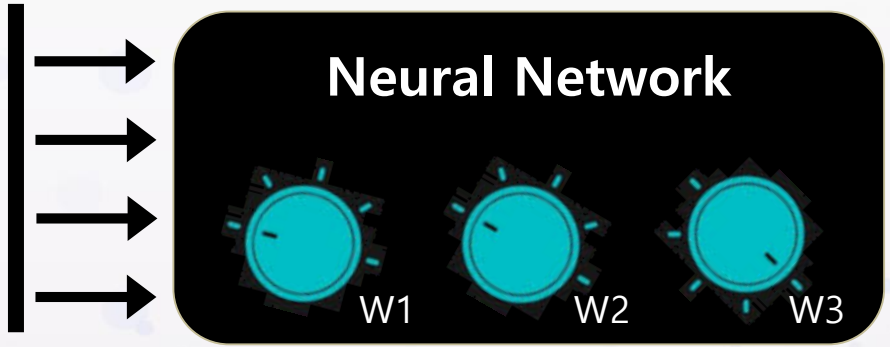
# 참조. 딥러닝 작동 원리

## 케라스 모델 프로세스







학습 데이터

기계 학습 [예측]

전체 데이터

Training-Data
(약 70%)

Test-Data
(약 30%)

모델 생성/
훈련

훈련결과

ML-Predict

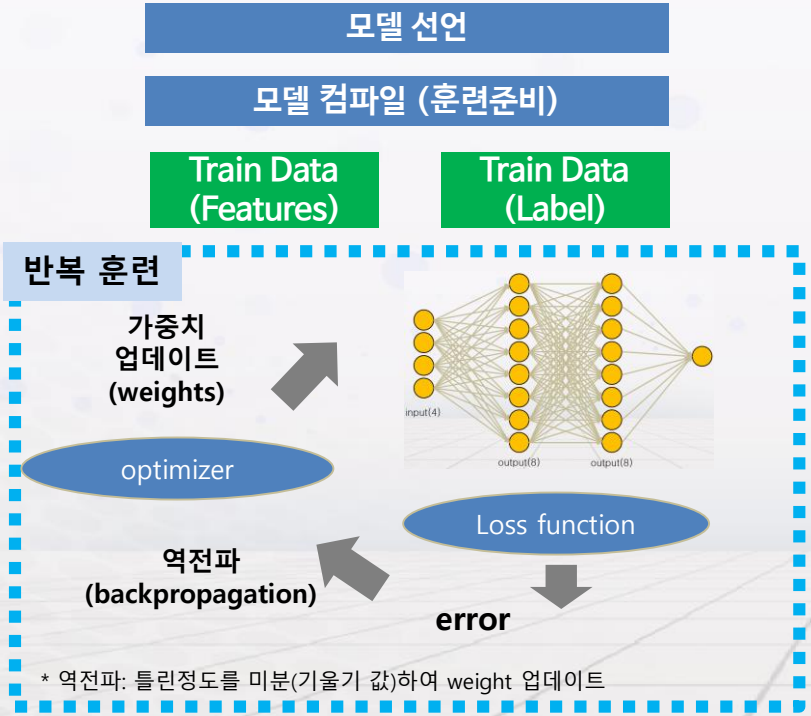
Feature는 숫자형태로
변경 필요!

테스트 데이터

반복 훈련 (feature, label)
loss function!

# 참조. 딥러닝 작동 원리

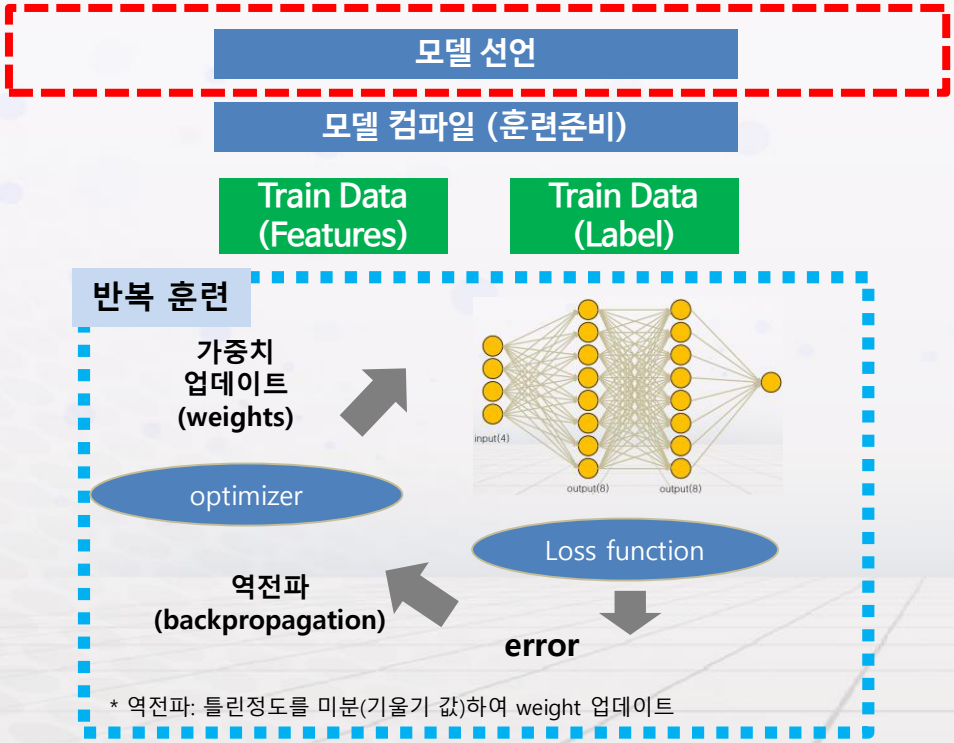
## 케라스 모델 프로세스




| 주요용어 | 내용 |
|---|---|
| 모델선언 | Kears에서 Layer 적용 모델 선언 |
| 컴파일 | 손실함수 및 최적화방법 설정<br>- 손실함수: 훈련동안 최적화될 지표<br>- 최적화방법: 답이 틀렸을 경우 코칭방법 설정<br>(단 훈련셋에 검증구간을 별도 분리) |
| 훈련 | 훈련데이터의 Feature와 Label을 활용하여 손실함수 지표를 최적화하기위하여 역전파(손실함수 결과 개선을 위해 가중치 수정) 반복 수행 |

# 참조. 딥러닝 작동 원리

## 케라스 모델 프로세스

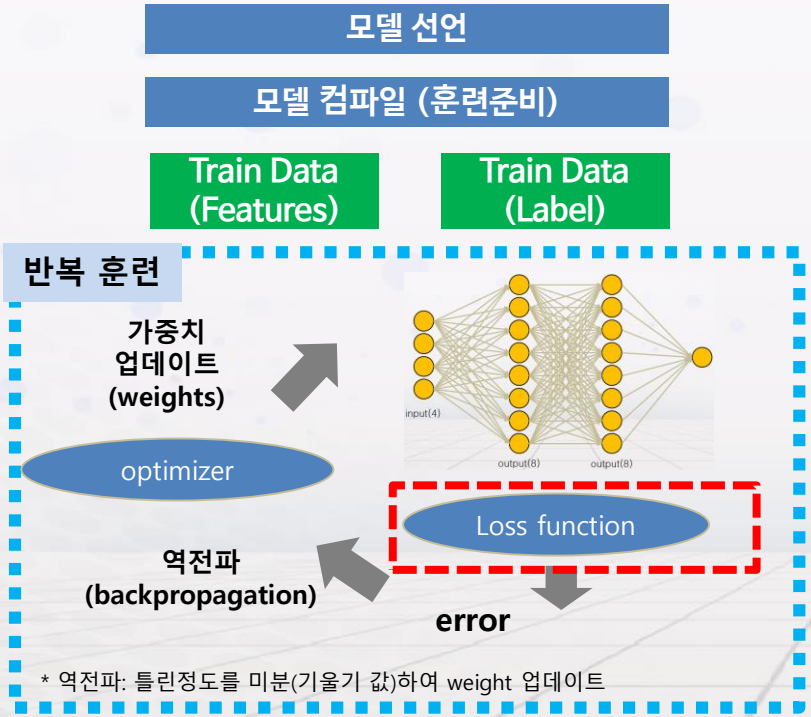



| Model | Size | Top-1 Accuracy | Top-5 Accuracy | Parameters | Depth |
|---|---|---|---|---|---|
| Xception | 88 MB | 0.790 | 0.945 | 22,910,480 | 126 |
| VGG16 | 528 MB | 0.713 | 0.901 | 138,357,544 | 23 |
| VGG19 | 549 MB | 0.713 | 0.900 | 143,667,240 | 26 |
| ResNet50 | 98 MB | 0.749 | 0.921 | 25,636,712 | - |
| ResNet101 | 171 MB | 0.764 | 0.928 | 44,707,176 | - |
| ResNet152 | 232 MB | 0.766 | 0.931 | 60,419,944 | - |
| ResNet50V2 | 98 MB | 0.760 | 0.930 | 25,613,800 | - |
| ResNet101V2 | 171 MB | 0.772 | 0.938 | 44,675,560 | - |
| ResNet152V2 | 232 MB | 0.780 | 0.942 | 60,380,648 | - |
| ResNeXt50 | 96 MB | 0.777 | 0.938 | 25,097,128 | - |
| ResNeXt101 | 170 MB | 0.787 | 0.943 | 44,315,560 | - |
| InceptionV3 | 92 MB | 0.779 | 0.937 | 23,851,784 | 159 |
| InceptionResNetV2 | 215 MB | 0.803 | 0.953 | 55,873,736 | 572 |
| MobileNet | 16 MB | 0.704 | 0.895 | 4,253,864 | 88 |
| MobileNetV2 | 14 MB | 0.713 | 0.901 | 3,538,984 | 88 |
| DenseNet121 | 33 MB | 0.750 | 0.923 | 8,062,504 | 121 |
| DenseNet169 | 57 MB | 0.762 | 0.932 | 14,307,880 | 169 |
| DenseNet201 | 80 MB | 0.773 | 0.936 | 20,242,984 | 201 |
| NASNetMobile | 23 MB | 0.744 | 0.919 | 5,326,716 | - |
| NASNetLarge | 343 MB | 0.825 | 0.960 | 88,949,818 | - |

https://keras.io/applications/

# 참조. 딥러닝 작동 원리

## 케라스 모델 프로세스

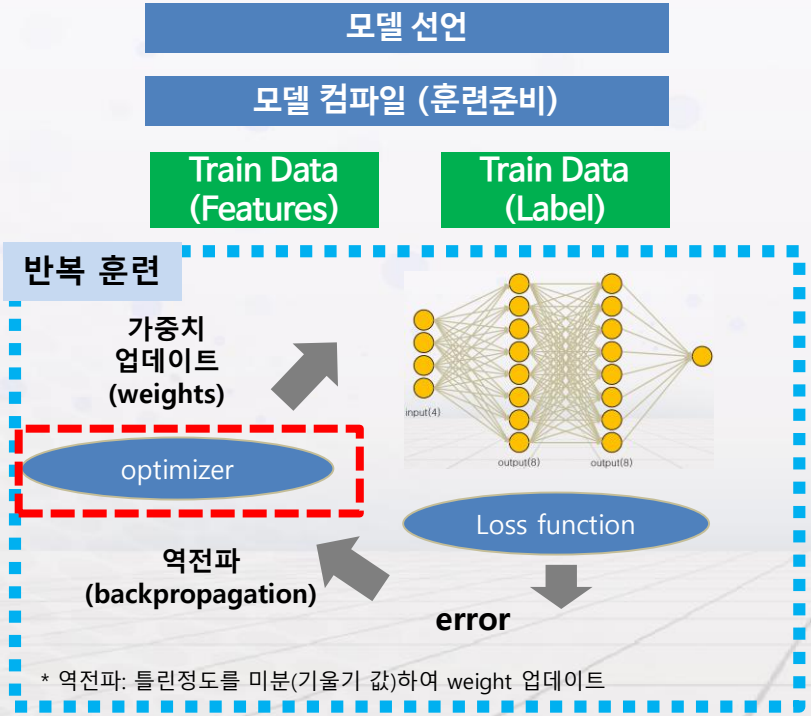



| Loss Function (손실함수) | 내용 |
|---|---|
| Mean_squared_error | 연속 숫자 예측 |
| Mean_absolute_error | |
| Mean_absolute_percentage_error | |
| Mean_squared_logarithmic_error | |
| Squared_hinge | |
| Hinge | |
| Categorical_hinge | |
| Logcosh | |
| Categorical_crossentropy | 멀티 카테고리 예측 |
| Sparse_categorical_crossentropy | |
| Binary_corssentropy | 2개 카테고리 예측 |

https://keras.io/losses/

# 참조. 딥러닝 작동 원리

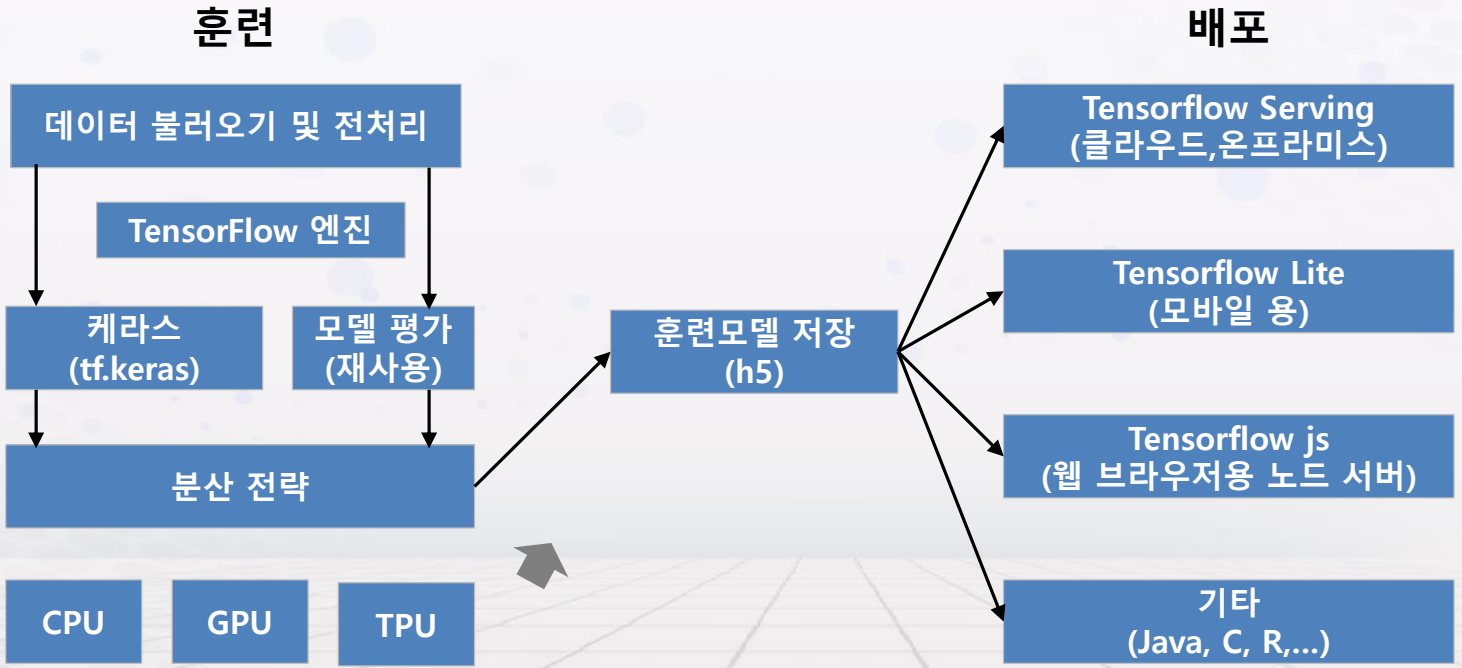
## 케라스 모델 프로세스




| Optimizer (최적화) | 비고 |
|---|---|
| GD (Gradient Descent) | 정확하지만 느리다 |
| SGD (Stochastic Gradient Descent) | 빠르지만 찾는 방향 뒤죽박죽 |
| RMSprop | 스탭줄일 시 이전 맥락 확인 |
| Adagrad | 안가본곳은 빠르게 가본곳 천천히 |
| Adadelta | 스탭너무 작아져서 정지 안되게 |
| Adam | RMSprop + Momentom<br>방향/스탭사이즈 적절하게 |

# 참조. 딥러닝 작동 원리

## 전체 프로세스




* TPU: Tensor Processing Unit

# 참조. 딥러닝 작동 원리

## 주요 수식

머신러닝은 특성을 선별하여 훈련 시켰다면
**딥러닝**은 **가능한 특성은 전부 포함** 시킨다.

$$Y = w * X + b$$


판매량


휴일정보


할인정보

# 참조. 딥러닝 작동 원리

## 주요 수식

실측 값 $Y = w * X + b$

예측 값 $\hat{Y} = w * \hat{X} + b$

# 참조. 딥러닝 작동 원리

## 주요 수식

실측 값

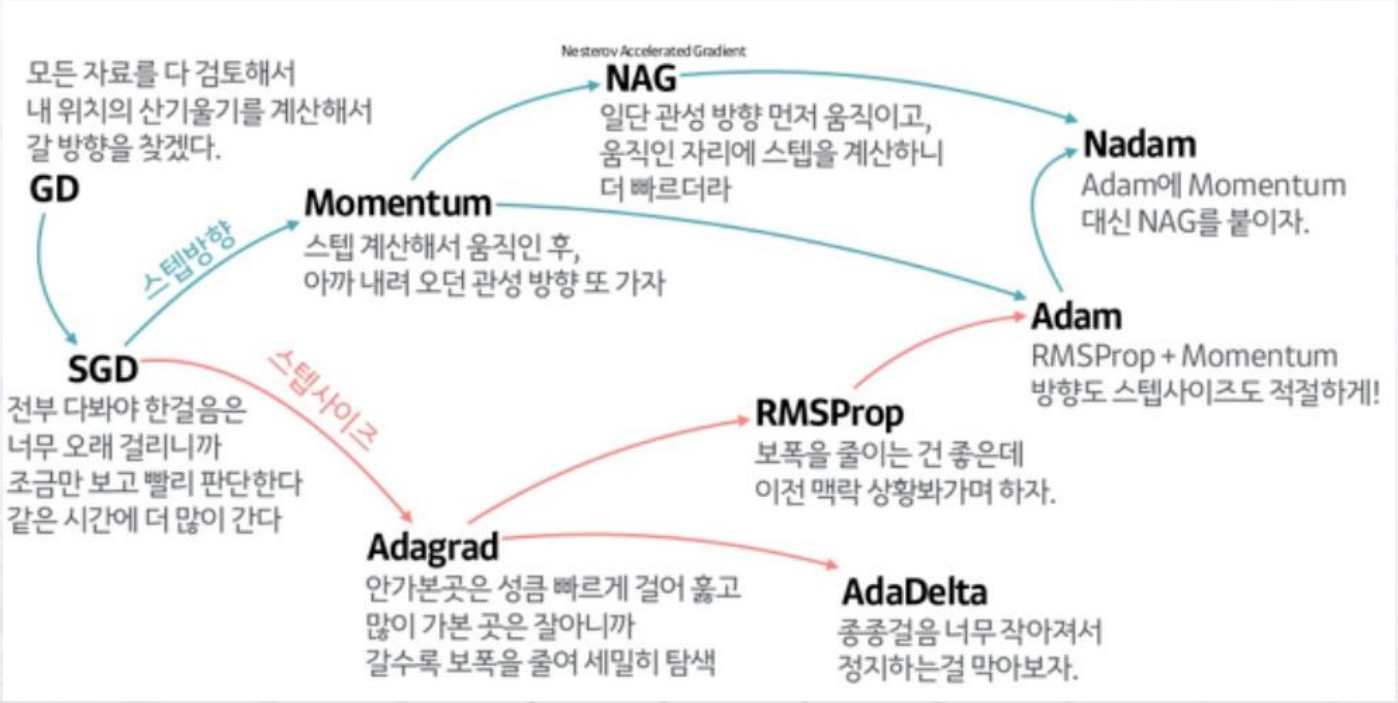
$$60 = w * 30\% + b$$

$$80 = w * 50\% + b$$

예측 값

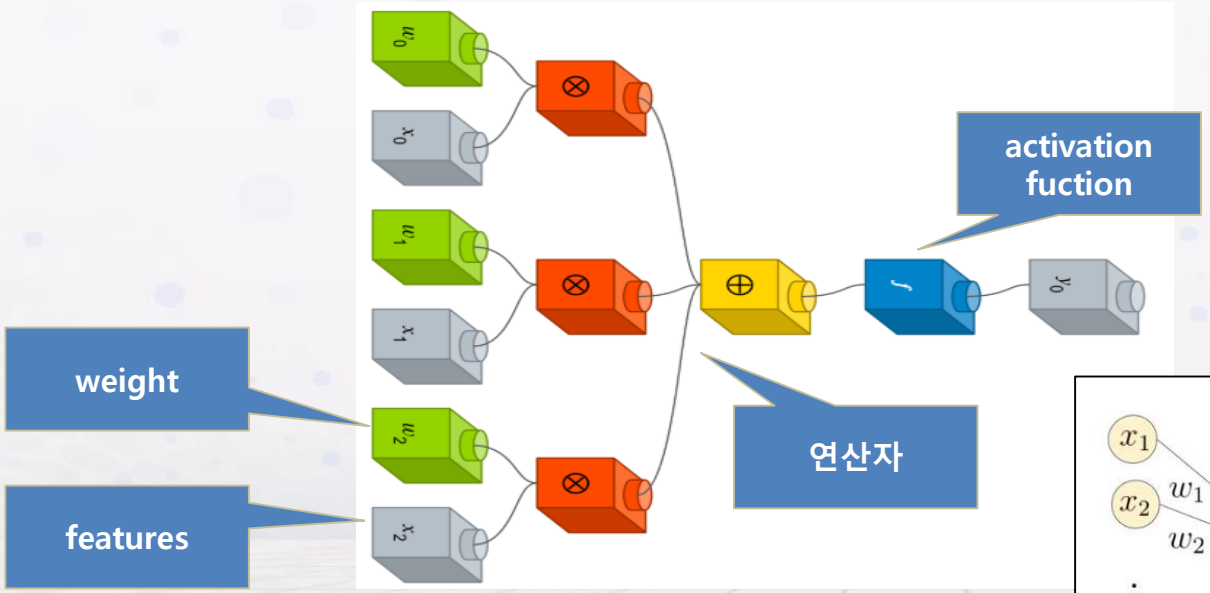
$$\hat{Y} = w * 40\% + b$$

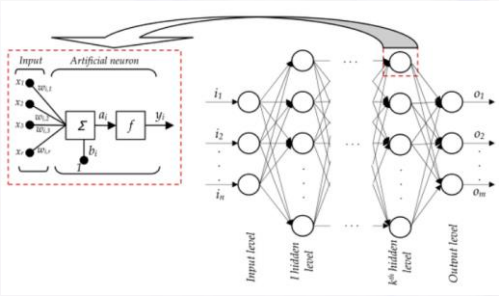
# 참조. 딥러닝 작동 원리

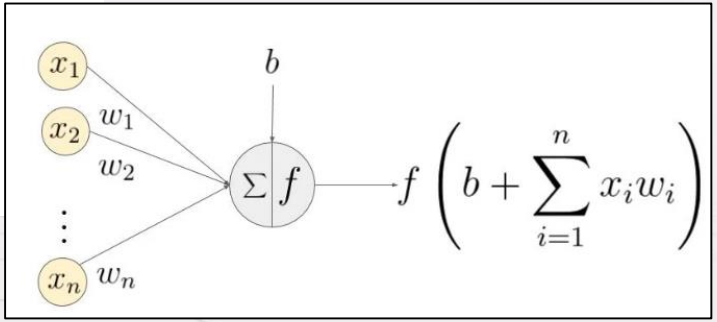

모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
스텝방향
SGD
전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다
Momentum
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자
Nesterov Accelerated Gradient
NAG
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라
Nadam
Adam에 Momentum
대신 NAG를 붙이자.
스텝사이즈
Adagrad
안가본곳은 성큼 빠르게 걸어 훑고
많이 가본 곳은 잘아니까
갈수록 보폭을 줄여 세밀히 탐색
RMSProp
보폭을 줄이는 건 좋은데
이전 맥락 상황봐가며 하자.
Adam
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!
AdaDelta
종종걸음 너무 작아져서
정지하는걸 막아보자.

# 참조. 딥러닝 작동 원리

## Neuron 상세 내용




https://tykimos.github.io/2017/01/27/MLP_Layer_Talk/

# 참조. 딥러닝 작동 원리

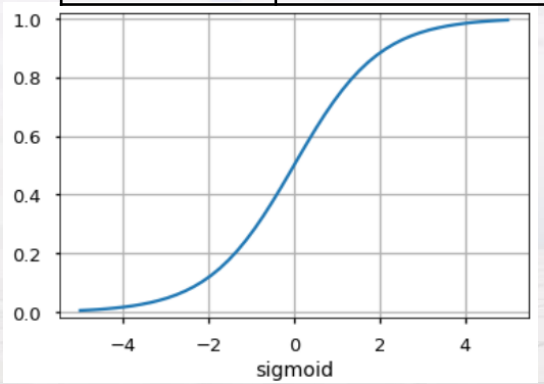
## activation 함수

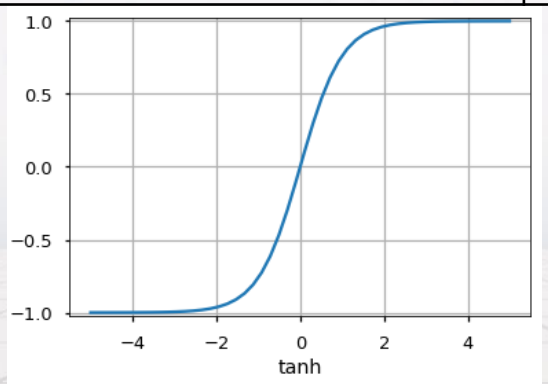
Activation functions on CIFAR-10

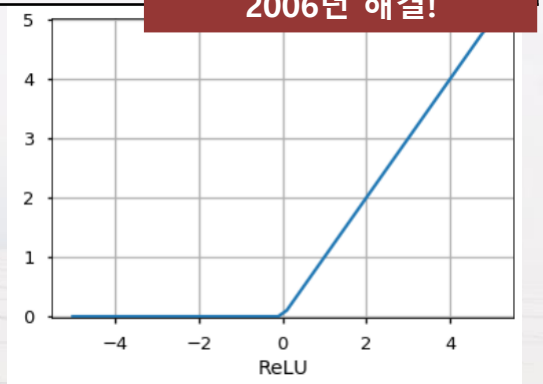
| maxout | ReLU | VLReLU | tanh | Sigmoid |
|---|---|---|---|---|
| **93.94** | **92.11** | 92.97 | 89.28 | n/c |
| 93.78 | 91.74 | 92.40 | 89.48 | n/c |
| – | 91.93 | **93.09** | – | n/c |
| 91.75 | 90.63 | 92.27 | **89.82** | n/c |
| n/c† | 90.91 | 92.43 | 89.54 | n/c |

| 함수종류 | 설명 | 비고 |
|---|---|---|
| linear | 디폴트, 입력뉴런과 가중치로 계산된 결과값 그대로 출력 | |
| sigmoid | 시그모이드함수, 이진분류 문제에서 출력층(output)에 주로 사용 | 0~1 사이 실수 값 출력 |
| relu | 음수에 대해 0으로 처리하는 함수, 기존의 sigmoid를 개선함 | max(0, x) |
| softmax | 소프트맥스, 다중 클래스분류문제 출력층(output)에 주로 사용 | 이미지 분류 시 유용 |
| tanh | sigmoid 함수를 재활용하기 위한 함수. sigmoid의 범위를 -1에서 1 | |

역전파 시 0~1 사이로 전파하여 손실 부분을 2006년 해결!



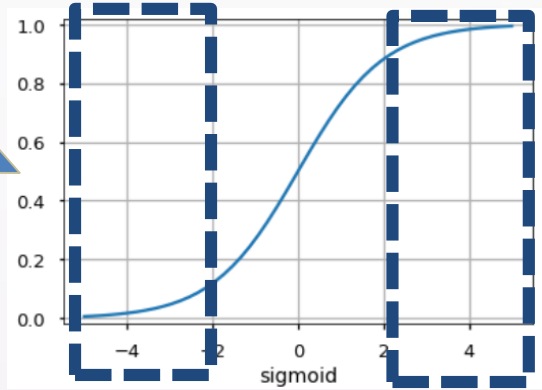



# 참조. 딥러닝 작동 원리

Sigmoid 활용 시
미분 값 0에 가까워져
역전파로 전달 시
Weight 업데이트 거의 안됨

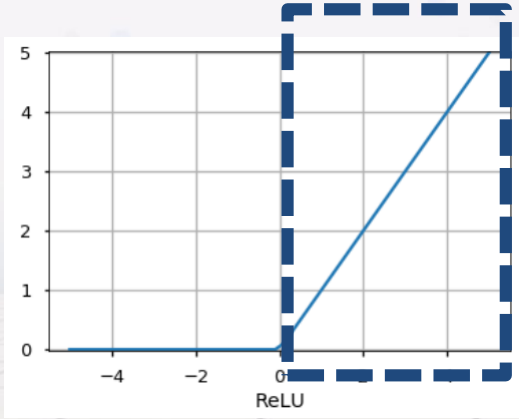
-> Vanishing Gradient
로 인해 학습이 안되는 현상을
Under fitting이라고 함




0~1 사이 값

Rectified Linear Unit

미분 값이 전부 0 이여서
역전파 시
Weight 업데이트 안되는
경우 없음!




F(x) = max(0, x)

https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

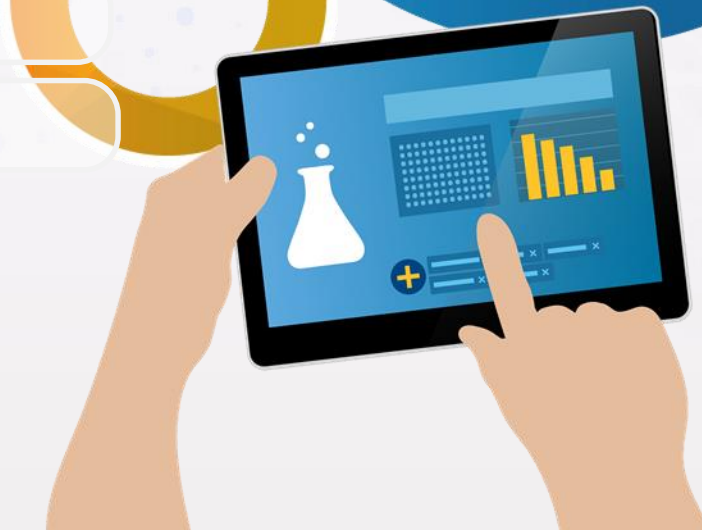
# 합성곱신경망(CNN)을 활용한 이미지 분류기 만들기

김 효 관

교육목표: 합성곱 신경망을 실전데이터를 활용하여 실습하고 응용할 수 있다.

# CONTENTS

1. 미니프로젝트 (리마인드 실습)
2. 미니프로젝트 (실전)
3. 핵심정리 및 Q&A



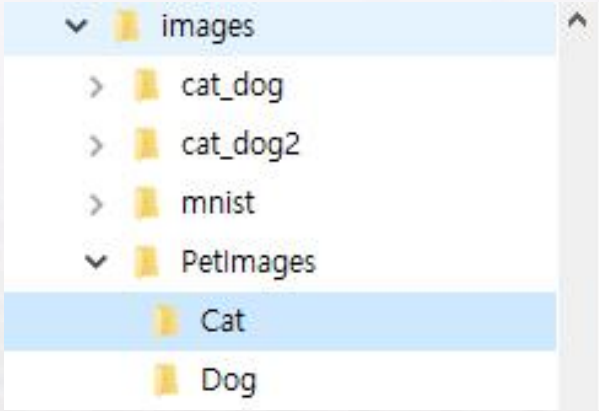
# 1. 미니프로젝트 (리마인드 실습)

## 데이터 준비

데이터를 다운받은 후 아래 폴더에 저장하세요.

https://www.microsoft.com/en-us/download/details.aspx?id=54765

or 제공된 PetImages 압축해제




# 1. 미니프로젝트 (리마인드 실습)

## 데이터 불러오기 전략

[훈련/테스트 세트] : 이미지/답지 별도 저장

폴더 리스트 반복

cat, dog

이미지 리스트 반복

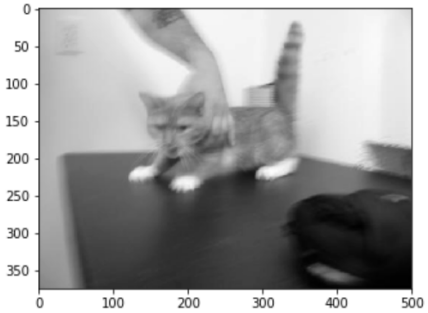
img_4.jpg, ..

데이터 저장

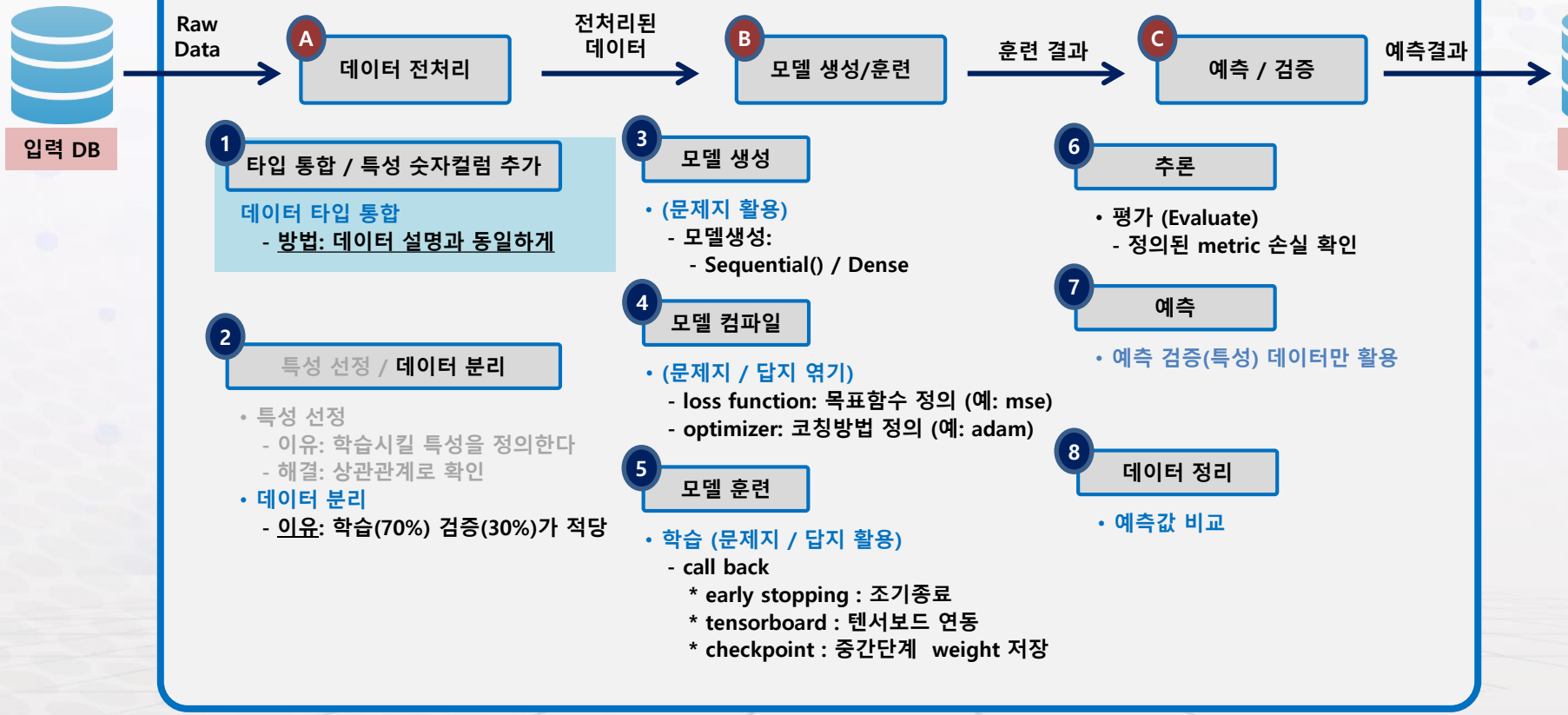
- 컬러: 그레이
- 사이즈: 50 * 50
- 이미지 저장
- 폴더 인덱스-> 답지

```
for category in categories:
    path = os.path.join(basedir, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
        plt.imshow(img_array, cmap="gray")
        plt.show()
        break
    break
```

```
for category in categories:
    path = os.path.join(basedir, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
        plt.imshow(img_array, cmap="gray")
        plt.show()
        break
    break
```



# 1. 미니프로젝트 (리마인드 실습)



# 1. 미니프로젝트 (리마인드 실습)

## 1. 데이터 불러오기

```
import numpy as np
import matplotlib.pyplot as plt
import os
# pip install opencv-python
import cv2
%matplotlib inline

basedir = "../images/PetImages/"

categories = os.listdir(basedir)
categories
```

합성곱 신경망 (Conv)를 위해
channel shape 추가 (depth)

```
x_train shape: (55000, 28, 28, 1) y_train shape: (55000, 10)
55000 train set
5000 validation set
10000 test set
```

# 1. 미니프로젝트 (리마인드 실습)

## 1. 데이터 불러오기

```
IMG_SIZE = 50
train_images=[]
train_labels=[]

def create_training_data():
    for category in categories:
        path = os.path.join(basedir, category)
        class_num = categories.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                # 이미지, 정답지 별도 리스트에 추가
                train_images.append(new_array)
                train_labels.append(category)
            except Exception as e:
                pass
create_training_data()
```

# 1. 미니프로젝트 (리마인드 실습)

## 2. 데이터 분리

```
from sklearn.model_selection import train_test_split

trainingData_features,₩
validData_features,₩
trainingData_labels,₩
validData_labels = ₩
train_test_split(train_images, train_labels, test_size = 0.2, random_state=2)

print(trainingData_features.shape)
print(validData_features.shape)
print(trainingData_labels.shape)
print(validData_labels.shape)
```

# 1. 미니프로젝트 (리마인드 실습)

## 2. 데이터 분리

feature는 정규화 후 형태를
(길이, 이미지사이즈, 이미지사이즈, 1) 로 변경
trainingData_features, validData_features
label은 onehot encoder활용하여
trainingData_labels_one, validData_labels_one 변수에
각각 저장하세요

# 1. 미니프로젝트 (리마인드 실습)

## 3. 모델 생성

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
data_shape = trainingData_features[0].shape

model = keras.Sequential()
# 신경망의 첫 번째 레이어에서 입력 데이터 크기를 정의해야 합니다.
model.add(Conv2D(filters=32, kernel_size=2,
                 padding='same',
                 activation='relu',
                 input_shape=data_shape))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(nclasses, activation='softmax'))
```

# 1. 미니프로젝트 (리마인드 실습)

## 4. 모델 컴파일

```
model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

# 1. 미니프로젝트 (리마인드 실습)

## 5. 모델 훈련

```
import os
from datetime import datetime
now = datetime.now()
logdt = now.strftime("%Y%m%d_%H%M%S")
logdate = os.path.join("log_catdog",logdt)
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, TensorBoard

callback_list = [
    EarlyStopping(monitor="val_loss",patience=10),
    ModelCheckpoint(filepath="./catdog_weight.h5",monitor="val_loss",save_best_only=True, verbose=1),
    TensorBoard(log_dir= logdate, write_graph=True, write_images=True)
]

model.fit(x=trainingData_features,
          y=trainingData_labels_one,
          validation_data = (validData_features, validData_labels_one),
          epochs=6,
          batch_size=32,
          callbacks=callback_list)
```

# 1. 미니프로젝트 (리마인드 실습)

## 6. 모델 추론

```
model.evaluate(x=validData_features,
               y=validData_labels_one)
```

**모델 추론**

```
1  model.evaluate(x=validData_features, y=validData_labels_one)
```

```
4990/4990 [==============================] - 2s 493us/step
[0.5343276406576734, 0.7348697185516357]
```

# 1. 미니프로젝트 (리마인드 실습)

## 7. 예측

```
testimage = "../images/PetImages/Cat/1.jpg"

inputImages = cv2.imread(testimage, cv2.IMREAD_GRAYSCALE)

inputImages = cv2.resize(inputImages, (IMG_SIZE,IMG_SIZE))

testdata_features = inputImages.reshape(1,IMG_SIZE, IMG_SIZE,1)

testdata_features = testdata_features/255.0

answer = model.predict(testdata_features)

print(answer)

np.array(np.argmax(answer))
```

# 1. 미니프로젝트 (리마인드 실습)

## 8. 모델 저장/재사용

### 모델 구조 저장

```
# serialize model structure to JSON
model_json = model.to_json()
with open("catdog_model.json", "w") as json_file:
    json_file.write(model_json)
model.save("catdog_model.h5")

model.save_weights("catdog_model_weight.h5")
```

### 저장된 모델 불러오기 및 예측

```
from tensorflow.keras.models import model_from_json

json_file = open("catdog_model.json","r")

loaded_model_json = json_file.read()

json_file.close()

loaded_model = model_from_json(loaded_model_json)

loaded_model.load_weights("./catdog_weight.h5")

loaded_model.predict(testdata_features)

print(answer)

np.array(np.argmax(answer))
```

# 1. 미니프로젝트 (리마인드 실습)

## 재학습 (참조)

모델 선언 구조 저장

```
# 모델 저장
model_json = model.to_json()

with open("model.json", "w") as json_file:
    json_file.write(model_json)

model.save_weights("linear_keras_sellout.h5")
```

역전파를 통한 업데이트된 가중치 저장

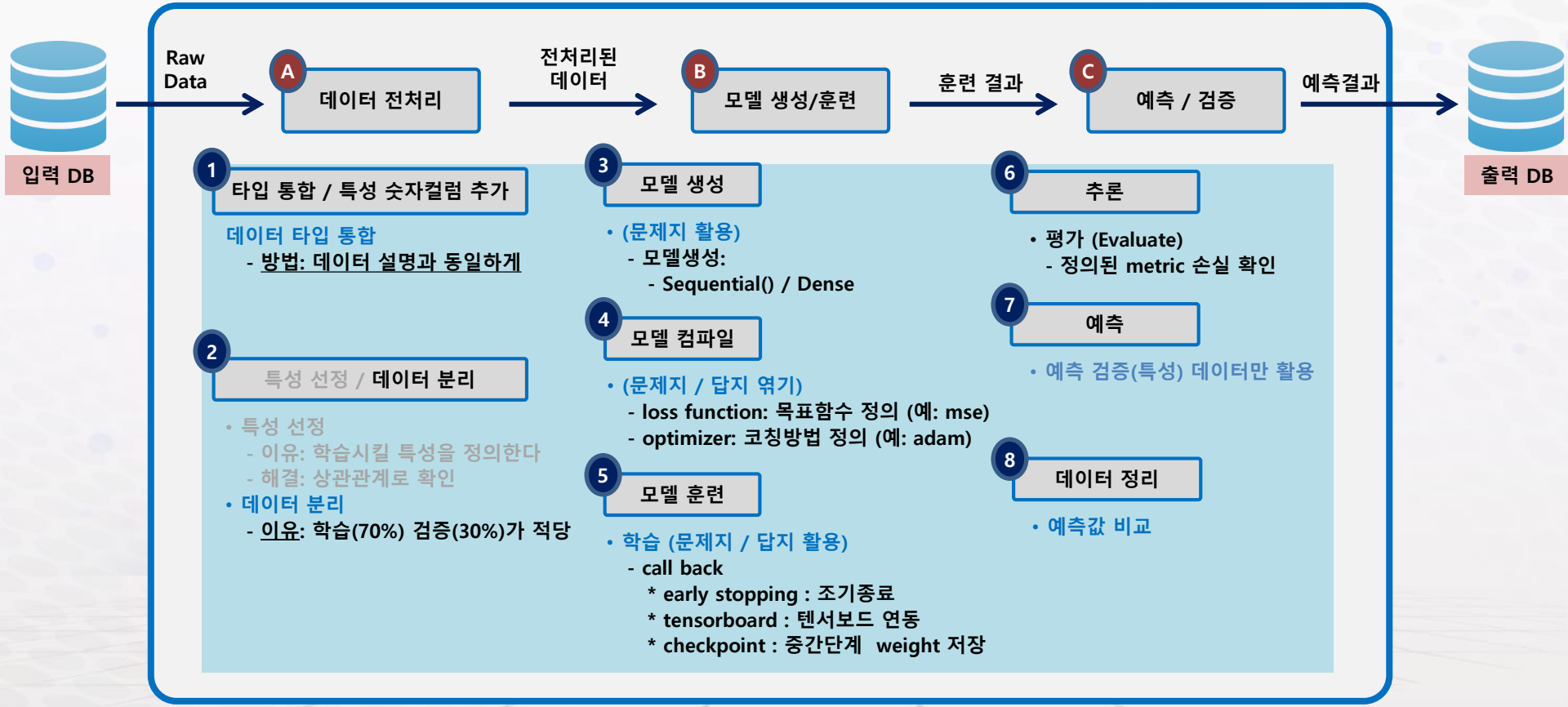
```
# 모델 불러오기
from tensorflow.keras.models import model_from_json
json_file = open("model.json", "r")
loaded_model_json = json_file.read()
json_file.close()

loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("catdog_weight.h5")
```

```
# 모델 재 컴파일
loaded_model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

loaded_model.summary()

# 모델 재 학습
from tensorflow.keras.callbacks import EarlyStopping
early_stopping_monitor = EarlyStopping(patience=50)
EPOCHS = 100
#모델 훈련 (훈련/검증을 80%, 20%로 나눔)
history =loaded_model.fit(trainingData_features,
                  trainingData_label,
                  validation_split=0.2, epochs= EPOCHS,
callbacks=[early_stopping_monitor])
```

# 2. 미니프로젝트 (실전)

아래의 자료를
활용하여 test 폴더에 있는 자료를
cat, dog로 분류한 후 각각
고양이 사진은 -> cat_dog/output/cat 폴더 내
강아지 사진은 -> cat_dog/output/dog 폴더 내
자동으로 분류해주는 모델을 생성하고 작성한 코드를
haiteam@kopo.ac.kr로 제출하세요.

# 2. 미니프로젝트 (실전)


입력 DB
Raw Data
A
데이터 전처리
전처리된 데이터
B
모델 생성/훈련
훈련 결과
C
예측 / 검증
예측결과
출력 DB
1
타입 통합 / 특성 숫자컬럼 추가
데이터 타입 통합
- 방법: 데이터 설명과 동일하게
2
특성 선정 / 데이터 분리
• 특성 선정
- 이유: 학습시킬 특성을 정의한다
- 해결: 상관관계로 확인
• 데이터 분리
- 이유: 학습(70%) 검증(30%)가 적당
3
모델 생성
• (문제지 활용)
- 모델생성:
- Sequential() / Dense
4
모델 컴파일
• (문제지 / 답지 엮기)
- loss function: 목표함수 정의 (예: mse)
- optimizer: 코칭방법 정의 (예: adam)
5
모델 훈련
• 학습 (문제지 / 답지 활용)
- call back
* early stopping : 조기종료
* tensorboard : 텐서보드 연동
* checkpoint : 중간단계 weight 저장
6
추론
• 평가 (Evaluate)
- 정의된 metric 손실 확인
7
예측
• 예측 검증(특성) 데이터만 활용
8
데이터 정리
• 예측값 비교

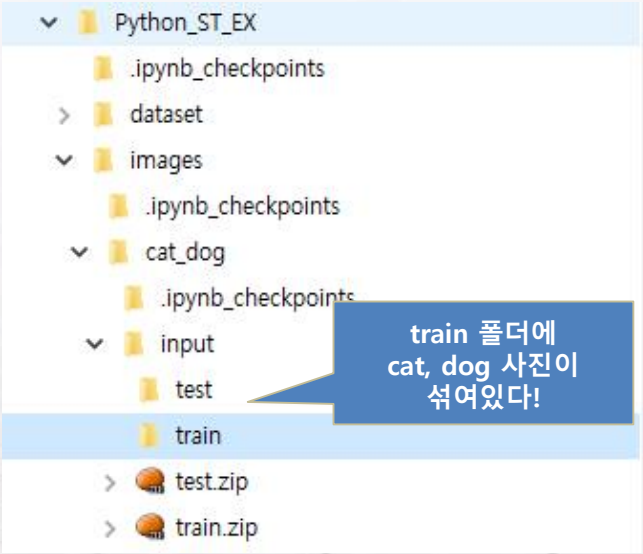
# 2. 미니프로젝트 (실전)

## 데이터 준비

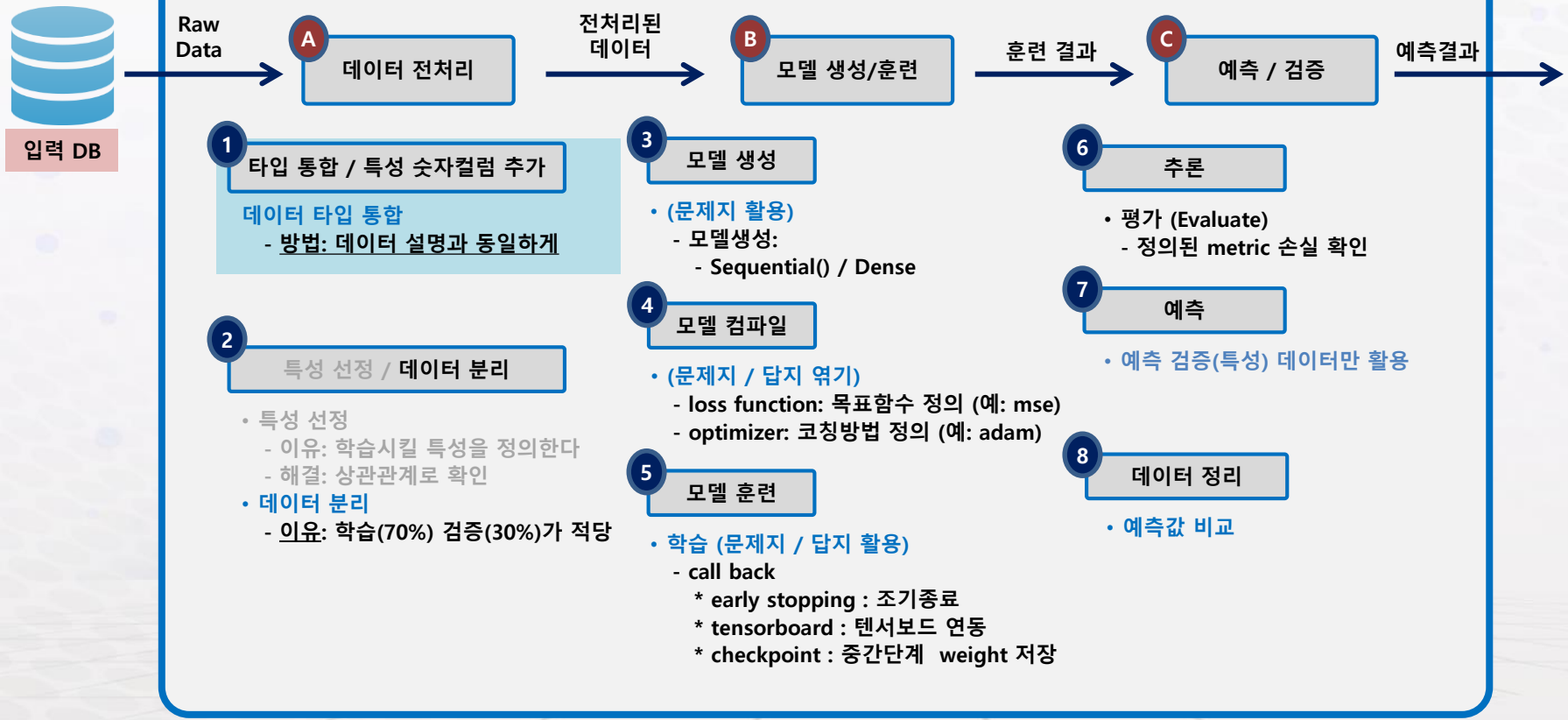
데이터를 다운받은 후 아래 폴더에 저장하세요.

https://www.kaggle.com/c/dogs-vs-cats/data

or 제공된 cat_dog 압축파일 해제



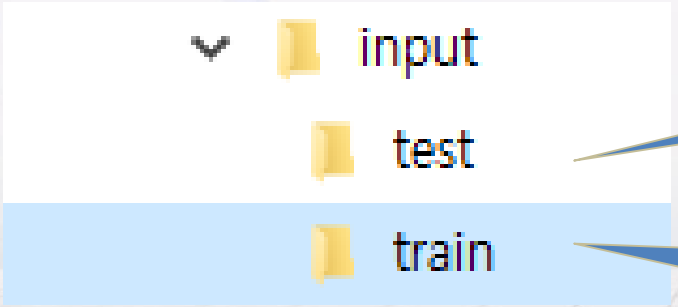

# 2. 미니프로젝트 (실전)


입력 DB
Raw Data
A
데이터 전처리
전처리된 데이터
B
모델 생성/훈련
훈련 결과
C
예측 / 검증
예측결과
출력 DB
1
타입 통합 / 특성 숫자컬럼 추가
데이터 타입 통합
- 방법: 데이터 설명과 동일하게
2
특성 선정 / 데이터 분리
• 특성 선정
- 이유: 학습시킬 특성을 정의한다
- 해결: 상관관계로 확인
• 데이터 분리
- 이유: 학습(70%) 검증(30%)가 적당
3
모델 생성
• (문제지 활용)
- 모델생성:
- Sequential() / Dense
4
모델 컴파일
• (문제지 / 답지 엮기)
- loss function: 목표함수 정의 (예: mse)
- optimizer: 코칭방법 정의 (예: adam)
5
모델 훈련
• 학습 (문제지 / 답지 활용)
- call back
* early stopping : 조기종료
* tensorboard : 텐서보드 연동
* checkpoint : 중간단계 weight 저장
6
추론
• 평가 (Evaluate)
- 정의된 metric 손실 확인
7
예측
• 예측 검증(특성) 데이터만 활용
8
데이터 정리
• 예측값 비교

# 2. 미니프로젝트 (실전)

## 데이터 준비

각 폴더별 데이터 구성



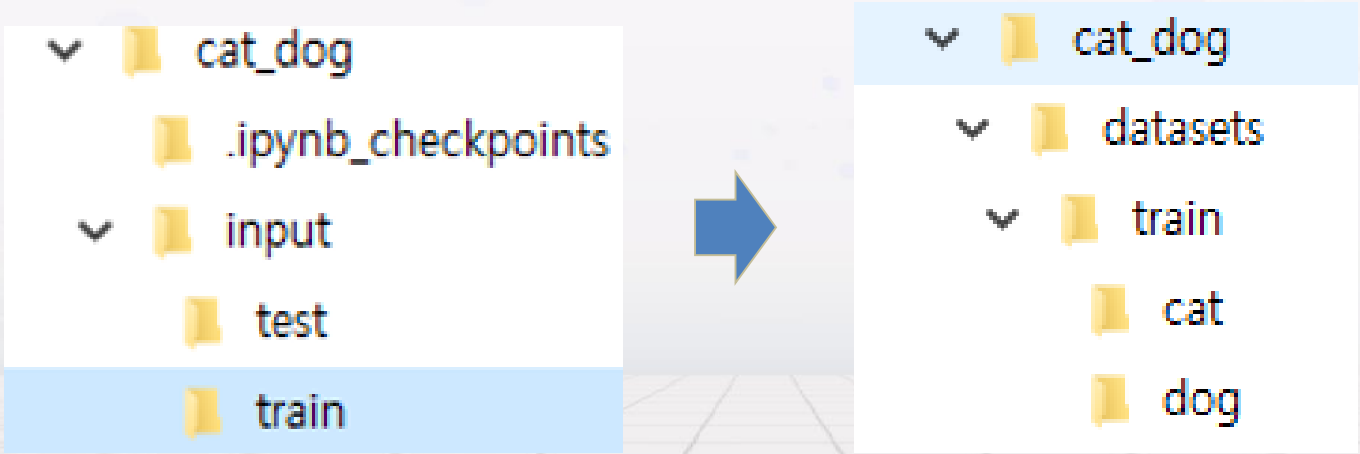

12500장의 검증 데이터
1.jpg ~ 12499.jpg

고양이 사진 12500장/
강아지 사진 12500장

# 2. 미니프로젝트 (실전)

## 데이터 분리 전략

폴더별로 분리하자!



# 2. 미니프로젝트 (실전)

## 라이브러리 활용 (shutils)

섞여있는 파일 구조를 폴더별로(cat, dog) 정리한다!!

```
import os, shutil

org_test_path='../images/cat_dog/input/test'
copy_test_path = '../images/cat_dog/output/'
```

원 본테스트 파일 위치
(cat, dog 섞여 있음)

해당 폴더 내 cat, dog 폴더
예측한 결과를 활용하여
데이터 폴더 분리 및 저장

# 3. 합성곱 신경망 실습

## 1. 훈련/검증데이터 구성

섞여있는 파일 구조를 폴더별로(cat, dog) 정리한다!!

```
import os, shutil

org_train_path='../images/cat_dog/input/train'
copy_train_path = '../images/cat_dog/datasets'
```

원본 파일 위치
(cat, dog 섞여 있음)

해당 폴더 내 train 이하 cat, dog 폴더
생성 후 데이터 분리

# 3. 합성곱 신경망 실습

## 1. 훈련/검증데이터 구성

섞여있는 파일 구조를 폴더별로(cat, dog) 정리한다!!

```
def copy_files(category_path, start_num, end_num, train_val_path):
    ## 이동 대상 이미지 파일경로 리스트에 담기
    image_paths=[]
    for i in range(start_num, end_num):
        image_paths.append(os.path.join(org_train_path,category_path+'.'+str(i)+'.jpg'))
    ## 이동할 폴더 경로 담기
    target_copy_paths=os.path.join(copy_train_path, train_val_path,category_path)

    # 신규 디렉토리가 존재하지 않으면 생성
    if not os.path.isdir(target_copy_paths):
        os.makedirs(target_copy_paths)

    for image_path in image_paths:
        ## shutil(원본경로, 이동할경로)
        shutil.copy(image_path, target_copy_paths)

    print('copy job completed')
```

os.path.join(c:/a',’b',’c')
-> c:/a/b/c

```
copy_files("dog",0,10000,"train")
copy_files("cat",0,10000,"train")
```

```
copy_files("dog",10000,12500,"validation")
copy_files("cat",10000,12500,"validation")
```

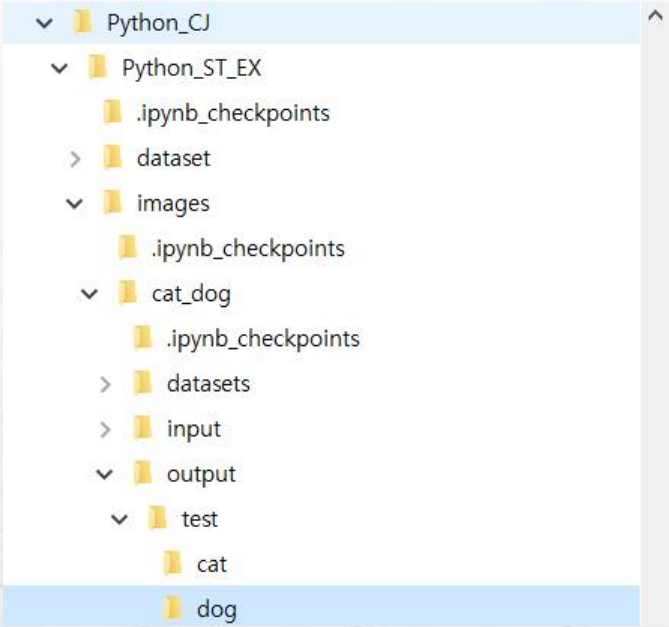
# 2. 미니프로젝트 (실전)

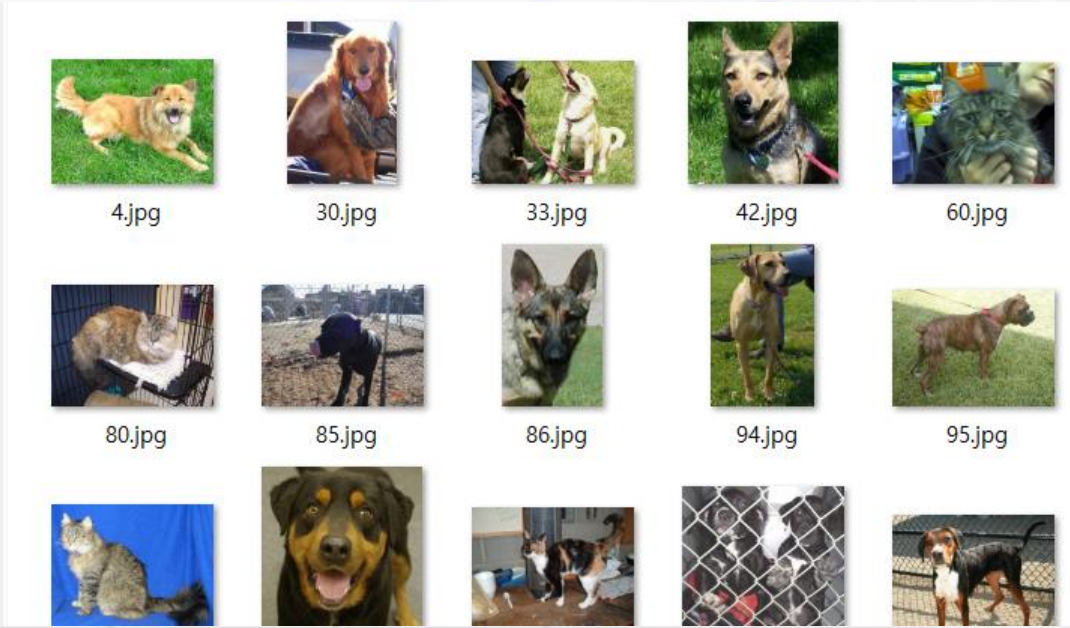
아래의 자료를
활용하여 test 폴더에 있는 자료를
cat, dog로 분류한 후 각각
고양이 사진은 -> cat_dog/output/cat 폴더 내
강아지 사진은 -> cat_dog/output/dog 폴더 내
자동으로 분류해주는 모델을 생성하고 작성한 코드를
haiteam@kopo.ac.kr로 제출하세요.

# 2. 미니프로젝트 (실전)

## 결과!

test 데이터를 폴더별로 정리한다.



# 3. 핵심정리 및 Q&A

## 기억합시다

1. 합성곱 신경망 실습을 통해 작동원리를 이해한다.
2. 실제 응용영역에서 합성곱신경망을 활용하는 방법을 이해한다.

The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves $k$ filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves $k$ filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

감사합니다.