Discriminative Algorithms

- Discriminative Algorithm

  - input data 의 feature 를 기준으로 label 예측 (ex. Spam 분류)

  - p(y | X) → “the probability of y given X

- Generative Algorithm

  - 주어진 label 을 기준으로 feature 예측

    (feature extraction (x) → feature filling(o))

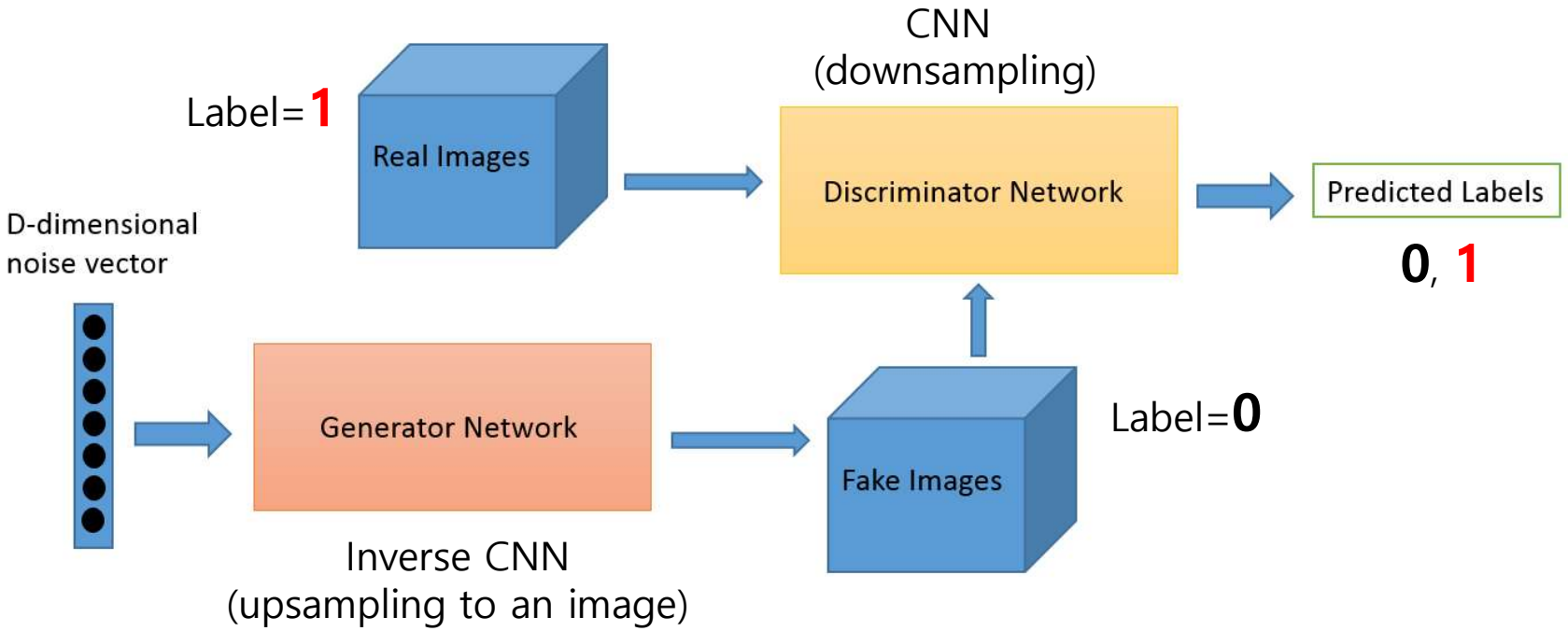
  - p(X | y) → “the probability of features given y”

GAN process

1. Generator 가 random number 를 취하여 random image 생성

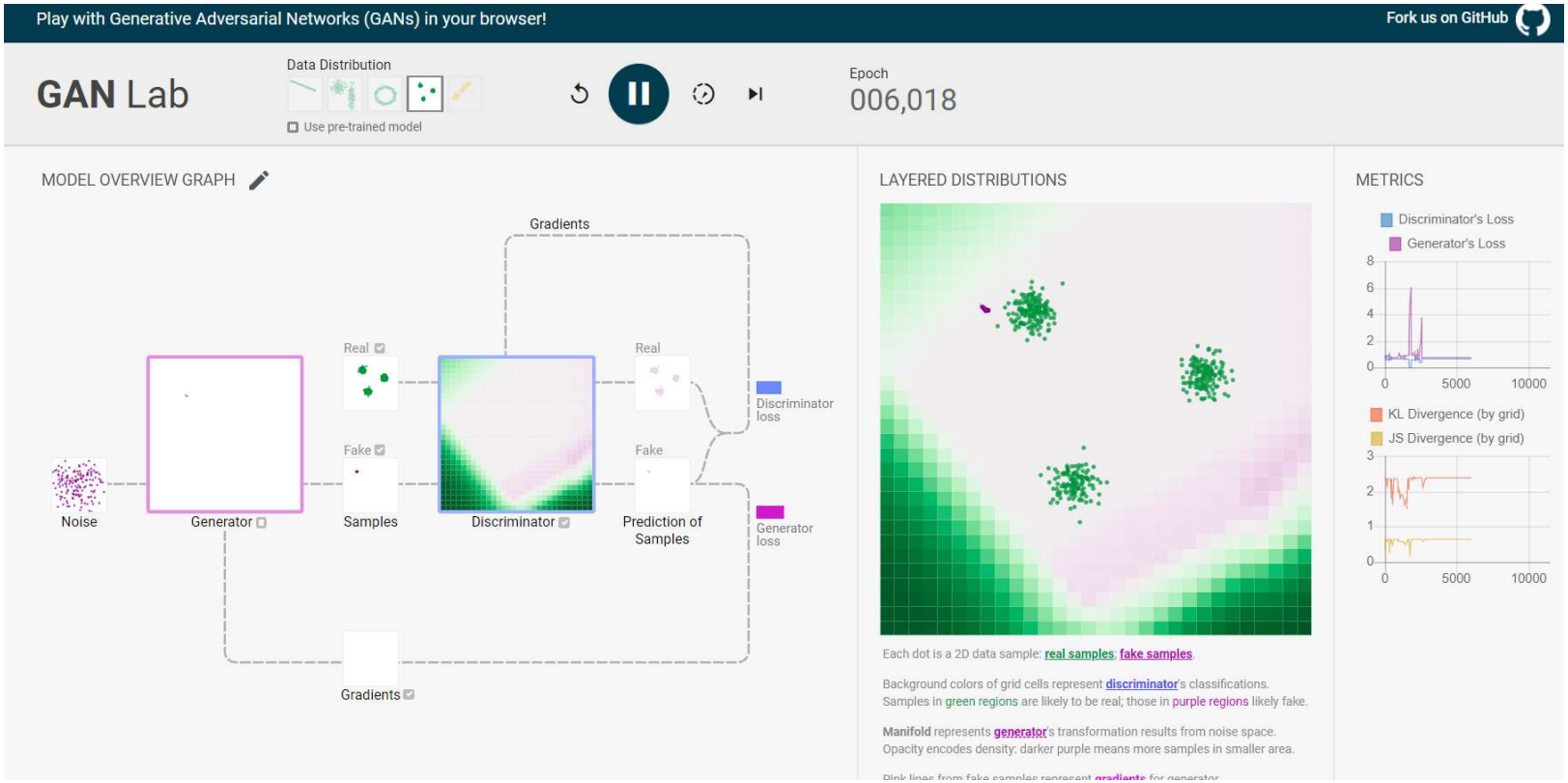
2. 생성된 image 를 actual dataset 에서 받은 image 와 함께 discriminator 에게 공급

3. Discriminator 는 real 과 fake image 를 가지고 0~1 사이의 확률을 반환
   (1 – real, 0 – fake)



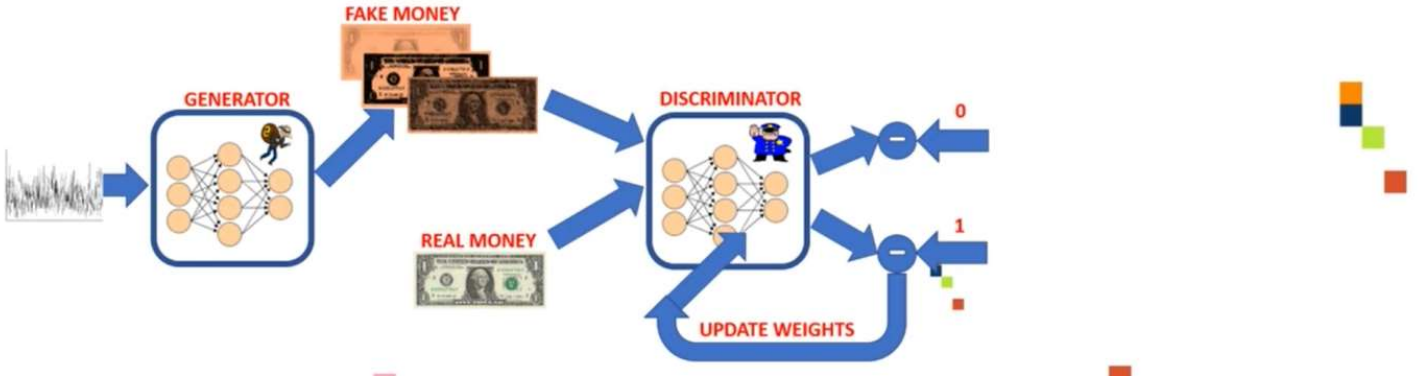
GAN Lab

- https://poloclub.github.io/ganlab/

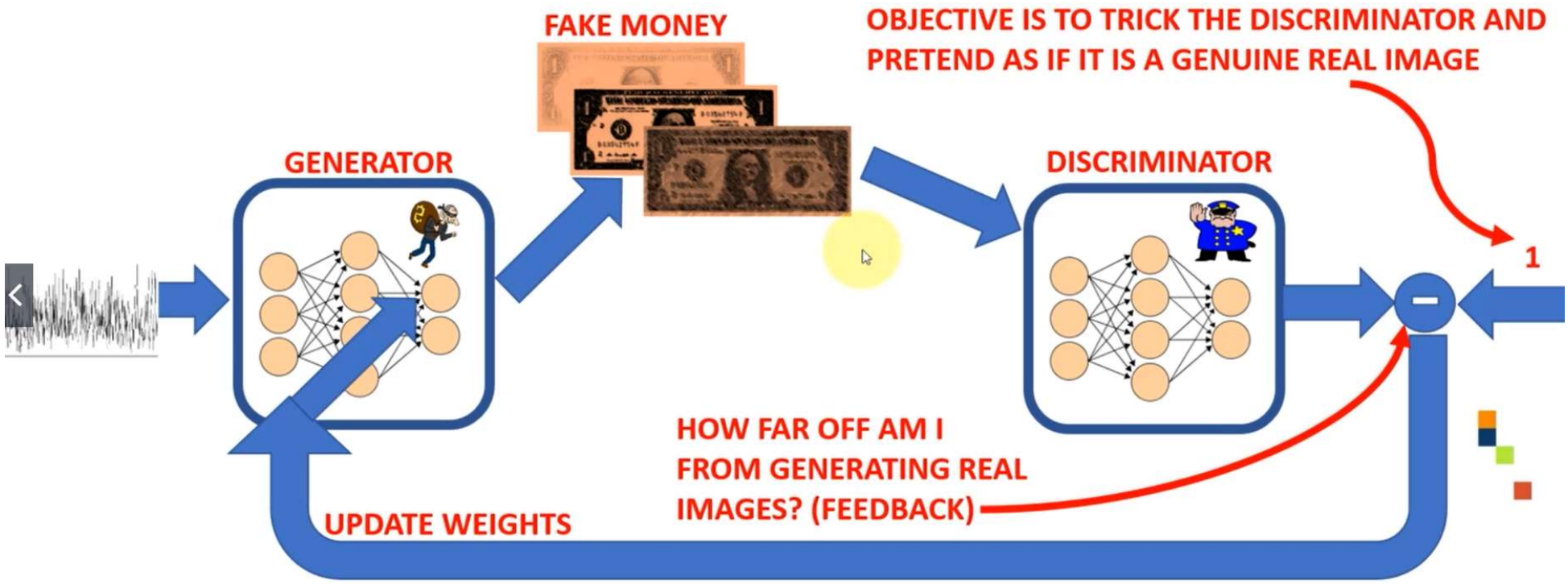


Training of Discriminator

- The discriminator training is performed as follows:
  - The generator will take in random noise and generate fake images
  - Both fake images and real ones are fed to the discriminator network (***Feedforward path***)
  - The discriminator will generate predictions based on the input images (both real and fake)
  - Discriminator predictions are then compared to the true labels to calculate the error
  - The problem is a basic binary classification ANN training, the discriminator is trained to predict 1 for real images and 0 for fake ones
  - The error is propagated through the network to update the discriminator weights **(Backpropagation)**
  - The generator weights are not updated at this stage

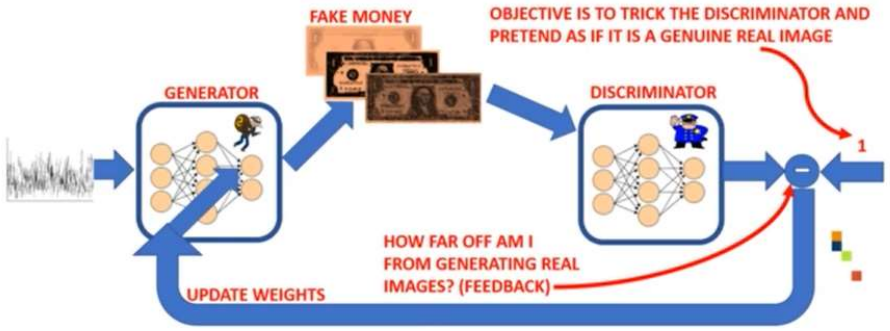


Training of Generator



Training Generator

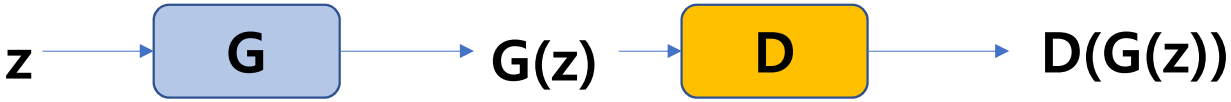
- The generator training is performed as follows:
  - The generator will generate fake images (as usual!)
  - The fake images are fed to the discriminator and it will generate predictions (probably classified as fake~0)
  - The discriminator output will be compared to ***ones (1)*** because it is trying to fool the discriminator to think that this is a real image!
  - Another way of visualizing this, is that the discriminator predictions are compared to one so that the error will represent the feedback that answers the following question:
    - ***How far off am I from generating real images?***
    - ***What should I do better to generate more realistic images?***
  - The generator weights are updated while the discriminator weights are frozen.



GAN implementation

x is MNIST (784 dimension)

Output probability (1 dimension)




```
discriminator = Sequential()
discriminator.add(Dense(1024, input_dim=784))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))

discriminator.add(Dense(512))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))

discriminator.add(Dense(256))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))

discriminator.add(Dense(1, activation='sigmoid'))
```

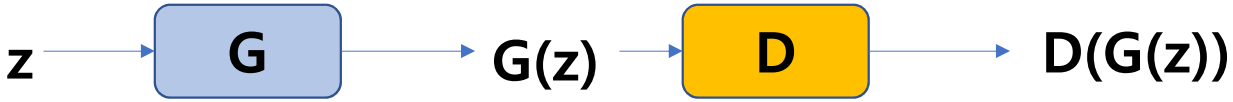
Discriminator

Input size : 784
Output size: 1

GAN implementation

x is MNIST (784 dimension)

Output probability (1 dimension)




```
generator = Sequential()
generator.add(Dense(256, input_dim=random_dim))
generator.add(LeakyReLU(0.2))

generator.add(Dense(512))
generator.add(LeakyReLU(0.2))

generator.add(Dense(1024))
generator.add(LeakyReLU(0.2))

generator.add(Dense(784, activation='tanh'))
```

Generator

Input size : 100
Output size: 784

GAN object function

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

fake

Expectation

x 는 real data 로 부터 표본추출

D(real) 확률

z 는 N(0,1)로 부터 표본추출

D(fake) 확률

- D(x) classifier (Discriminator) 는 real 이면 1, fake 면 0 을 return 하도록 훈련.
  따라서, D(G(z)) 은 G (Generator) 가 만들어낸 G(z) 가 real 이라고 판단되면 1, fake 라고 판단되면 0 return
- $\max_D$ V(D, G) 가 되려면 D(x) 와 1-D(G(z)) 이 모두 1 이 되어야 한다.
- $\min_G$ V(D, G) 가 되려면 1-D(G(z)) 이 0 가 되어야 한다. (D(x) 는 G 와 무관하므로 무시)

Loss function (손실함수) 와 Optimizer 정의

- Discriminator

```
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

```
real_loss = cross_entropy(tf.ones_like(real_output), real_output)
fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
total_loss = real_loss + fake_loss
```

- Generator

```
cross_entropy(tf.ones_like(fake_output), fake_output)
```

- Optimizer

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

GAN special layers

- Generator 는 tf.keras.layers.Conv2DTranspose 를 사용하여 upsampling

- Discriminator 는 Conv2D 를 이용한 전형적인 CNN 이고 output layer 에 sigmoid activation 사용

실습 : CNN 을 이용한 mnist dataset 위조

- Discriminator 의 Goal
  - mnist dataset 을 "진짜" 로 인식하고, Generator 에서 공급되는 image 를 fake 로 구분

- Generator 의 Goal
  – discriminator 가 "진짜" 로 인식할 fake image 생성
    (Gaussian random noise 로 부터 image 생성)

- GAN 의 training 은 시간이 오래 걸린다 → GPU 필요