

Perceptron 실습

Homework #1

PERCEPTRON

```
# importing Python library
import numpy as np

# define Unit Step Function
def sigmoid(x):
    return 1/(1+np.exp(x))
```

```
def numerical_derivative(f, x):
    dx = 1e-4
    gradf = np.zeros_like(x)

    it = np.nditer(x, flags = ['multi_index'],
                    op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float((tmp_val)+dx)
        fx1 = f(x)

        x[idx] = float((tmp_val)-dx)
        fx2 = f(x)
        gradf[idx] = (fx1-fx2)/(2*dx)

        x[idx] = tmp_val
        it.iternext()
    return gradf
```

PERCEPTRON: logicGate (1)

```
class logicGate:
    def __init__(self, gate_name, xdata, tdata, learning_rate=0.01, threshold=0.5):
        self.name = gate_name

        self.__xdata=xdata.reshape(4,2)
        self.__tdata=tdata.reshape(4,1)

        self.__w=np.random.rand(2,1)
        self.__b=np.random.rand(1)

        self.__learning_rate = learning_rate
        self.__threshold = threshold

    def __loss_func(self):
        delta = 1e-7

        z = np.dot(self.__xdata, self.__w) + self.__b
        y = sigmoid(z)

        return -np.sum(self.__tdata*np.log(y+delta) + (1-self.__tdata)*np.log((1-y)+delta))
```

PERCEPTRON : logicGate (2)

```
def err_val(self):
    delta = 1e-7

    z = np.dot(self.__xdata, self.__w)+self.__b
    y = sigmoid(z)
    return -np.sum(self.__tdata + np.log(y+delta) + (1-self.__tdata)*np.log((1-y)+delta))

def train(self):
    f = lambda x : self.__loss_func()
    print("init error : ", self.err_val())

    for stp in range(20000):
        self.__w -= self.__learning_rate * numerical_derivative(f, self.__w)
        self.__b -= self.__learning_rate * numerical_derivative(f, self.__b)
        if (stp % 2000 == 0):
            print("step : ", stp, "error : ", self.err_val(), f)
```

PERCEPTRON : logicGate (3)

```
def predict(self, input_data):  
    z = np.dot(input_data, self.__w) + self.__b  
    y = sigmoid(z)  
  
    if y[0] > self.__threshold:  
        result = 1  
    else:  
        result = 0  
    return y, result
```

Logic gate (AND, OR)

```
xdata = np.array([[0,0],[0,1],[1,0],[1,1]])
tdata = np.array([[0,0,0,1]])

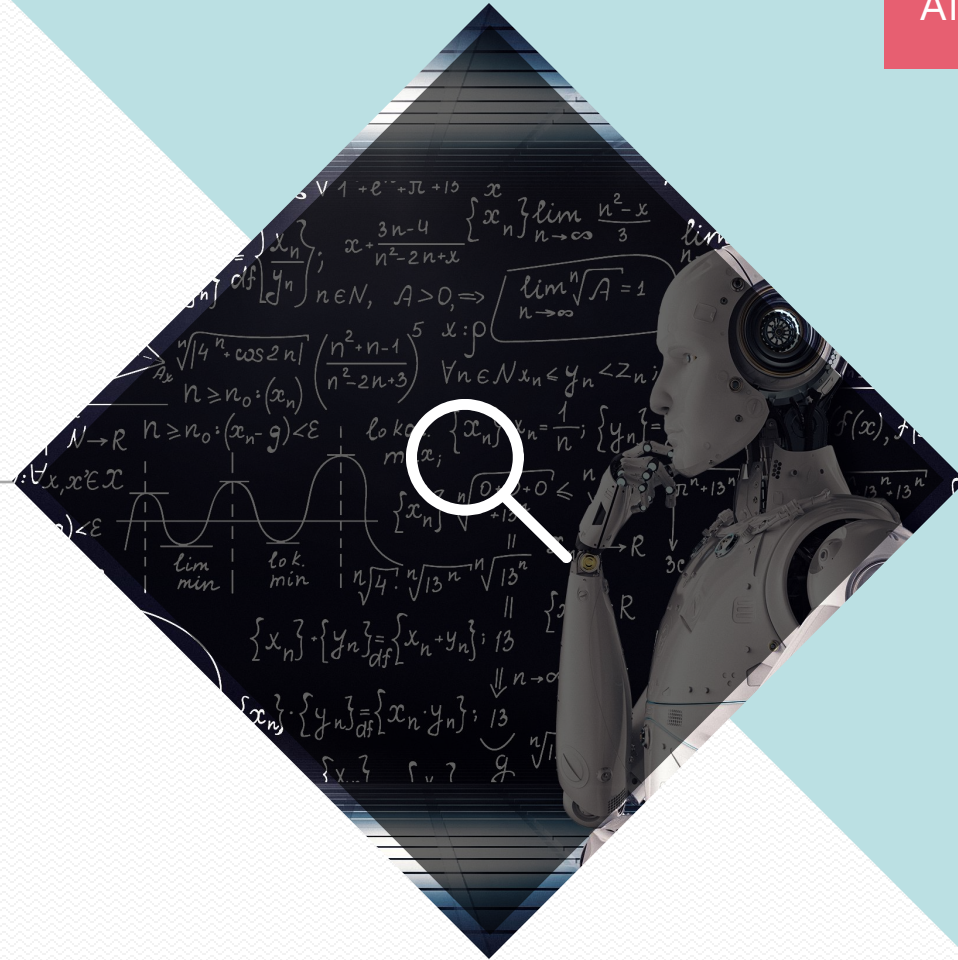
AND = logicGate("AND", xdata, tdata)
AND.train()
for in_data in xdata:
    (sig_val, logic_val) = AND.predict(in_data)
    print(in_data , " : ", logic_val)
```

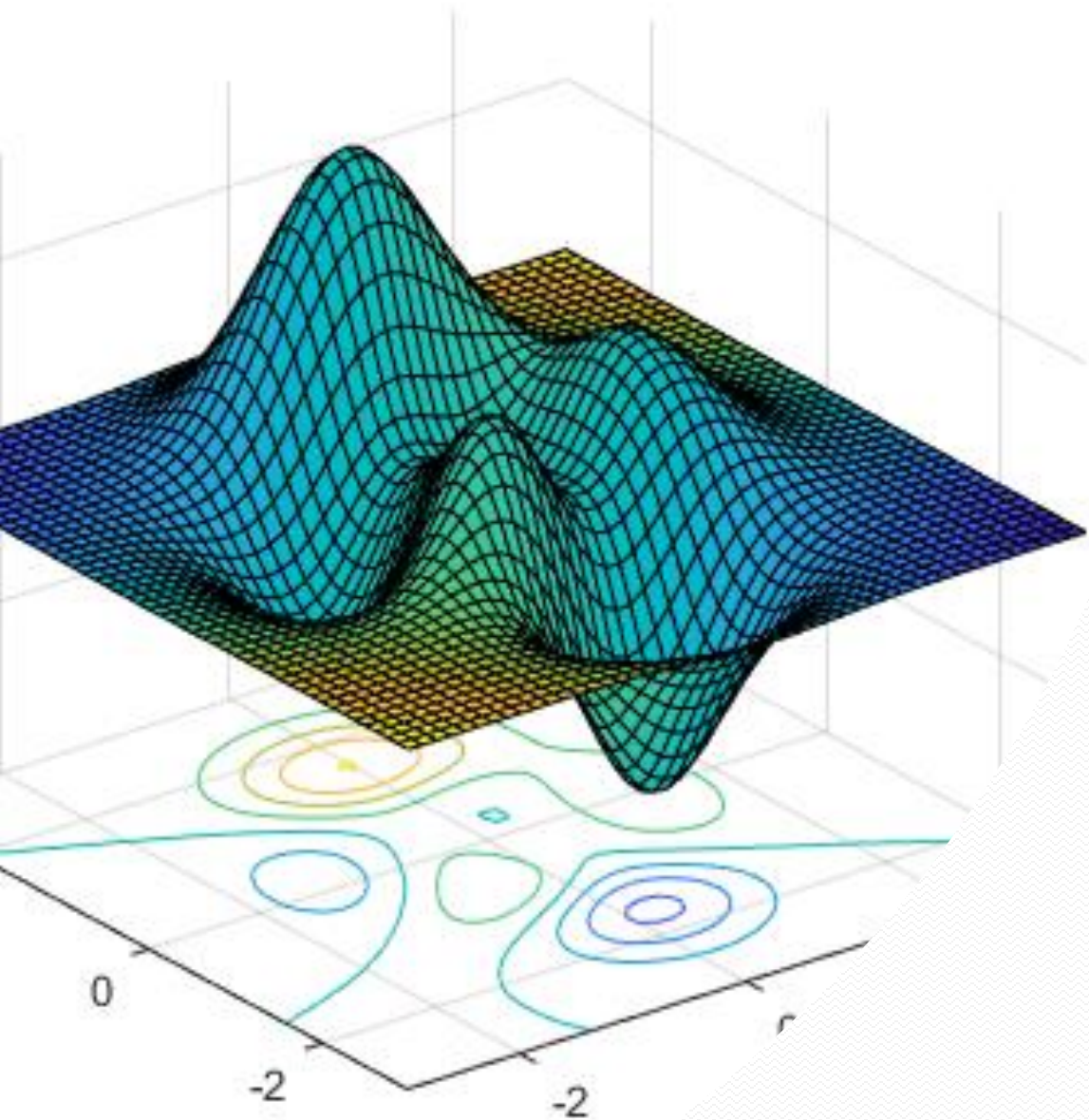
```
xdata = np.array([[0,0],[0,1],[1,0],[1,1]])
tdata = np.array([[0,1,1,1]])

OR = logicGate("OR", xdata, tdata)
OR.train()
for in_data in xdata:
    (sig_val, logic_val) = OR.predict(in_data)
    print(in_data , " : ", logic_val)
```


ANN 실습

Mnist & fashion mnist





ANN with Mnist 실습

Homework #2

VIRTUAL ENVIRONMENT

- Python3 virtual environment.
 - `$mkdir intro && cd intro`
 - `$sudo apt install python3-venv`
 - `$python3 -m venv .venv`
 - `$source .venv/bin/activate`
- Download & install python modules.
 - `(.venv)$ pip install tensorflow`
 - `(.venv)$ pip install numpy`
 - `(.venv)$ pip install matplotlib`
 - `(.venv)$ pip install PyQt5==5.14`

INITIAL SETTING & IMPORT MODULES

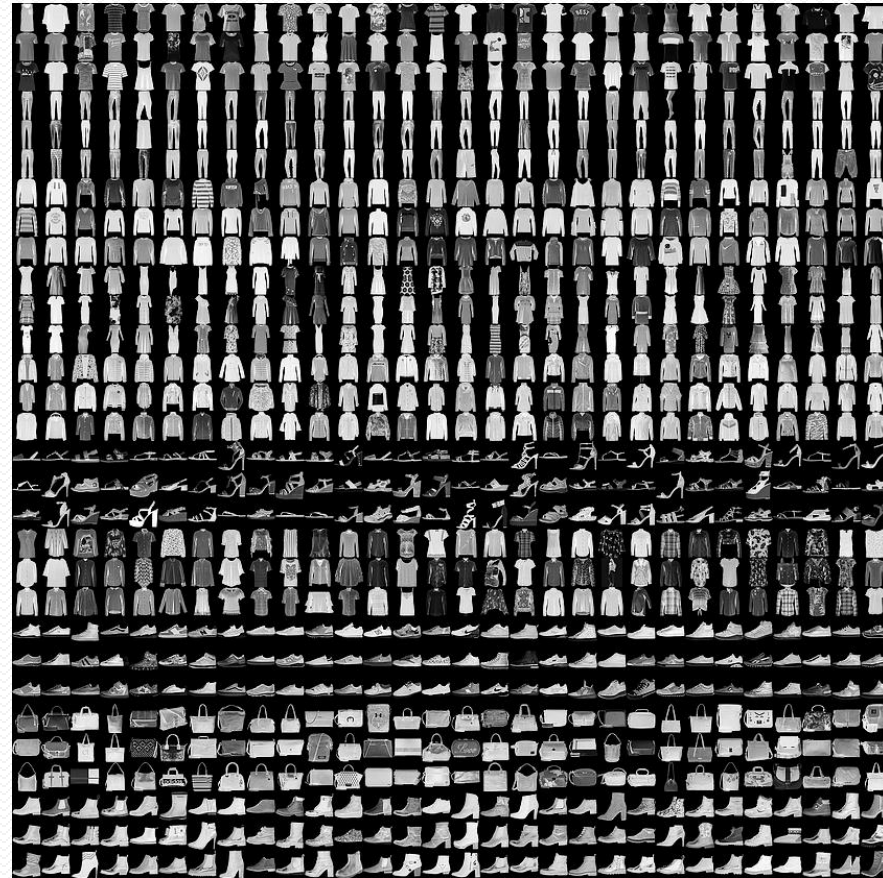
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```


DATASET PREPARATION (Hand and Fashion)

Model load: MNIST / Fashion MNIST Dataset

```
mnist = tf.keras.datasets.mnist
```

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```



DATASET PREPARATION

mnist 혹은 fashion_mnist 데이터셋을 준비하고, training할 이미지셋과 test할 이미지 셋을 구분해준다.
그 후 이미지의 색상을 정규화를 시켜주기 위해 255로 색상 값을 나눈다.

```
# Model load: MNIST / Fashion MNIST Dataset
```

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

```
# or
```

```
mnist = tf.keras.datasets.mnist
```

```
(f_image_train, f_label_train), (f_image_test, f_label_test) = fashion_mnist.load_data()
```

```
# normalized iamges
```

```
f_image_train, f_image_test = f_image_train / 255.0, f_image_test / 255.0
```


DATASET PREPARATION (cont'd)

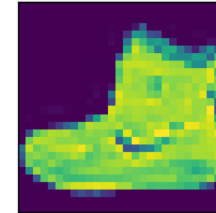
fashion_mnist 의 레이블은 숫자로 저장되어 있기 때문에 레이블과 클래스 이름을 매핑을 해줘야 한다.

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',  
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

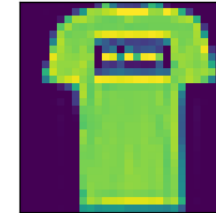
레이블	클래스
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

DATASET PREPARATION (show)

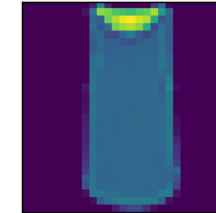
```
plt.figure(figsize=(10,10))
for i in range(10):
    plt.subplot(3,4,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(f_image_train[i])
    plt.xlabel(class_names[f_label_train[i]])
plt.show()
```



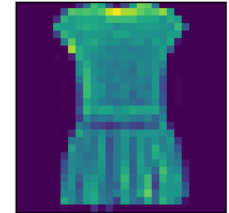
Ankle boot



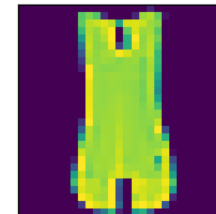
T-shirt/top



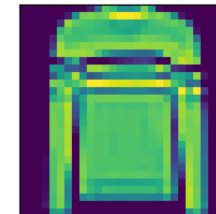
T-shirt/top



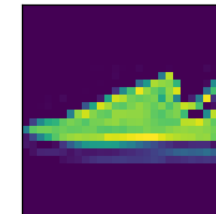
Dress



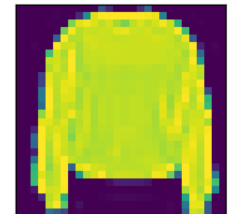
T-shirt/top



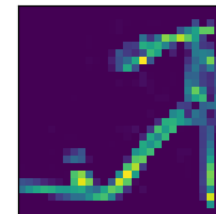
Pullover



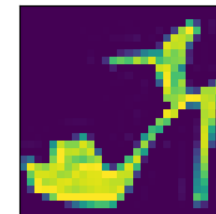
Sneaker



Pullover



Sandal



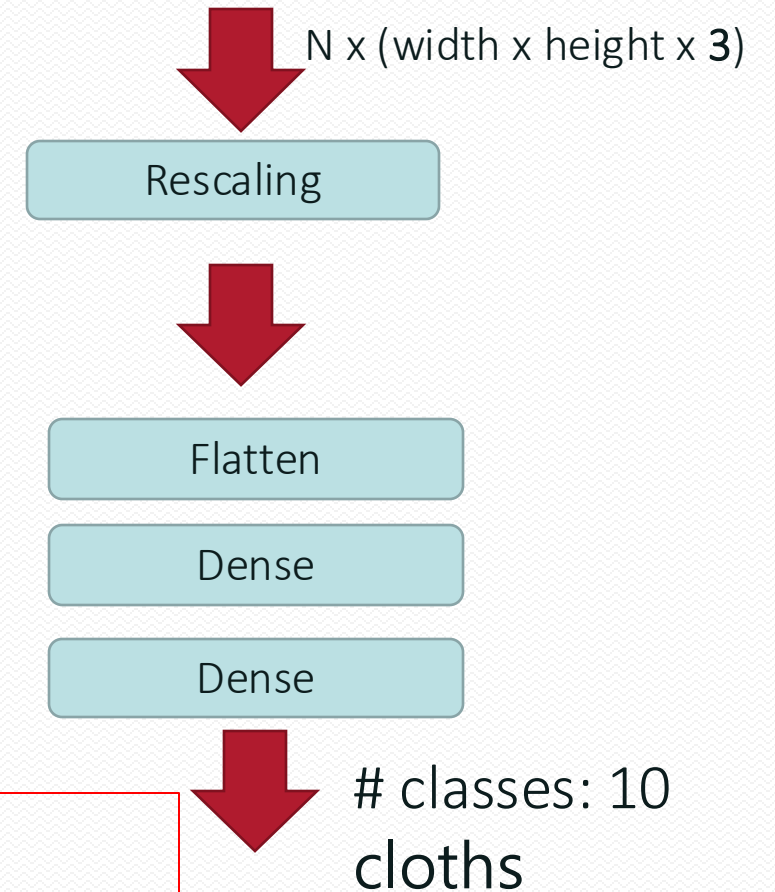
Sandal

MODEL CONSTRUCTION

https://www.tensorflow.org/api_docs/python/tf

- [`tf.keras.Sequential\(\)`](#)
- [`tf.keras.layers`](#)
 - Pooling
 - Flatten
 - Dense(ANN/FC)

```
# ANN
model = Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```



MODEL COMPILATION

- [tf.keras.Sequential.compile\(\)](#)
 - optimizer
 - loss
 - Metrics
- [tf.model.fit\(\)](#)
 - validation_data
 - epochs
 - Hyper parameters
 - Epochs
 - Dataset size
 - Batch size
 - Optimizer / loss function
- Save model
 - [tf.keras.Model.save\(\)](#)

```
model.compile(  
    optimizer='adam',  
    loss='sparse_categorical_crossentropy',  
    metrics=['accuracy'],  
)  
model.fit(image_train, label_train, epochs=10, batch_size=10)  
model.summary()  
model.save('fashion_mnist.h5')
```

INFERENCE TRAINED MODELS

- Load model
 - [tf.keras.models.load_model\(\)](#)

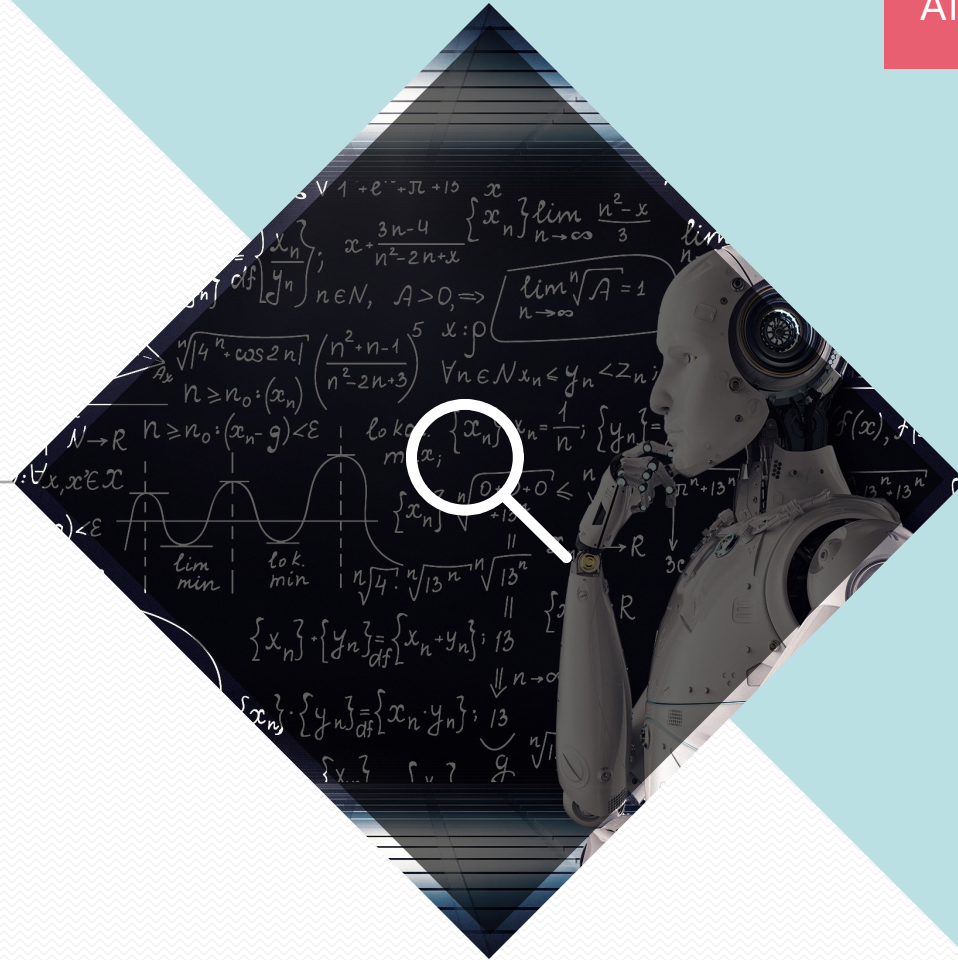
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

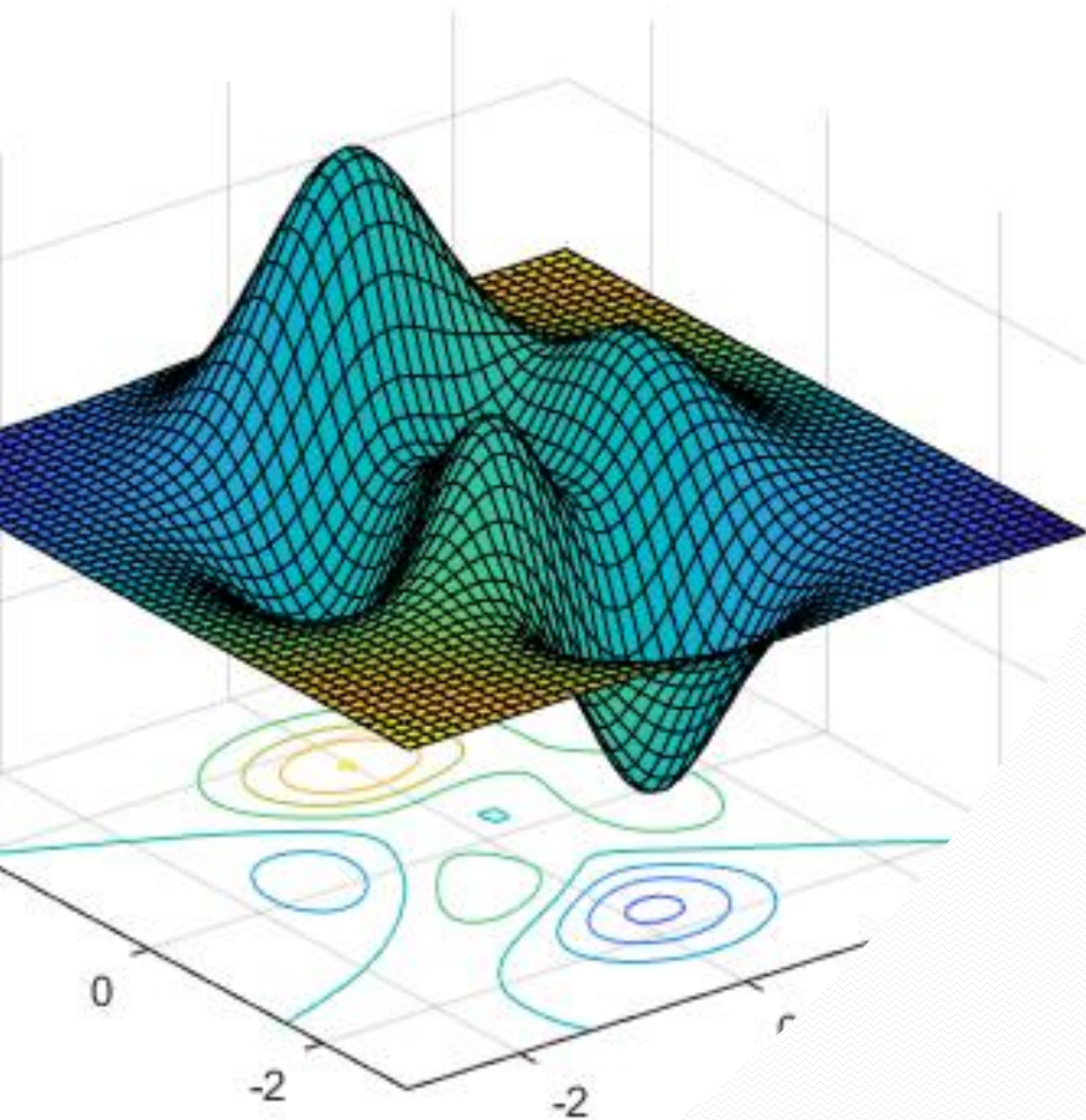
```
model = tf.keras.models.load_model('./fashion_mnist.h5')
fashion_mnist = tf.keras.datasets.fashion_mnist
(f_image_train, f_label_train), (f_image_test, f_label_test) =
fashion_mnist.load_data()
```

```
f_image_train, f_image_test = f_image_train / 255.0, f_image_test / 255.0
```

```
num = 10
predict = model.predict(f_image_test[:num])
print(f_label_test[:num])
print(" * Prediction, ", np.argmax(predict, axis = 1))
```

Medical images classification



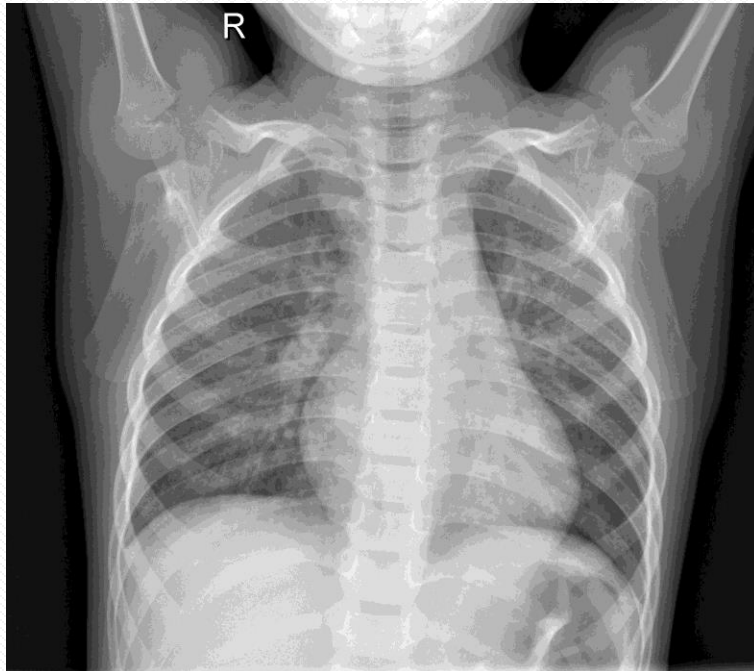


Medical images classification

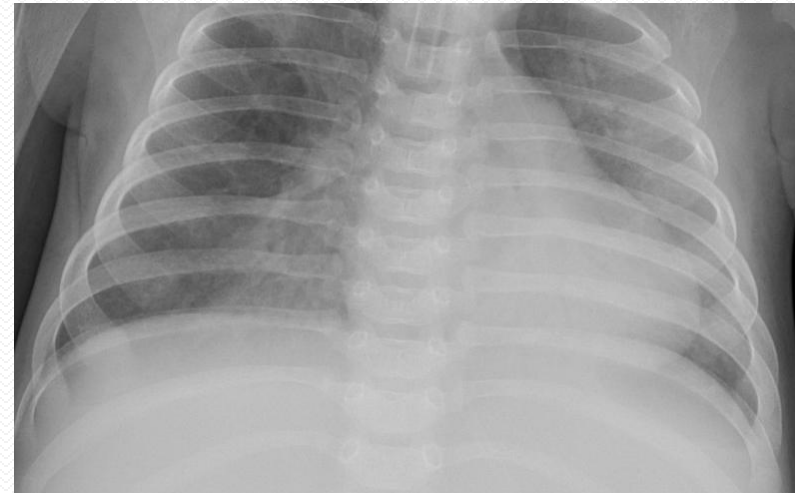
MEDICAL IMAGE CLASSIFICATION: DATASET PREPARATION

<https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia>

Normal



Pneumonia



MEDICAL IMAGE CLASSIFICATION: IMPORT

```
import numpy as np # for linear algebra
import matplotlib.pyplot as plt # for plotting things
import os
from PIL import Image # for reading images

# Keras Libraries <- CNN
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, Model, Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, AveragePooling2D,
Flatten, Dense, Input, BatchNormalization, Concatenate, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
# from sklearn.metrics import classification_report, confusion_matrix # <- define
# evaluation metrics
```

MEDICAL IMAGE CLASSIFICATION: DATASET PATH (1)

학습할 이미지들의 파일경로를 가져오기

```
mainDIR = os.listdir('./chest_xray')
print(mainDIR)
train_folder= './chest_xray/train/'
val_folder = './chest_xray/val/'
test_folder = './chest_xray/test/'
# train
os.listdir(train_folder)
train_n = train_folder+'NORMAL/'
train_p = train_folder+'PNEUMONIA/'
#Normal pic
print(len(os.listdir(train_n)))
rand_norm= np.random.randint(0,len(os.listdir(train_n)))
norm_pic = os.listdir(train_n)[rand_norm]
print('normal picture title: ',norm_pic)
norm_pic_address = train_n+norm_pic
#Pneumonia
rand_p = np.random.randint(0,len(os.listdir(train_p)))
sic_pic = os.listdir(train_p)[rand_norm]
sic_address = train_p+sic_pic
print('pneumonia picture title:', sic_pic)
```

MEDICAL IMAGE CLASSIFICATION: DATASET SHOW (2)

학습할 이미지를 불러오고 **PLOT**

```
# Load the images
norm_load = Image.open(norm_pic_address)
sic_load = Image.open(sic_address)


#Let's plt these images
f = plt.figure(figsize= (10,6))
a1 = f.add_subplot(1,2,1)
img_plot = plt.imshow(norm_load)
a1.set_title('Normal')

a2 = f.add_subplot(1, 2, 2)
img_plot = plt.imshow(sic_load)
a2.set_title('Pneumonia')
plt.show()
# let's build the CNN model
```

MEDICAL IMAGE CLASSIFICATION: BUILD MODEL

model.summary()

Homework #3



MEDICAL IMAGE CLASSIFICATION: DATASET PREPARATION (cont'd)

```
validation_generator = test_datagen.flow_from_directory('./chest_xray/val/',  
    target_size=(64, 64),  
    batch_size=32,  
    class_mode='binary')
```

```
test_set = test_datagen.flow_from_directory('./chest_xray/test',  
    target_size = (64, 64),  
    batch_size = 32,  
    class_mode = 'binary')
```

```
model_fin.summary()
```

Layer (type)	Output Shape	Param #
=====		
conv2d_3 (Conv2D)	(None, 62, 62, 32)	896

max_pooling2d_3 (MaxPooling2D)	(None, 31, 31, 32)	0

conv2d_4 (Conv2D)	(None, 29, 29, 32)	9248

max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 32)	0

flatten_2 (Flatten)	(None, 6272)	0

dense_3 (Dense)	(None, 128)	802944

dense_4 (Dense)	(None, 1)	129
=====		
Total params: 813,217		
Trainable params: 813,217		
Non-trainable params: 0		

DATASET PREPARATION: fit (training)

Homework #4

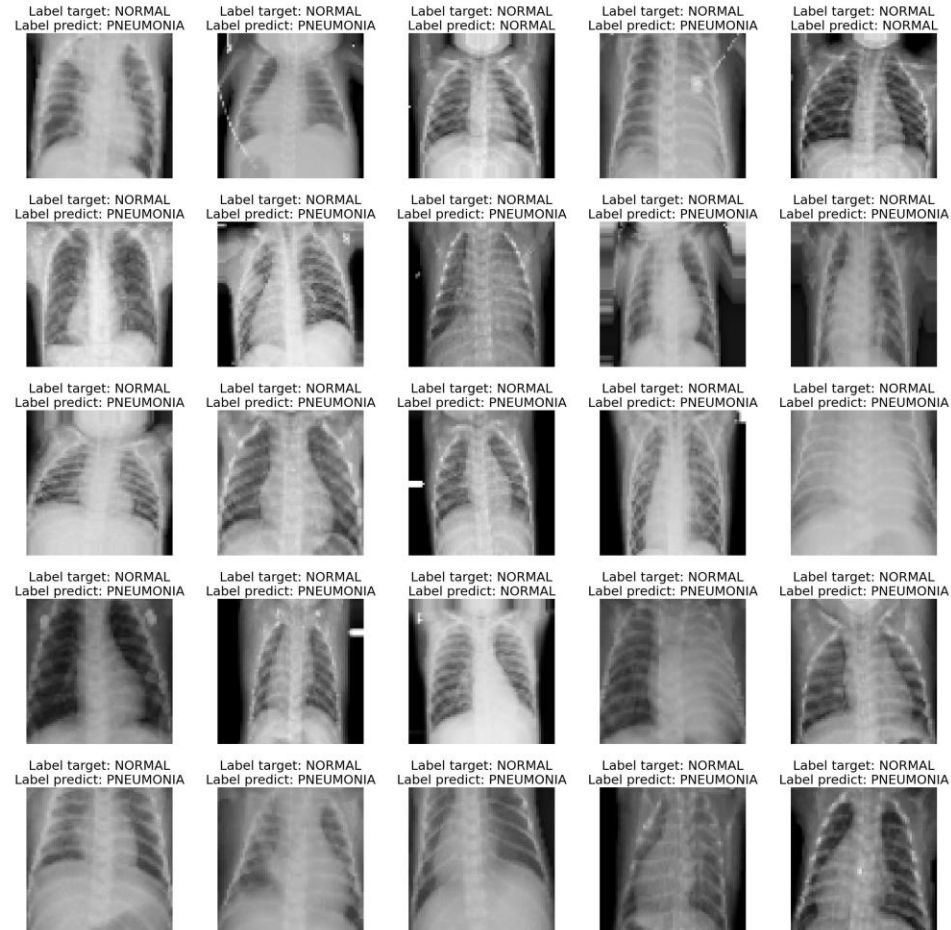
DATASET PREPARATION

- **Homework**

- 현재까지 수행한 python 및 jupyter파일을 github에 업로드. (Homework 1 & 2)
- x-ray로 학습 (training)을 하기위한 모델을 형성하고 모델의 layer를 작성.
(모델 성능에 대한 제한은 없으므로 성능이 낮더라도 상관없음.) (Homework 3)
- 학습시키기 위해 batchsize, epochs, validation데이터 셋을 model_fit에 넣어 트레이닝 수행.
(Homework #4)
- x-ray로 학습한 모델을 저장하고, 저장된 모델을 예측 (predict)하는 python코드를 작성해서
작성한 코드를 github에 upload (Homework #5).

DATASET PREPARATION

출력 예시 (Homework #5)

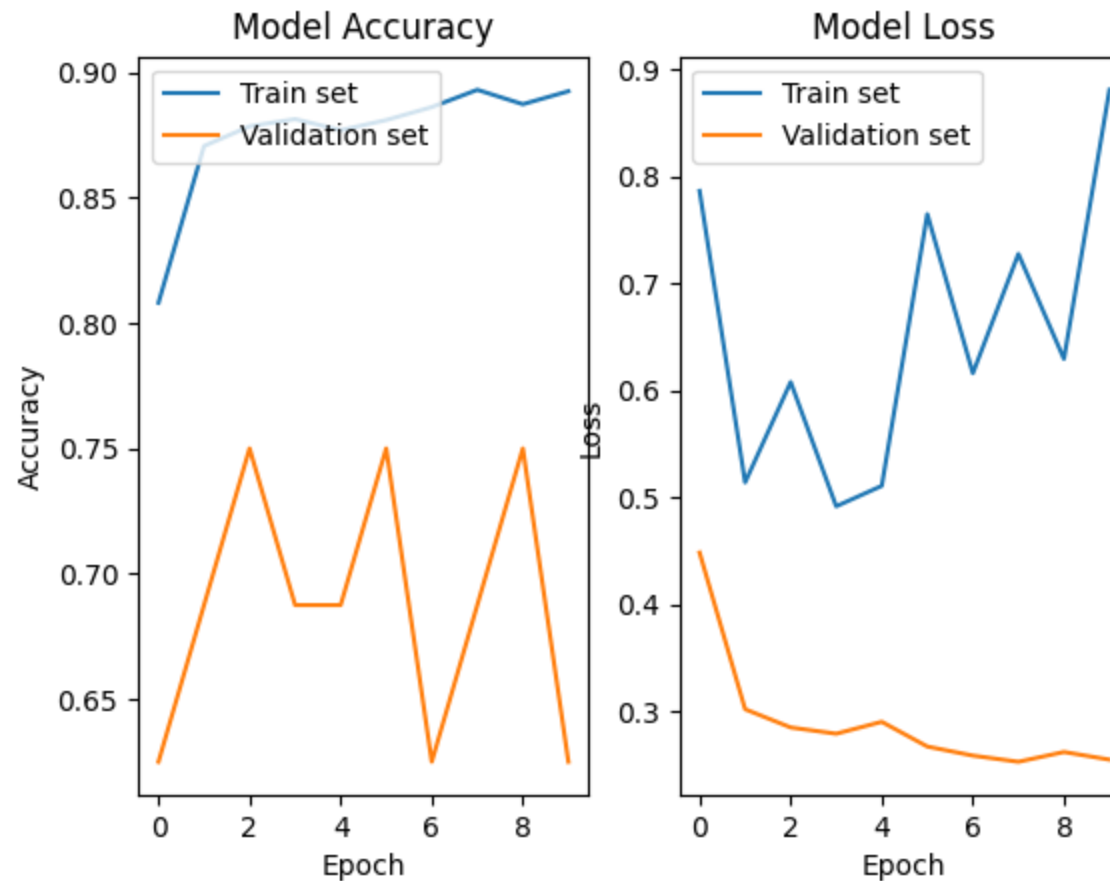


DATASET PREPARATION

- **Homework (2)**

- 학습한 모델의 성능분석을 다음페이지에 나온 코드를 활용해서 추출 한 후 github에 upload.

출력 예시 (Homework #6)



DATASET PREPARATION: model specification

```
# Accuracy
plt.plot(cnn_model.history['accuracy'])
plt.plot(cnn_model.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training set', 'Validation set'], loc='upper left')
plt.savefig('train_accuracy.png')
plt.show(block=False)
plt.clf()
```

```
# Loss
plt.plot(cnn_model.history['val_loss'])
plt.plot(cnn_model.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training set', 'Test set'], loc='upper left')
plt.savefig('train_loss.png')
plt.show(block=False)
plt.clf()
```



THANK YOU