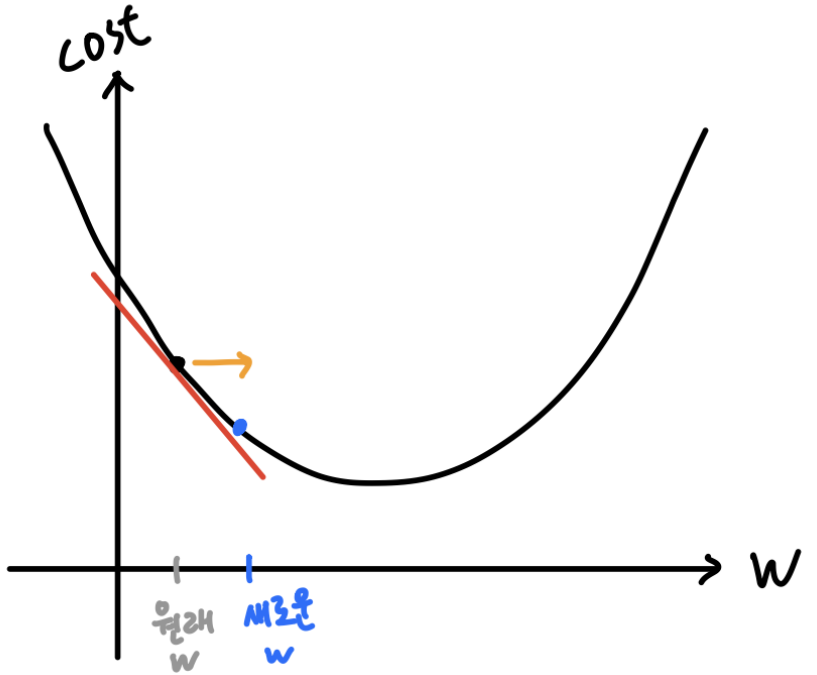
디지털 금융 특수논제- Introduction to Deep Learning

2024-01-03

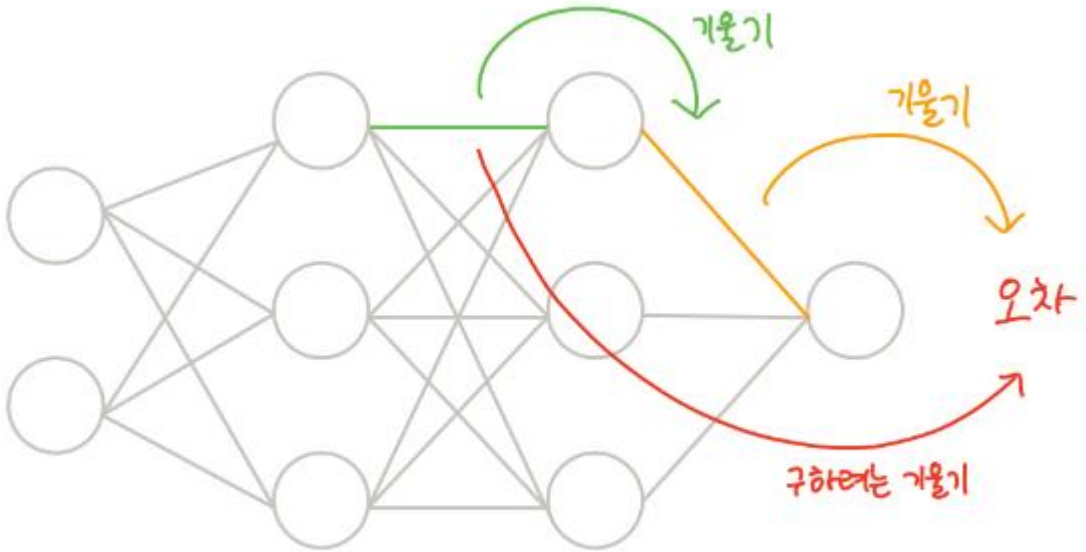
우지환, Ph.D., MBA

Weight는 어떻게 학습될까?



새로운 w = 원래 w − 0.01 × 기울기

learning rate



기울기 X 기울기 = 구하려는 기울기

$$\frac{\partial C}{\partial B} \times \frac{\partial B}{\partial A} = \frac{\partial C}{\partial A}$$

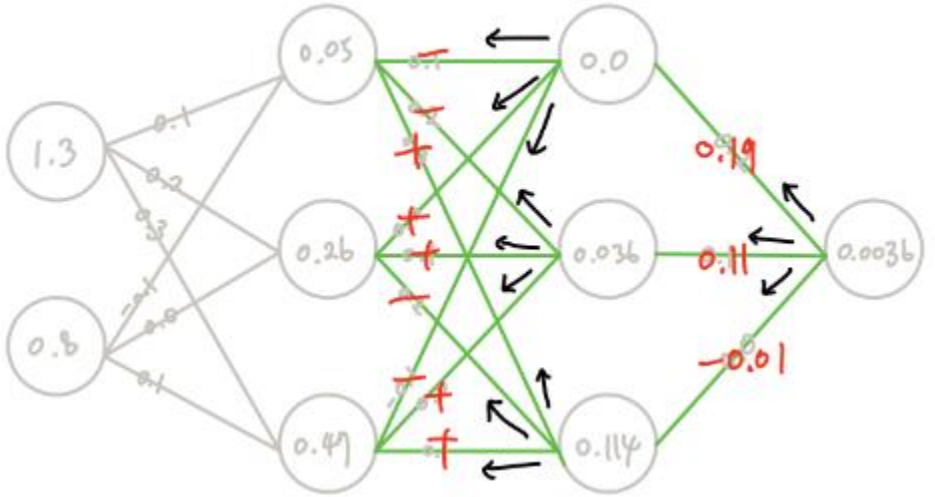
연쇄 법칙

$$W_{11}^{(2)} - \alpha \frac{\partial 오차}{\partial W_{11}^{(2)}}$$

$$\frac{\partial 오차}{\partial W_1^{(3)}} \times \frac{\partial W_1^{(3)}}{\partial W_{11}^{(2)}}$$

Source from https://www.philgineer.com/2021/09/27-5.html

오차 역전파 방법(Backpropagation)-1



$W^{(2)}$

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.0 & 0.1 & 0.2 \\ -0.1 & 0.0 & 0.1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0.09 & 0.17 & 0.32 \\ 0.08 & 0.11 & 0.18 \\ -0.13 & 0.01 & 0.12 \end{bmatrix}$$

Source from https://www.philgineer.com/2021/09/27-5.html

오차 역전파 방법(Backpropagation)-2

$$f(x, y, z) = (x + y)z$$

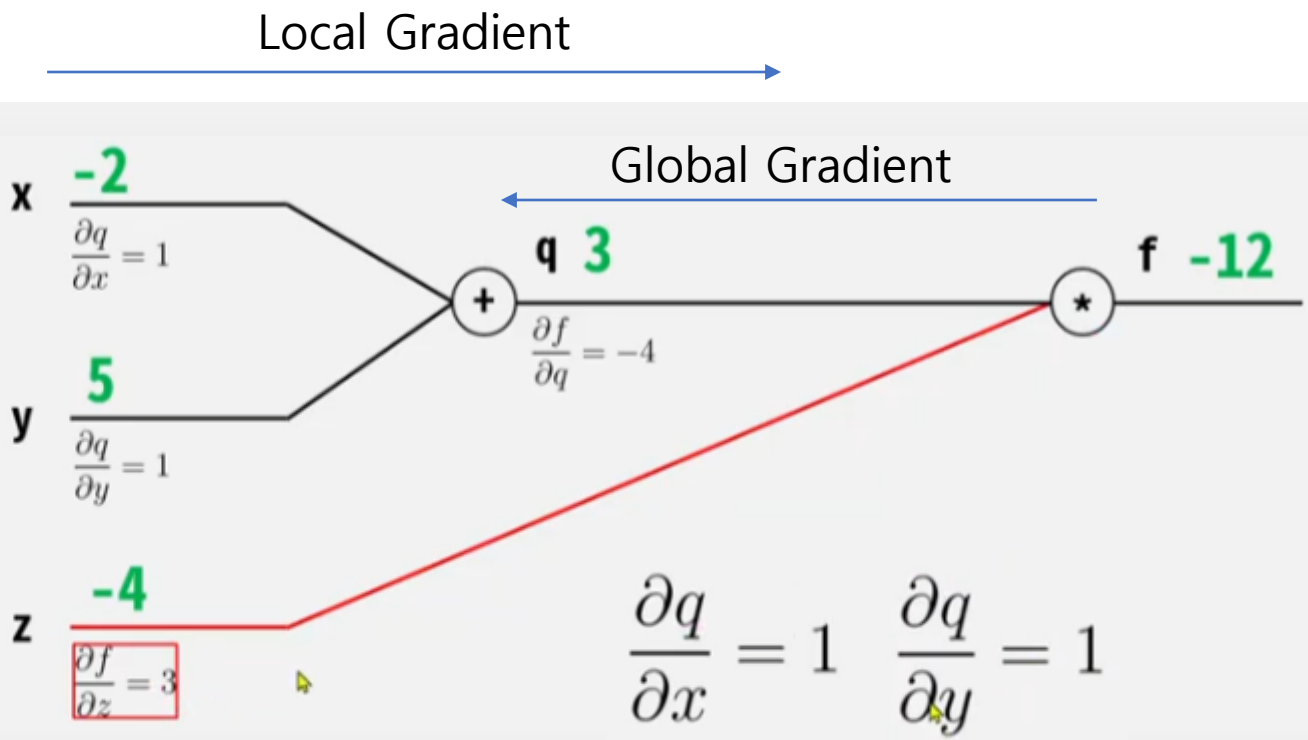
x=-2, y=5, z=-4

$$q = x + y$$

$$f = qz$$

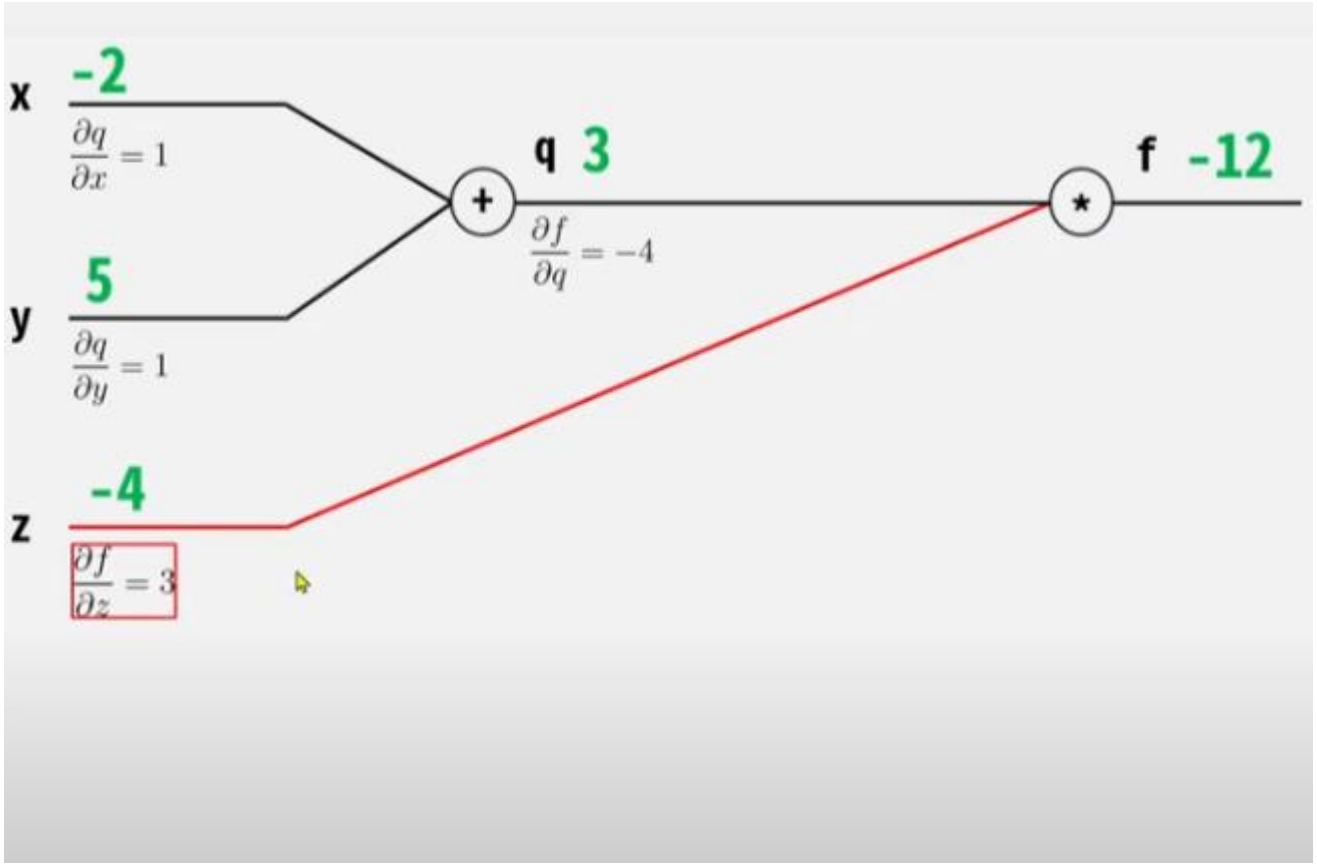
$$\frac{\partial f}{\partial q} = z = -4$$

$$\frac{\partial f}{\partial z} = q = (x + y) = (-2 + 5) = 3$$



오차 역전파 방법(Backpropagation)-3

- Forward Pass시 Local Gradient를 미리 계산하여 저장한다
- 저장한 Local Gradient와 Global Gradient를 곱하여 최종 미분 값을 얻는다.

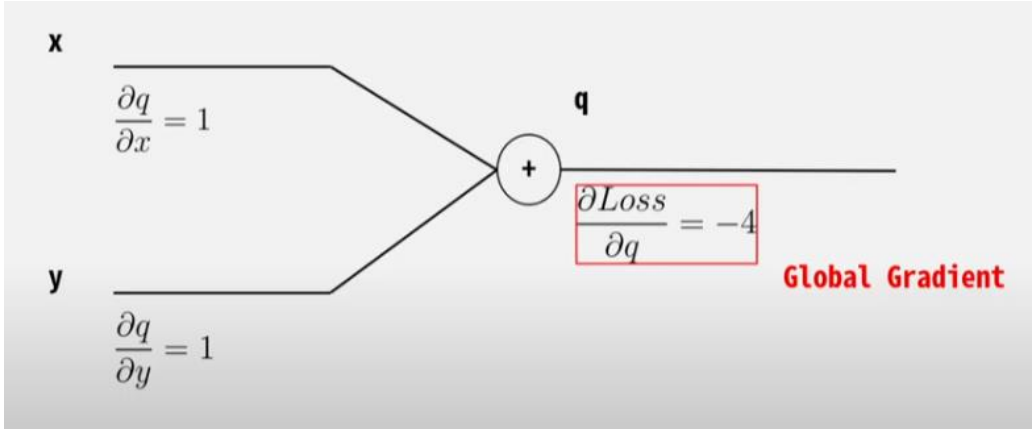


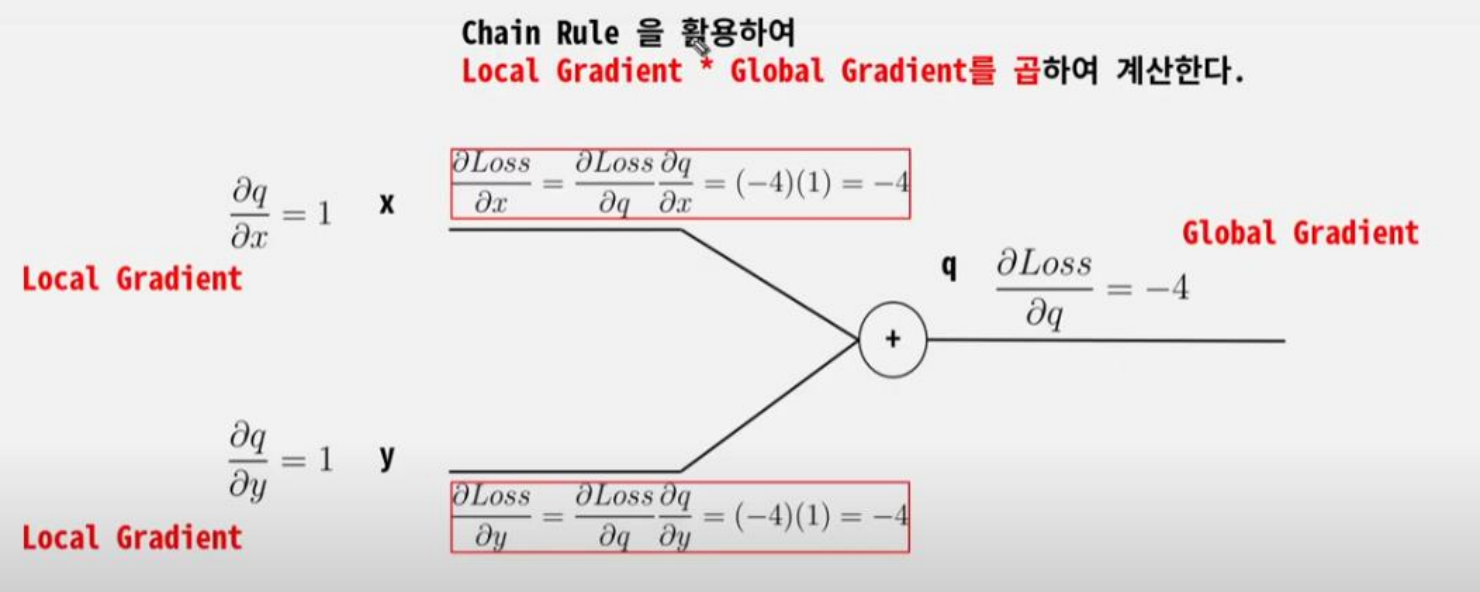
Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = (-4)(1) = -4$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = (-4)(1) = -4$$

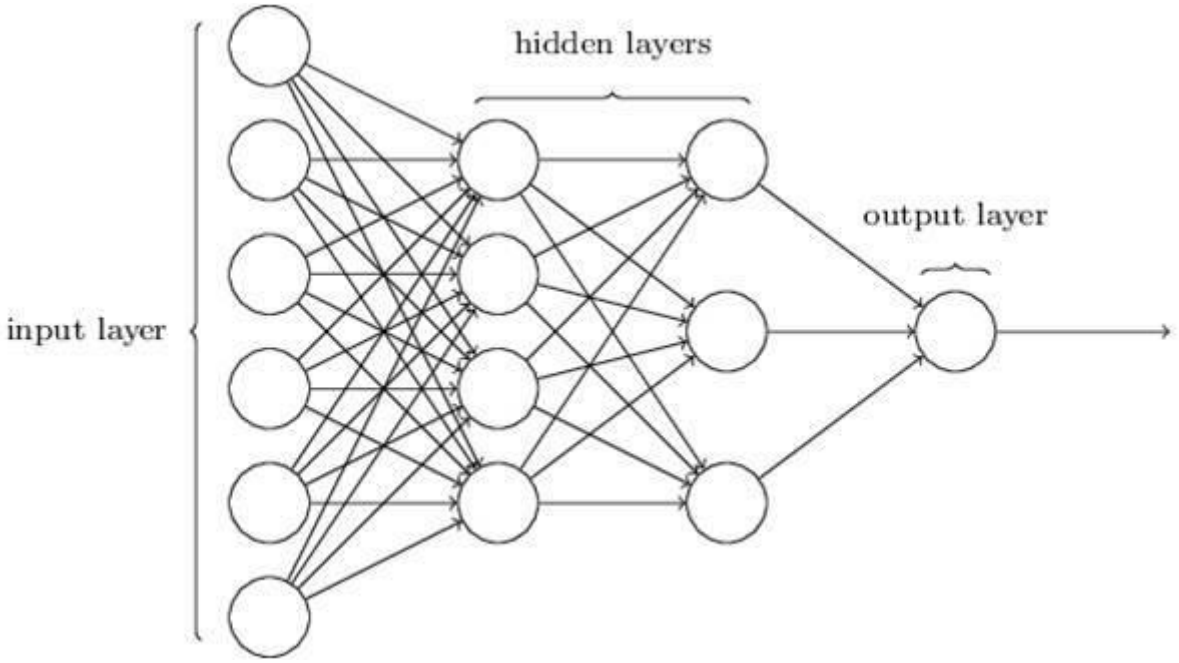
오차 역전파 방법(Backpropagation)-4

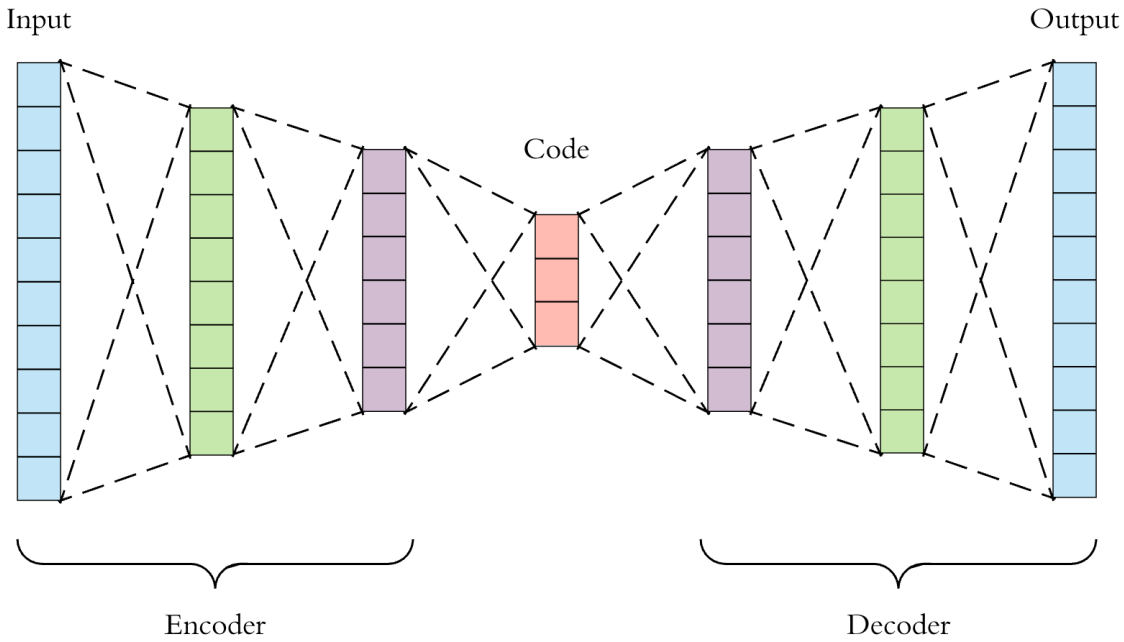




다층 퍼셉트론(MLP: Multi Layer Perceptron)

- 지금까지 우리가 배운 것은 Deep Learning의 기초인 MLP 입니다.
- 선형함수와 활성화 함수를 연결 시켜 층을 만들고, 여러 층을 통해서 복잡한 연산을 합니다.





CNN(Convolutional Neural Network)-1

- 배우 정우성님을 인식하는데 필요한 Feature는 무엇일까요?



CNN(Convolutional Neural Network)-2

- 이미지 처리를 위한 뉴럴 네트워크로 이해하시면 됩니다.

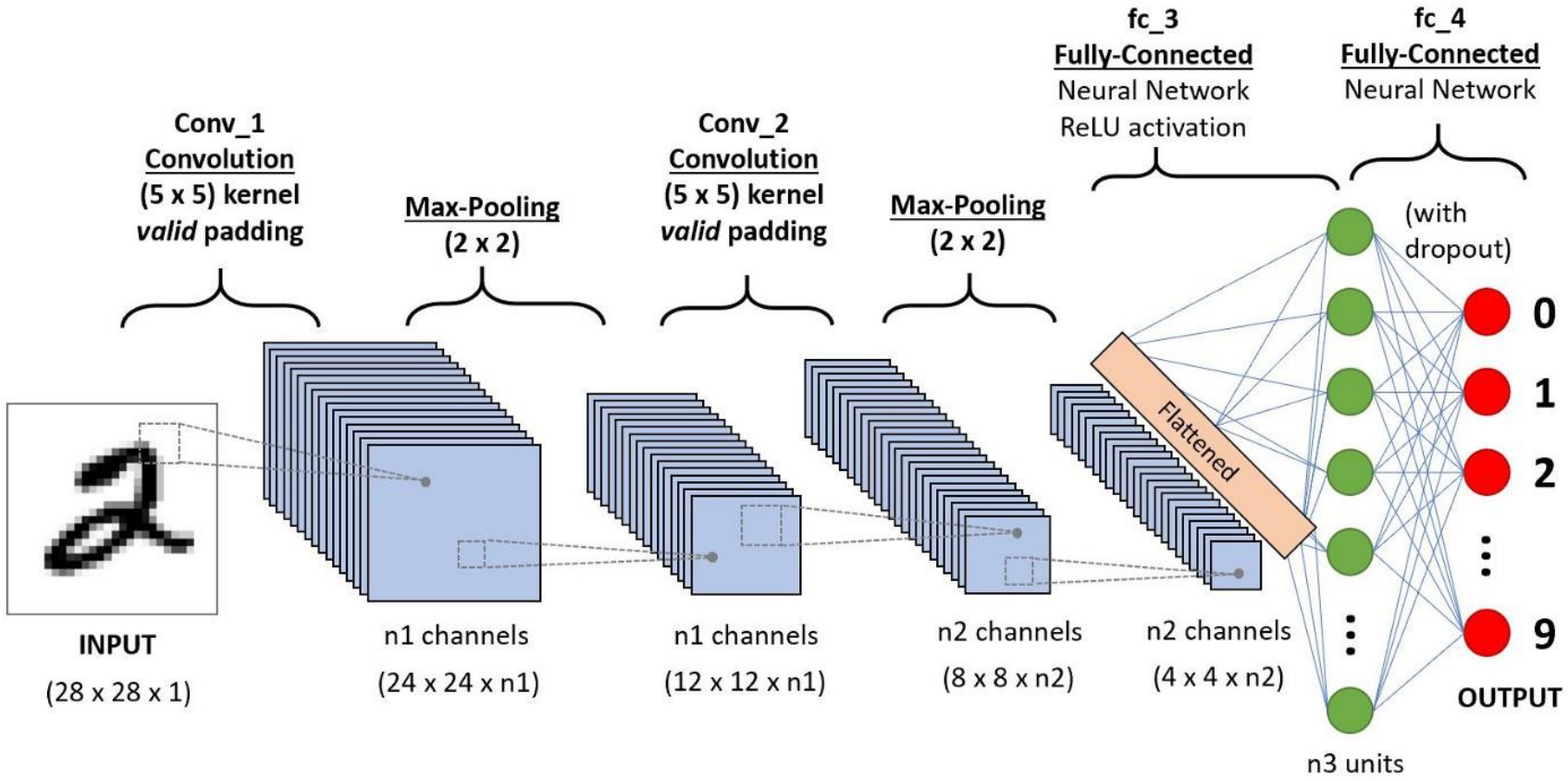
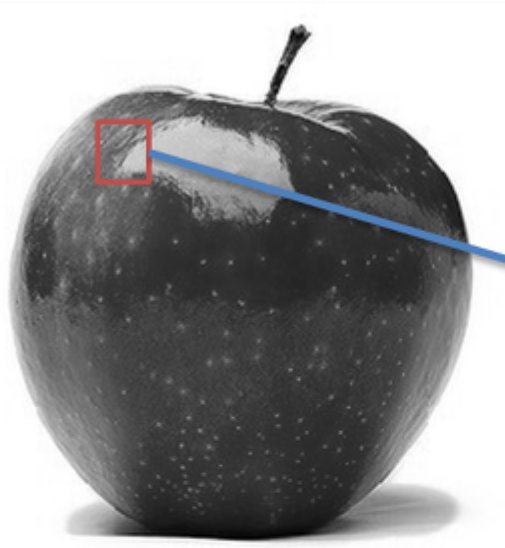


Image 구성

- **Gray Image는 픽셀의 밝기로 구성되어 있습니다. (0~255)**



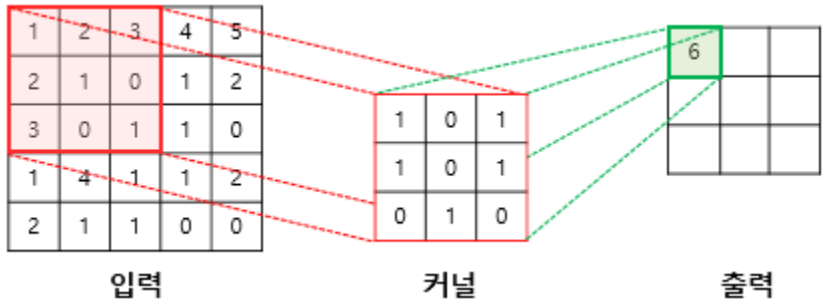
Gray-scale Image

206	225	193	185	182	174	151	137	132	161	140	119	132	120	141	121	125
230	194	191	197	189	160	164	143	156	157	99	160	127	113	121	122	107
195	188	213	185	148	178	153	160	148	116	123	123	155	142	107	151	117
183	191	190	170	177	167	148	164	145	134	127	158	140	112	128	97	146
194	177	189	184	171	139	179	149	108	142	146	140	122	114	115	109	164
177	171	170	175	161	165	172	121	149	153	127	116	131	148	133	133	140
180	177	157	156	168	171	137	145	153	134	138	149	133	128	137	123	119
185	160	171	149	157	142	132	147	124	129	118	138	132	118	165	138	104
165	159	145	176	159	125	159	137	131	142	152	152	116	135	147	106	122
153	180	186	168	139	160	151	158	114	155	172	83	125	154	107	124	152
176	191	153	127	166	140	144	149	164	158	71	184	166	81	147	150	132
177	145	124	151	152	154	140	179	156	92	161	201	108	101	165	128	131
139	131	152	146	140	158	173	159	92	170	171	89	123	161	124	136	99
145	136	169	150	141	134	175	106	158	155	142	121	144	137	102	112	107
141	157	158	121	139	169	137	135	165	124	145	129	105	104	118	112	118
158	149	122	135	153	140	107	156	121	152	156	118	124	129	118	104	94
165	142	145	132	156	117	135	146	127	138	107	95	116	120	102	94	93
130	168	151	132	132	134	125	139	116	132	126	111	129	106	99	102	123
171	173	149	136	133	111	130	121	120	102	104	127	120	111	106	102	118
185	171	150	109	133	125	120	114	105	121	109	111	111	103	115	100	96
181	138	124	129	102	123	107	138	119	101	108	109	114	95	102	109	125
155	137	131	109	114	105	128	119	104	102	103	121	104	129	103	124	110
140	120	139	128	103	116	110	122	110	106	103	112	110	108	124	120	104
119	111	136	112	125	125	122	115	90	119	105	98	132	101	126	91	122
125	127	132	91	134	121	82	117	109	96	97	112	130	109	113	126	129

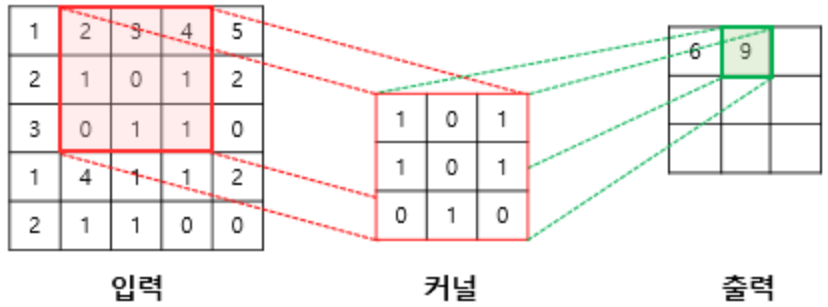
합성곱 연산(Convolution)

```
Conv2D(64, (3, 3), activation='relu')
```

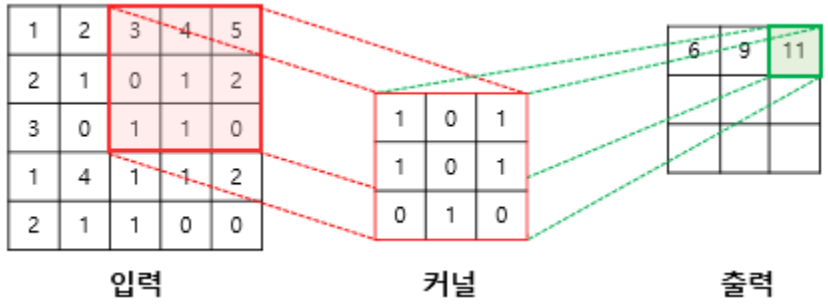
- 큰 영상에서 특징점들만 추출하는 과정



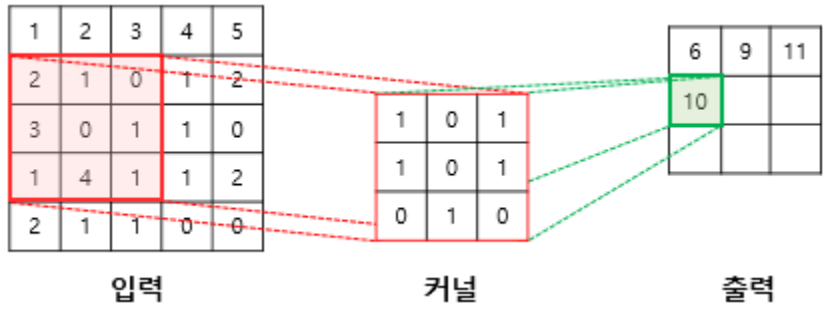
(1×1) + (2×0) + (3×1) + (2×1) + (1×0) + (0×1) + (3×0) + (0×1) + (1×0) = 6



(2×1) + (3×0) + (4×1) + (1×1) + (0×0) + (1×1) + (0×0) + (1×1) + (1×0) = 9



(3×1) + (4×0) + (5×1) + (0×1) + (1×0) + (2×1) + (1×0) + (1×1) + (0×0) = 11

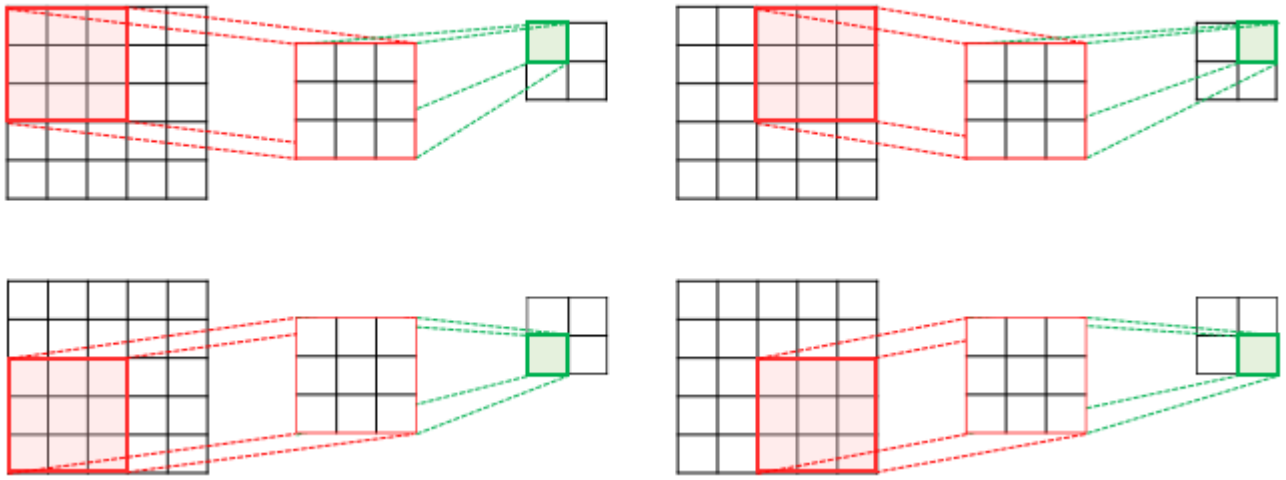


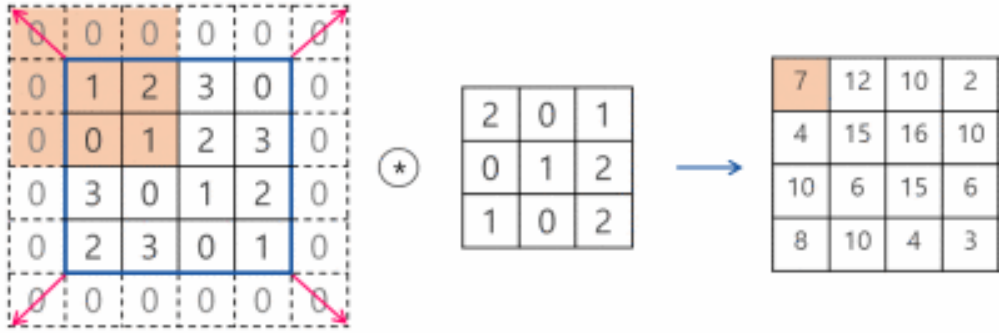
(2×1) + (1×0) + (0×1) + (3×1) + (0×0) + (1×1) + (1×0) + (4×1) + (1×0) = 10

Stride

- 커널이 이동하는 범위를 의미합니다.
- Stride에 따라서 최종 특성 맵의 크기가 달라집니다.

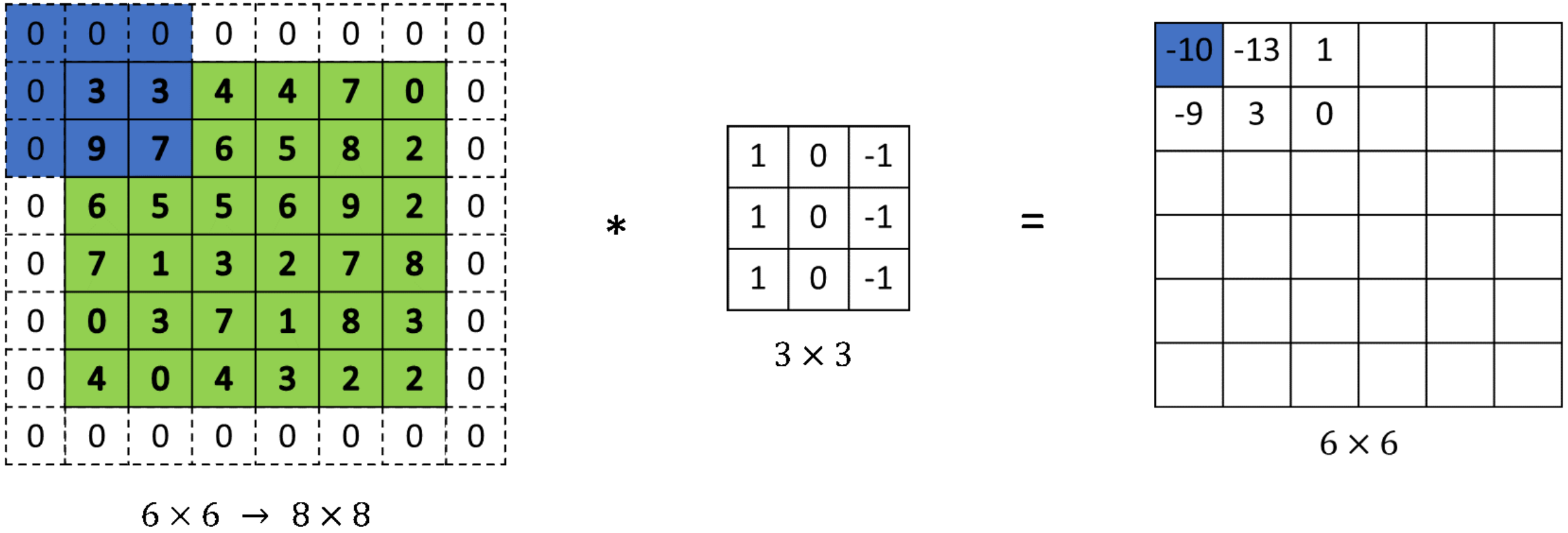
```
Conv2D(input_shape = (10, 10, 3), filters = 10, kernel_size = (3,3), strides = (1,1), padding = 'same')
```





Padding

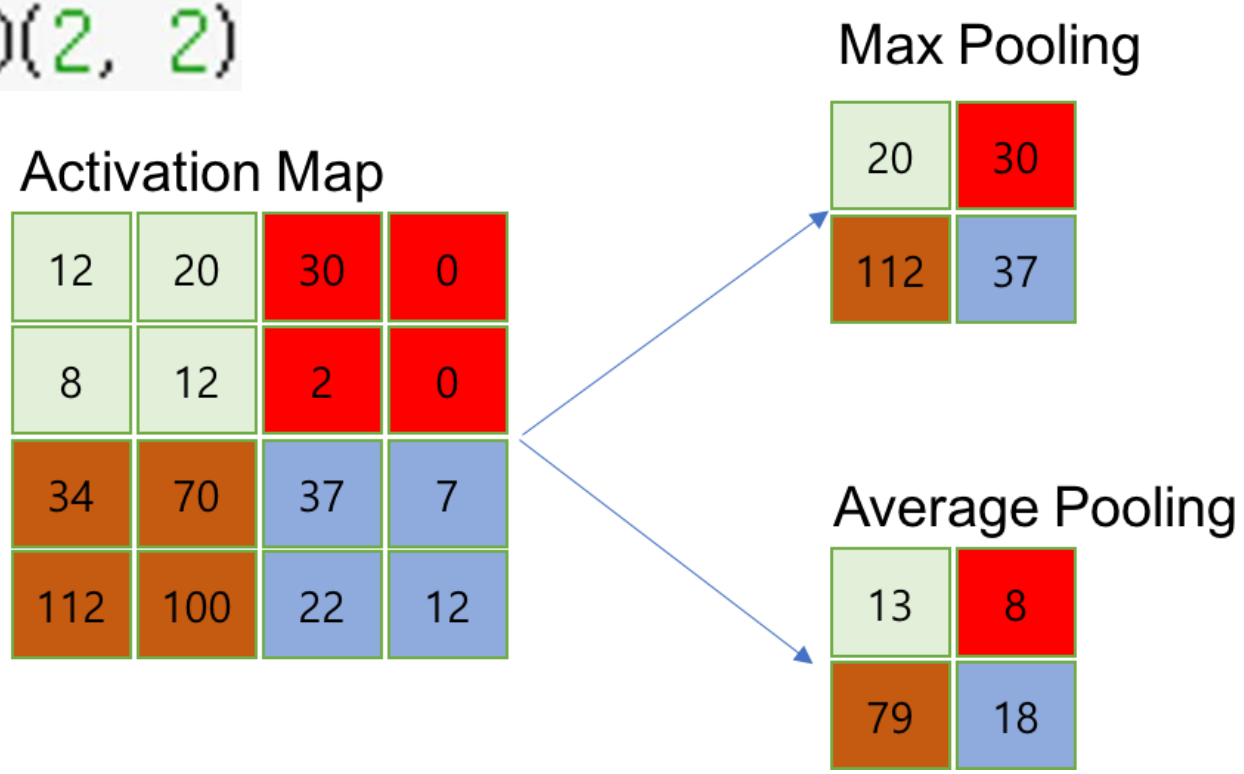
- 이미지의 가장자리를 0으로 채워서 크기를 키우는 작업
- 컨볼루션 연산 후에 아웃풋 이미지 크기를 유지, Edge 정보 활용



Pooling

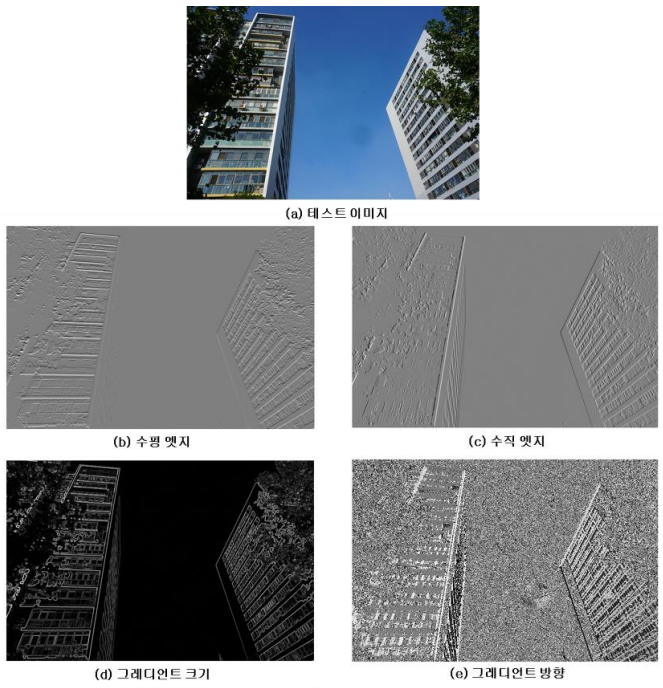
- 필터 사이즈 내에서 가장 큰 값만 선택하거나, 평균을 선택
- 모든 정보가 필요하지 않기 때문에 적당량의 데이터만 선택

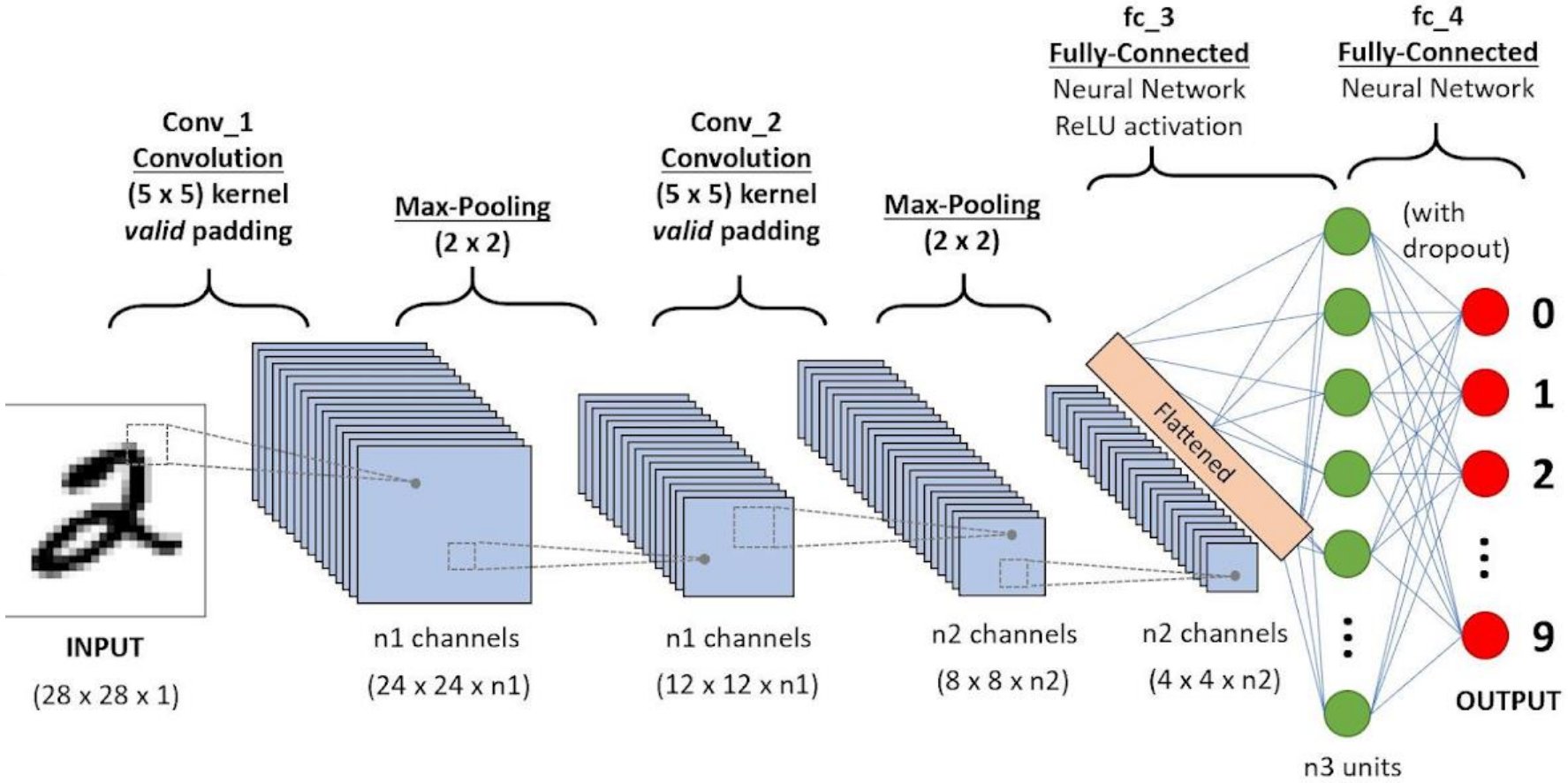
```
MaxPooling2D(2, 2)
```



멀티 필터(커널) 사용

- 다양한 특성을 가지는 필터를 사용하여 연산 진행



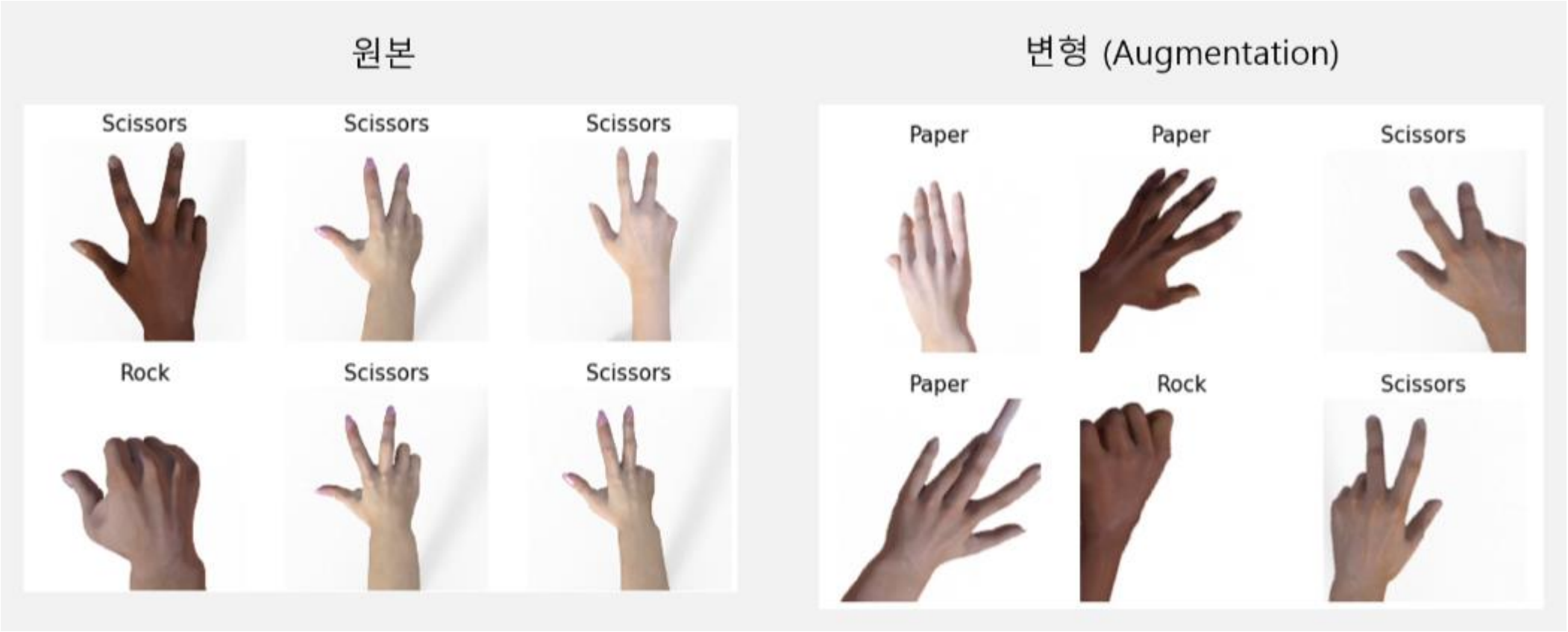


Data Augmentation 필요성

- 같은 사람일까요? Human V.S. AI

Data Augmentation 필요성

- **Rotation, Shift, 굴절, 스케일링, Flip 등…**



Data Augmentation 함수(1)

- **Rotation, Shift, 굴절, 스케일링, Flip 방법에 대한 제공**

```
training_datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2,
)
```

- rescale : 이미지의 픽셀 값을 조정
- rotation_range : 이미지 회전
- width_shift_range : 가로 방향으로 이동
- height_shift_range : 세로 방향으로 이동
- shear_range : 이미지 굴절
- zoom_range : 이미지 확대
- horizontal_flip : 횡 방향으로 이미지 반전
- fill_mode : 이미지를 이동이나 굴절시켰을 때 빈 픽셀 값에 대하여 값을 채우는 방식
- validation_split : train set / validation set 분할 비율

Data Augmentation 함수(2)

- **Rotation, Shift, 굴절, 스케일링, Flip 방법에 대한 제공**

```
training_generator = training_datagen.flow_from_directory(TRAINING_DIR,
                                                          batch_size=128,
                                                          target_size=(150, 150),
                                                          class_mode='categorical',
                                                          subset='training',
                                                         )
```

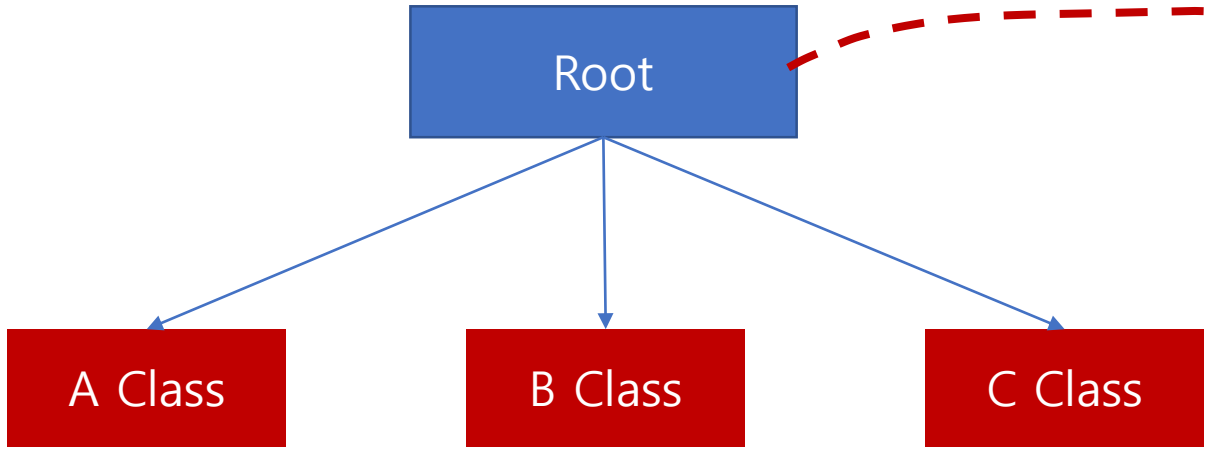
```
Found 2016 images belonging to 3 classes.
```

```
validation_generator = training_datagen.flow_from_directory(TRAINING_DIR,
                                                          batch_size=32,
                                                          target_size=(150, 150),
                                                          class_mode='categorical',
                                                          subset='validation',
                                                         )
```

```
Found 504 images belonging to 3 classes.
```

Image Data Generator 구조

- **Root 폴더를 지정 후, Root 폴더 안에, Class 폴더를 만들어서 저장**
- **Image Data Generator가 자동으로 labeling 진행**




```
training_generator = training_datagen.flow_from_directory(TRAINING_DIR,
                                                          batch_size=128,
                                                          target_size=(150, 150),
                                                          class_mode='categorical',
                                                          subset='training',
                                                         )
```

```
Found 2016 images belonging to 3 classes.
```

```
validation_generator = training_datagen.flow_from_directory(TRAINING_DIR,
                                                            batch_size=32,
                                                            target_size=(150, 150),
                                                            class_mode='categorical',
                                                            subset='validation',
                                                           )
```

```
Found 504 images belonging to 3 classes.
```

실습 - import

```
import urllib.request
import zipfile
import numpy as np
from IPython.display import Image

import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dropout, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint
```

실습 – Load Data

- 가위바위보에 대한 손의 사진을 다운 받아서 특정 폴더에 압축을 해제

```
url = 'https://storage.googleapis.com/download.tensorflow.org/data/rps.zip'
urllib.request.urlretrieve(url, 'rps.zip')
local_zip = 'rps.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('tmp/')
zip_ref.close()
```

실습 – 전처리(ImageDataGenerator)

- **데이터 경로 지정 및, Image Augmentation 진행**

```
TRAINING_DIR = "tmp/rps/"
training_datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2
    )
```

`ImageDataGenerator`를 정의합니다.

다음의 옵션 값들로 Image Aumentation(이미지 변형) 옵션을 적절히 조절해 주세요

- `rescale`: 이미지의 픽셀 값을 조정
- `rotation_range`: 이미지 회전
- `width_shift_range`: 가로 방향으로 이동
- `height_shift_range`: 세로 방향으로 이동
- `shear_range`: 이미지 굴절
- `zoom_range`: 이미지 확대
- `horizontal_flip`: 횡 방향으로 이미지 반전
- `fill_mode`: 이미지를 이동이나 굴절시켰을 때 빈 픽셀 값에 대하여 값을 채우는 방식
- `validation_split`: validation set의 구성 비율

실습 – 전처리(ImageDataGenerator)

- **ImageDataGenerator를 잘 만들어 주었다면, `flow_from_directory`로 이미지를 어떻게 공급해 줄 것인가를 지정해 주어야합니다.**

* train / validation set 전용 generator를 별도로 정의합니다.
* `batch_size`를 정의합니다 (128)
* `target_size`: (150 x 150). 이미지를 알아서 타겟 사이즈 만큼 잘라내어 공급합니다.
* `class_mode`는 3개 이상의 클래스인 경우 'categorical' 이진 분류의 경우 `binary`를 지정합니다.
* `subset`을 지정합니다. (training / validation)

실습 – 전처리(ImageDataGenerator)

training_generator에 대한 `from_from_directory`를 정의합니다.

- 2016 개의 이미지가 출력되어야 합니다.

[코드]

```
[7]  training_generator = training_datagen.flow_from_directory(TRAINING_DIR,
                                                               batch_size=128,
                                                               target_size=(150, 150),
                                                               class_mode='categorical',
                                                               subset='training',
                                                              )
```

```
Found 2016 images belonging to 3 classes.
```

실습 – 모델 정의

```
model = Sequential([
    # Conv2D, MaxPooling2D 조합으로 층을 쌓습니다. 첫번째 입력층의 input_shape은 (150, 150, 3)으로 지정합니다.
    Conv2D(64, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dropout(0.5),
    Dense(512, activation='relu'),
    Dense(3, activation='softmax'),
])
```

실습 – 모델 정의

```
model.summary()
```

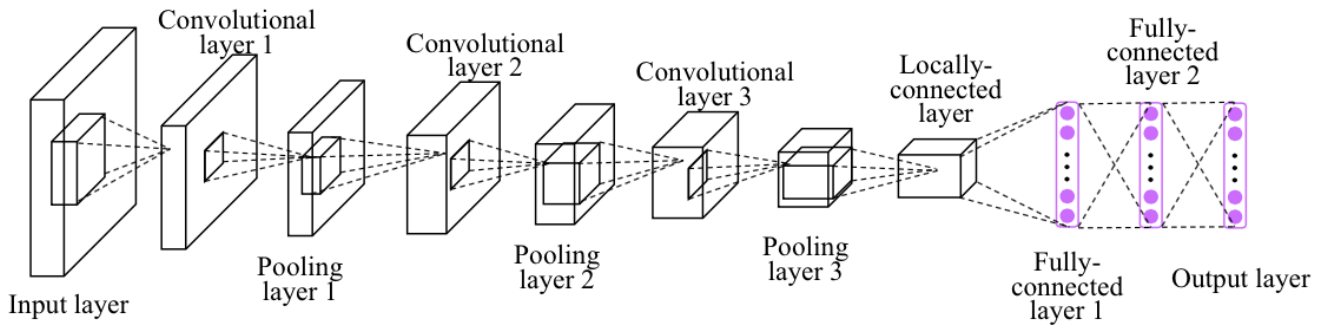
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 64)      1792

max_pooling2d (MaxPooling2D  (None, 74, 74, 64)        0
)

conv2d_1 (Conv2D)            (None, 72, 72, 64)        36928

max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)        0
2D)

conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856

max_pooling2d_2 (MaxPooling  (None, 17, 17, 128)       0
2D)

conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584

max_pooling2d_3 (MaxPooling  (None, 7, 7, 128)         0
2D)

flatten (Flatten)            (None, 6272)              0

dropout (Dropout)            (None, 6272)              0

dense (Dense)                (None, 512)               3211776

dense_1 (Dense)              (None, 3)                 1539

=================================================================
Total params: 3,473,475
Trainable params: 3,473,475
Non-trainable params: 0
_________________________________________________________________
```

param_number = output_channel_number * (input_channel_number * kernel_height * kernel_width + 1)

```
print(64*(3*3*3+1))
```

```
1792
```

- Color 채널이 있을 경우 모델



실습 – 컴파일 & ModelCheckpoint

STEP 4. 컴파일 (compile)

+ 코드 + 텍스트

1. `optimizer`는 가장 최적화가 잘되는 알고리즘인 'adam'을 사용합니다.
2. `loss`는 무엇을 지정하면 좋을까요? (`categorical_crossentropy` / `sparse_categorical_crossentropy`)
3. `metrics`를 'acc' 혹은 'accuracy'로 지정하면, 학습시 정확도를 모니터링 할 수 있습니다.

```
[ ]  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
```

STEP 5. ModelCheckpoint

`val_loss` 기준으로 epoch 마다 최적의 모델을 저장하기 위하여, ModelCheckpoint를 만듭니다.

- `checkpoint_path`는 모델이 저장될 파일 명을 설정합니다.
- `ModelCheckpoint`을 선언하고, 적절한 옵션 값을 지정합니다.

[코드]

```
[ ]  checkpoint_path = "tmp_checkpoint.ckpt"
     checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                                  save_weights_only=True,
                                  save_best_only=True,
                                  monitor='val_loss',
                                  verbose=1)
```

실습 – 학습(fit)

▾ STEP 6. 학습 (fit)

```
[14] model.fit(training_generator,
               validation_data=(validation_generator),
               epochs=25,
               callbacks=[checkpoint],
               )
```

```
16/16 [==============================] - 20s 1s/step - loss: 0.1566 - acc: 0.9549 - val_loss: 1.3782 - val_acc: 0.6230
Epoch 12/25
16/16 [==============================] - ETA: 0s - loss: 0.1602 - acc: 0.9489
Epoch 12: val_loss did not improve from 0.62882
16/16 [==============================] - 20s 1s/step - loss: 0.1602 - acc: 0.9489 - val_loss: 1.0569 - val_acc: 0.6448
Epoch 13/25
16/16 [==============================] - ETA: 0s - loss: 0.1165 - acc: 0.9653
Epoch 13: val_loss improved from 0.62882 to 0.56866, saving model to tmp_checkpoint.ckpt
16/16 [==============================] - 20s 1s/step - loss: 0.1165 - acc: 0.9653 - val_loss: 0.5687 - val_acc: 0.8135
Epoch 14/25
16/16 [==============================] - ETA: 0s - loss: 0.0909 - acc: 0.9678
Epoch 14: val_loss did not improve from 0.56866
16/16 [==============================] - 21s 1s/step - loss: 0.0909 - acc: 0.9678 - val_loss: 0.9134 - val_acc: 0.7480
Epoch 15/25
16/16 [==============================] - ETA: 0s - loss: 0.0867 - acc: 0.9707
Epoch 15: val_loss did not improve from 0.56866
16/16 [==============================] - 20s 1s/step - loss: 0.0867 - acc: 0.9707 - val_loss: 0.9273 - val_acc: 0.7837
Epoch 16/25
16/16 [==============================] - ETA: 0s - loss: 0.0806 - acc: 0.9762
Epoch 16: val_loss did not improve from 0.56866
16/16 [==============================] - 20s 1s/step - loss: 0.0806 - acc: 0.9762 - val_loss: 1.2571 - val_acc: 0.6111
Epoch 17/25
16/16 [==============================] - ETA: 0s - loss: 0.0653 - acc: 0.9797
Epoch 17: val_loss did not improve from 0.56866
```

실습 – Load Weights

STEP 7. 학습 완료 후 Load Weights (ModelCheckpoint)

학습이 완료된 후에는 반드시 `load_weights`를 해주어야 합니다.

그렇지 않으면, 열심히 ModelCheckpoint를 만든 의미가 없습니다.

[코드]

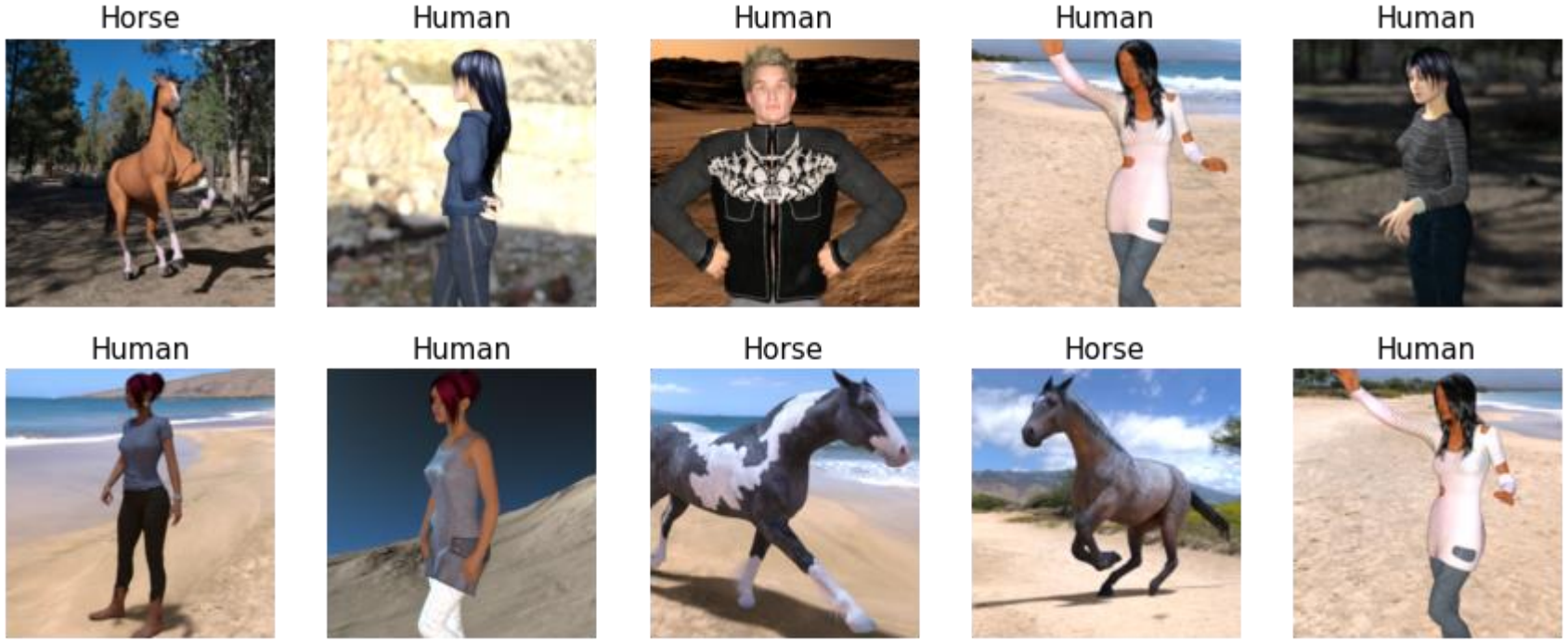
```
[21] model.load_weights(checkpoint_path)
```

0초

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fca4e797e50>
```

Data Augmentation 새로운 문제

- 사람인지 말인지 분류하는 문제



Training Data와 Validation Data가 각자 나누어져 있을 때

- 가장 먼저 중요한 패키지를 import 합니다.

STEP 1. import

```
import urllib.request
import zipfile
import numpy as np
from IPython.display import Image

import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint
```

데이터 로드 부분

_TRAIN_URL = "https://storage.googleapis.com/download.tensorflow.org/data/horse-or-human.zip"
_TEST_URL = "https://storage.googleapis.com/download.tensorflow.org/data/validation-horse-or-human.zip"

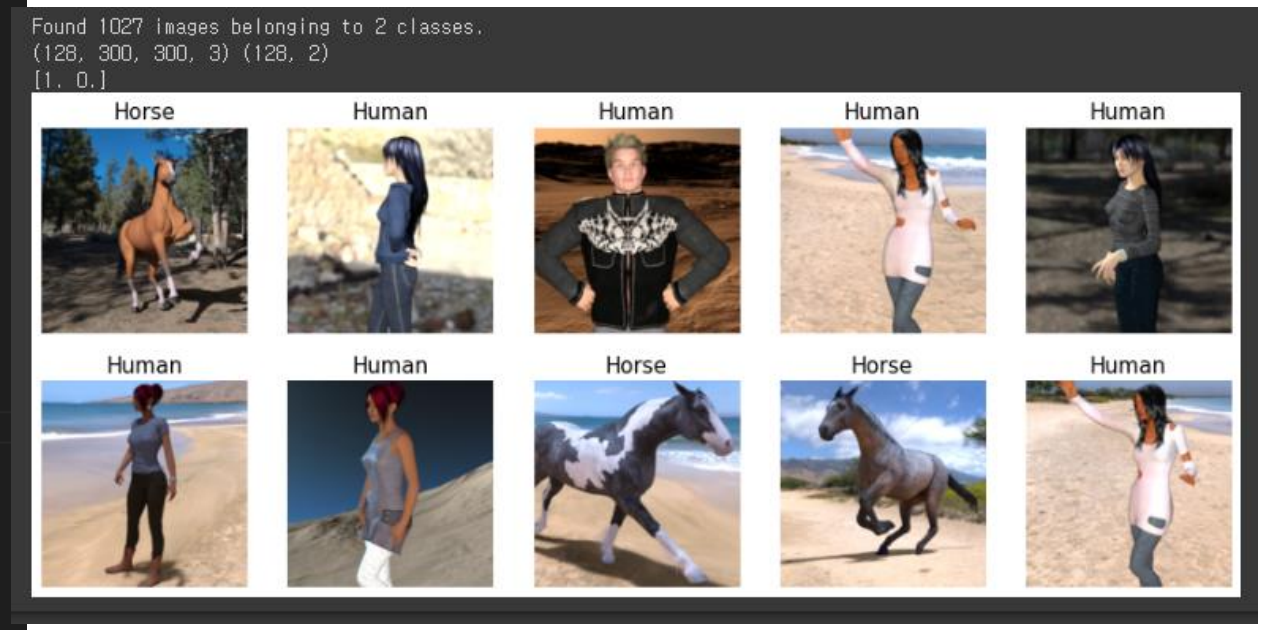
```
_TRAIN_URL = "https://storage.googleapis.com/download.tensorflow.org/data/horse-or-human.zip"
_TEST_URL = "https://storage.googleapis.com/download.tensorflow.org/data/validation-horse-or-human.zip"

urllib.request.urlretrieve(_TRAIN_URL, 'horse-or-human.zip')

local_zip = 'horse-or-human.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('tmp/horse-or-human/')
zip_ref.close()

urllib.request.urlretrieve(_TEST_URL, 'validation-horse-or-human.zip')
local_zip = 'validation-horse-or-human.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('tmp/validation-horse-or-human/')
zip_ref.close()
```

로드된 데이터 확인하기 (필수는 아님)

- 데이터가 어떻게 되어 있는지 살펴 봅니다.

```python
import matplotlib.pyplot as plt

class_map = {
    0: 'Horse',
    1: 'Human',
}

original_datagen = ImageDataGenerator(rescale=1./255)
original_generator = original_datagen.flow_from_directory('tmp/horse-or-human/',
                                                          batch_size=128,
                                                          target_size=(300, 300),
                                                          class_mode='categorical'
                                                         )

for x, y in original_generator:
    print(x.shape, y.shape)
    print(y[0])

    fig, axes = plt.subplots(2, 5)
    fig.set_size_inches(15, 6)
    for i in range(10):
        axes[i//5, i%5].imshow(x[i])
        axes[i//5, i%5].set_title(class_map[y[i].argmax()], fontsize=15)
        axes[i//5, i%5].axis('off')
    break
plt.show()
```

```
Found 1027 images belonging to 2 classes.
(128, 300, 300, 3) (128, 2)
[1. 0.]
```



전처리 (ImageDataGenerator)

```
[4]  TRAINING_DIR = 'tmp/horse-or-human/'
     VALIDATION_DIR = 'tmp/validation-horse-or-human/'
```

`ImageDataGenerator`를 정의합니다.

다음의 옵션 값들로 Image Aumentation(이미지 변형) 옵션을 적절히 조절해 주세요

- `rescale`: 이미지의 픽셀 값을 조정
- `rotation_range`: 이미지 회전
- `width_shift_range`: 가로 방향으로 이동
- `height_shift_range`: 세로 방향으로 이동
- `shear_range`: 이미지 굴절
- `zoom_range`: 이미지 확대
- `horizontal_flip`: 횡 방향으로 이미지 반전
- `fill_mode`: 이미지를 이동이나 굴절시켰을 때 빈 픽셀 값에 대하여 값을 채우는 방식
- `validation_split`: validation set의 구성 비율

[코드]

```
[5]  training_datagen = ImageDataGenerator(
         rescale=1/ 255.0,

     )
```

```
[6]  validation_datagen = ImageDataGenerator(
         rescale=1/ 255.0,
     )
```

TRAINING_DIR = 'tmp/horse-or-human/'
VALIDATION_DIR = 'tmp/validation-horse-or-human/'

```
training_datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2,
)
```

STEP 2. 전처리 (ImageDataGenerator)

- **ImageDataGenerator를 잘 만들어 주었다면, flow_from_directory로 이미지를 어떻게 공급해 줄 것인가를 지정해 주어야합니다.**
- train / validation set 전용 generator를 별도로 정의합니다.
- batch_size를 정의합니다.
- target_size를 정의합니다. (300 x 300). 이미지를 알아서 타겟 사이즈 만큼 잘라내어 공급합니다.
- class_mode는 3개 이상의 클래스인 경우 'categorical', 이진 분류의 경우 binary를 지정합니다.
- subset을 지정합니다. (training / validation)

training_generator에 대한 `from_from_directory`를 정의합니다.

[코드]

```
train_generator = training_datagen.flow_from_directory(
    TRAINING_DIR,
    target_size=(300, 300),
    batch_size=32,
    class_mode='categorical',
)
```

```
Found 1027 images belonging to 2 classes.
```

validation_generator에 대한 `from_from_directory`를 정의합니다.

[코드]

```
[8] validation_generator = validation_datagen.flow_from_directory(
        VALIDATION_DIR,
        target_size=(300, 300),
        batch_size=32,
        class_mode='categorical',
    )
```

```
Found 256 images belonging to 2 classes.
```

모델 정의

- 기존 모델과 동일하게 만드시면 됩니다.

```
[9]  model = Sequential([
        Conv2D(16, (3, 3), activation='relu', input_shape=(300, 300, 3)),
        MaxPooling2D(2, 2),
        Conv2D(32, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dropout(0.5),
        Dense(128, activation='relu'),
        Dense(32, activation='relu'),
        Dense(2, activation='softmax')
    ])
```

```
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 16)      448

 max_pooling2d (MaxPooling2D  (None, 149, 149, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 147, 147, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 73, 73, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 71, 71, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 35, 35, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 33, 33, 64)        36928

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 14, 14, 128)       73856

 max_pooling2d_4 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 flatten (Flatten)           (None, 6272)              0

 dropout (Dropout)           (None, 6272)              0

 dense (Dense)               (None, 128)               802944

 dense_1 (Dense)             (None, 32)                4128

 dense_2 (Dense)             (None, 2)                 66

=================================================================
Total params: 941,506
Trainable params: 941,506
Non-trainable params: 0
_________________________________________________________________
```

컴파일 및 ModelCheckpoint 설정

- 모델-컴파일-학습 순서는 동일합니다.

STEP 4. 컴파일 (compile)

1. `optimizer`는 가장 최적화가 잘되는 알고리즘인 'adam'을 사용합니다.
2. `loss`는 무엇을 지정하면 좋을까요?
3. `metrics`를 'acc' 혹은 'accuracy'로 지정하면, 학습시 정확도를 모니터링 할 수 있습니다.

[코드]

```
[11] model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
```

STEP 5. ModelCheckpoint

`val_loss` 기준으로 epoch 마다 최적의 모델을 저장하기 위하여, ModelCheckpoint를 만듭니다.

- `checkpoint_path`는 모델이 저장될 파일 명을 설정합니다.
- `ModelCheckpoint`을 선언하고, 적절한 옵션 값을 지정합니다.

[코드]

```
[12] checkpoint_path = "tmp_checkpoint.ckpt"
     checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                                  save_weights_only=True,
                                  save_best_only=True,
                                  monitor='val_loss',
                                  verbose=1)
```

학습 및 학습 파라미터 로드

[코드]

```
[18] model.fit(train_generator,
               validation_data=(validation_generator),
               epochs=5,
               callbacks=[checkpoint],
               )
```

```
Epoch 1/5
33/33 [==============================] - ETA: 0s - loss: 1.0599e-04 - acc: 1.0000
Epoch 1: val_loss improved from inf to 2.75170, saving model to tmp_checkpoint.ckpt
33/33 [==============================] - 8s 256ms/step - loss: 1.0599e-04 - acc: 1.0000 - val_loss: 2.7517 - val_acc: 0.8594
Epoch 2/5
33/33 [==============================] - ETA: 0s - loss: 7.4062e-05 - acc: 1.0000
Epoch 2: val_loss did not improve from 2.75170
33/33 [==============================] - 8s 243ms/step - loss: 7.4062e-05 - acc: 1.0000 - val_loss: 2.9316 - val_acc: 0.8438
Epoch 3/5
33/33 [==============================] - ETA: 0s - loss: 6.6064e-05 - acc: 1.0000
Epoch 3: val_loss did not improve from 2.75170
33/33 [==============================] - 8s 239ms/step - loss: 6.6064e-05 - acc: 1.0000 - val_loss: 2.8137 - val_acc: 0.8555
Epoch 4/5
33/33 [==============================] - ETA: 0s - loss: 4.8783e-05 - acc: 1.0000
Epoch 4: val_loss did not improve from 2.75170
33/33 [==============================] - 8s 237ms/step - loss: 4.8783e-05 - acc: 1.0000 - val_loss: 3.1265 - val_acc: 0.8359
Epoch 5/5
33/33 [==============================] - ETA: 0s - loss: 5.1654e-05 - acc: 1.0000
Epoch 5: val_loss did not improve from 2.75170
33/33 [==============================] - 8s 242ms/step - loss: 5.1654e-05 - acc: 1.0000 - val_loss: 2.8622 - val_acc: 0.8516
<keras.callbacks.History at 0x7f59e4615dc0>
```

STEP 7. 학습 완료 후 Load Weights (ModelCheckpoint)

학습이 완료된 후에는 반드시 `load_weights`를 해주어야 합니다.

그렇지 않으면, 열심히 ModelCheckpoint를 만든 의미가 없습니다.

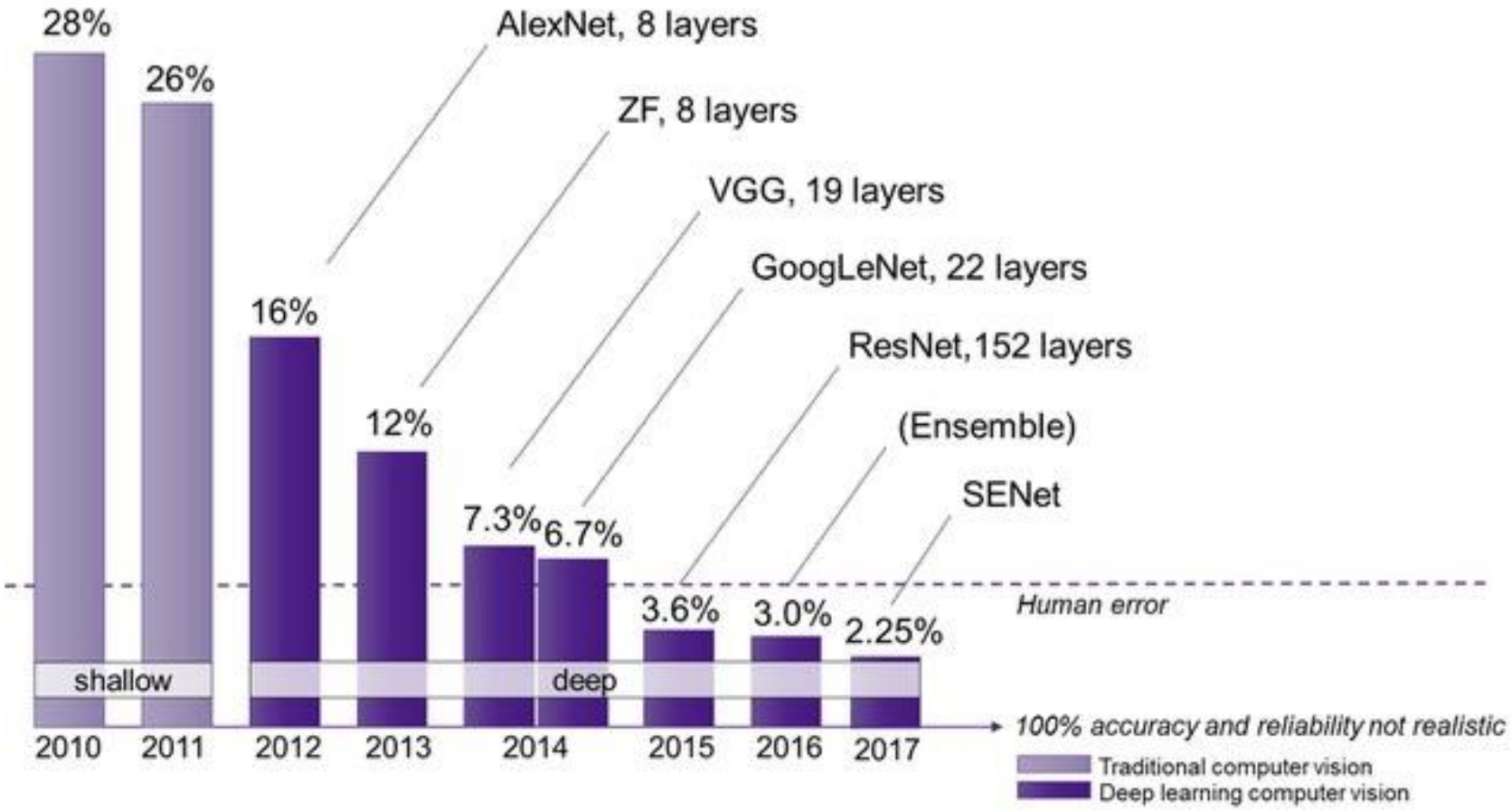
[코드]

```
[ ] model.load_weights(checkpoint_path)
```

ImageNet

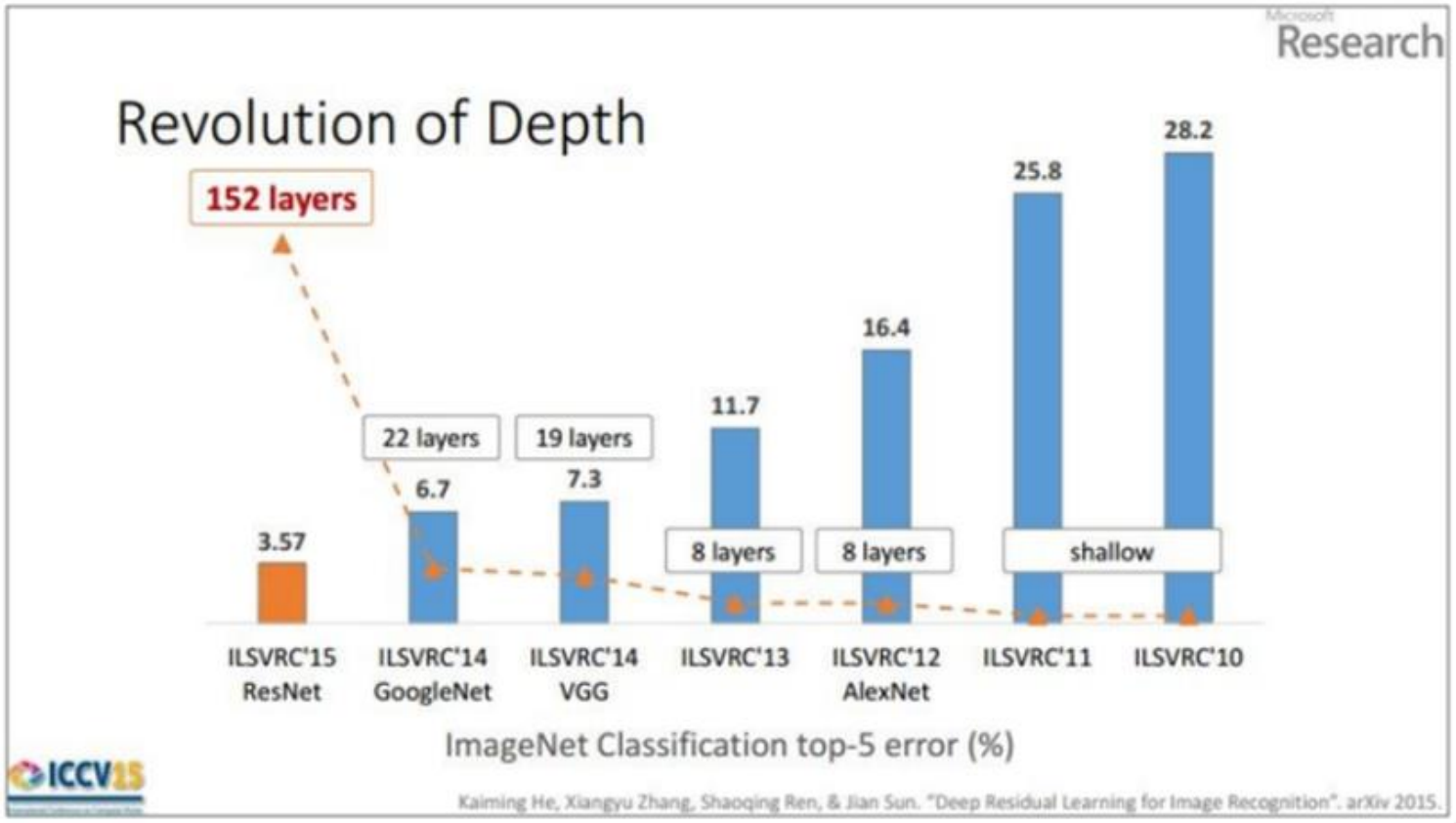
https://golden.com/wiki/ImageNet-38DEAM

- **1,000개의 Label을 가진 이미지 데이터 데이터 베이스로 경진대회에 활용**



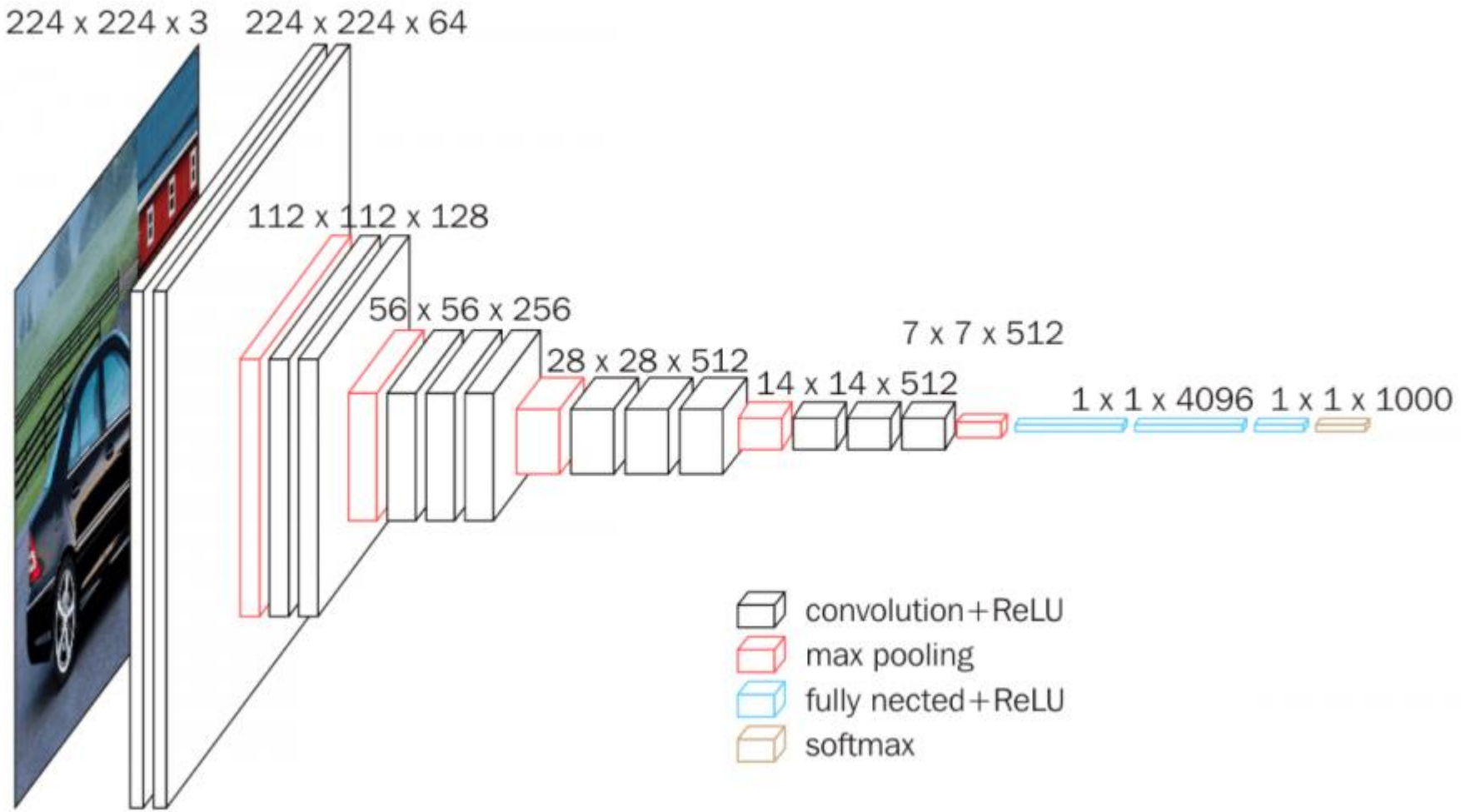
VGG Net

- **VGG Net 이후 부터 Layer의 깊이가 깊어지기 시작함**



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

실습 – 모델 정의


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution+ReLU
max pooling
fully nected+ReLU
softmax

VGG Net 활용하기

- **applications 패키지 안에 VGG16을 import 합니다.**
- weights='imagenet' : 기존 학습 파라미터 활용
- include_top=False: 마지막 층은 사용하지 않음 (1000개 분류하지 않을 때)
- transfer_model.trainable=False: VGG Net의 파라미터는 학습하지 않음

```
from tensorflow.keras.applications import VGG16
```

```
transfer_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
transfer_model.trainable=False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 3s 0us/step
```

```
[8] model = Sequential([
        transfer_model,
        Flatten(),
        Dropout(0.5),
        Dense(512, activation='relu'),
        Dense(128, activation='relu'),
        Dense(2, activation='softmax'),
    ])
```

VGG Net 형태

- **VGG Net에 속한 파라미터는 학습하지 않습니다.**
- **Non-trainable params: 14,714,88 참고**

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dropout (Dropout)           (None, 25088)             0

 dense (Dense)               (None, 512)               12845568

 dense_1 (Dense)             (None, 128)               65664

 dense_2 (Dense)             (None, 2)                 258

=================================================================
Total params: 27,626,178
Trainable params: 12,911,490
Non-trainable params: 14,714,688
_________________________________________________________________
```

실습 (1) : 주요 패키지 import & 데이터 로드

- **import 패키지**

```
[1]  import tensorflow_datasets as tfds
     import tensorflow as tf

     from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.callbacks import ModelCheckpoint
     from tensorflow.keras.applications import VGG16
```

- **tfds로 부터 데이터 다운로드**

```
[2]  dataset_name = 'cats_vs_dogs'

     # 처음 80%의 데이터만 사용
     train_dataset = tfds.load(name=dataset_name, split='train[:80%]')

     # 최근 20%의 데이터만 사용
     valid_dataset = tfds.load(name=dataset_name, split='train[80%:]')
```

실습 (2) : 데이터 전처리

- 데이터 전처리

```
def preprocess(data):
    # x, y 데이터를 정의합니다.
    x = data['image']
    y = data['label']
    # image 정규화(Normalization)
    x = x / 255
    # 사이즈를 (224, 224)로 변환합니다.
    x = tf.image.resize(x, size=(224, 224))
    # x, y  데이터를 return 합니다.
    return x, y
```

만든 전처리 함수(preprocessing)를 **dataset에 mapping**하고, **batch_size**도 지정합니다.

```
[4]  batch_size=32
```

```
[5]  train_data = train_dataset.map(preprocess).batch(batch_size)
     valid_data = valid_dataset.map(preprocess).batch(batch_size)
```

실습 (3) : 모델 정의

모델 정의 (Sequential)

이제 Modeling을 할 차례입니다.

`Sequential` 모델 안에서 층을 깊게 쌓아 올려 주면 됩니다.

1. `input_shape`는 224 X 224 컬러사진인 **(224, 224, 3)**으로 지정합니다.
2. transfer learning 기법을 통해 VGG16 모델을 활용한 전이학습 모델을 완성합니다.
3. 출력층은 class 갯수 2개의 뉴런이 요구됩니다.

```
[6]  transfer_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
     transfer_model.trainable=False
```

```
[7]  model = Sequential([
         transfer_model,
         Flatten(),
         Dropout(0.5),
         Dense(512, activation='relu'),
         Dense(128, activation='relu'),
         Dense(2, activation='softmax'),
     ])
```

실습 (4) : 컴파일 및 ModelCheckpoint 설정

컴파일 (compile)

1. `optimizer`는 가장 최적화가 잘되는 알고리즘인 'adam'을 사용합니다.
2. `loss`설정
   - 출력층 activation이 `sigmoid` 인 경우: `binary_crossentropy`
   - 출력층 activation이 `softmax` 인 경우:
     - 원핫인코딩(O): `categorical_crossentropy`
     - 원핫인코딩(X): `sparse_categorical_crossentropy`)
3. `metrics`를 'acc' 혹은 'accuracy'로 지정하면, 학습시 정확도를 모니터링 할 수 있습니다.

전처리 단계에서 **one-hot encoding** 을 해주었습니다. 따라서, `categorical_crossentropy`를 지정해주면 됩니다.

model.compile()

```
[9]  model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])
```

```
[10] checkpoint_path = "my_checkpoint.ckpt"
     checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                                  save_weights_only=True,
                                  save_best_only=True,
                                  monitor='val_loss',
                                  verbose=1)
```

실습 (5) : 학습 및 파라미터 로드

```
model.fit(train_data,
          validation_data=(valid_data),
          epochs=3,
          callbacks=[checkpoint],
          )
```

```
Epoch 1/20
582/582 [==============================] - ETA: 0s - loss: 0.2813 - acc: 0.8884
Epoch 1: val_loss improved from inf to 0.20612, saving model to my_checkpoint.ckpt
582/582 [==============================] - 128s 202ms/step - loss: 0.2813 - acc: 0.8884 - val_loss: 0.2061 - val_acc: 0.9093
Epoch 2/20
582/582 [==============================] - ETA: 0s - loss: 0.1864 - acc: 0.9223
Epoch 2: val_loss improved from 0.20612 to 0.17423, saving model to my_checkpoint.ckpt
582/582 [==============================] - 114s 196ms/step - loss: 0.1864 - acc: 0.9223 - val_loss: 0.1742 - val_acc: 0.9267
Epoch 3/20
582/582 [==============================] - ETA: 0s - loss: 0.1637 - acc: 0.9336
Epoch 3: val_loss did not improve from 0.17423
582/582 [==============================] - 113s 195ms/step - loss: 0.1637 - acc: 0.9336 - val_loss: 0.1891 - val_acc: 0.9168
```

학습 완료 후 Load Weights (ModelCheckpoint)

학습이 완료된 후에는 반드시 `load_weights`를 해주어야 합니다.

그렇지 않으면, 열심히 ModelCheckpoint를 만든 의미가 없습니다.

```
[ ] # checkpoint 를 저장한 파일명을 입력합니다.
    model.load_weights(checkpoint_path)
```