```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
import cv2


# training, validation, test set data 나누는 함수(특징 세로방향)
def data_division(n_data, Tr_rate, V_rate, Te_rate):

    np.random.shuffle(n_data)                                   #데이터 섞기

    tr_index = int(len(n_data) * Tr_rate / 10)                  #Tr_set 비율만큼 데이터 index 양 확인
    v_index = int(len(n_data) * V_rate / 10)                    #V_set 비율만큼 데이터 index 양 확인
    te_index = int(len(n_data) * Te_rate / 10)                  #Te_set 비율만큼 데이터 index 양 확인

    #비율대로 data 나누기
    tr_set = n_data[0:tr_index]                                 # n_data의 0부터 tr_index - 1까지 tr_set에 저장
    v_set = n_data[tr_index : tr_index + v_index]               # tr_index 부터 tr_index + v_index - 1까지 v_set에 저장
    te_set = n_data[tr_index + v_index : tr_index + v_index + te_index]  # tr_index + v_index부터 tr_index + v_index, te_index까지 te_set에 저장
```

```
    return tr_set, v_set, te_set                                    #
tr_set, v_set, te_set을 반환함


# 받아온 파일의 데이터 x와 y로 자동 분류 해주는 함수

def make_input_output(M):


    for i in range(M.shape[1]):                                     #M의
column 수만큼 반복

        #i 가 0일 때 x_matrix에 M 처음 column값 저장 => np.column stack을 쓰기 위
함

        if i == 0:

            x_matrix = M[:, 0]                                      #M
의 첫번째 column 성분 값들 x_matrix에 저장


        # i가 그 외 값일 때부터 마지막 전 값까지

        elif i < (M.shape[1] - 1):

            x_matrix = np.column_stack([x_matrix, M[:, i]])         #M의
마지막 column 성분 제외한 값들 x_matrix에 저장


        # i가 마지막 값일 때

        else:

            y = M[:, i]                                             #M
의 마지막 column 성분 y에 저장 (데이터의 라벨 따로 저장)



    y = y.reshape(y.shape[0], 1)                                    #y
```

size 다듬기

```
    x_matrix_t = np.transpose(x_matrix)                                    #한
데이터에 대한 특징들 한 column으로 나타나기 위해 transpose

    y_t = np.transpose(y)                                                   #
데이터 라벨을 가로로 나열하기 위함



    return x_matrix_t, y_t
#x_matrix_t, y_t 반환함


# 데이터의 class수 세는 함수

def y_class(y):


    y_class = np.unique(y)
#unique를 이용해 다른 속성 개수를 셈 => class 수 계산

    Q = len(y_class)
#numpy array로 받아지기 때문에 길이를 셈


    return Q                                                                #
Q값 반환


# 데이터의 특징 수 세는 함수

def ch_count(y):


    Q = len(y)                                                              #
받아온 데이터의 열 개수를 셈


    return Q                                                                #
Q값 반환
```

```
# One-Hot Encoding 구현 함수
def One_Hot_Encoding(y):

    Q = y_class(y)                                                    # class 개수 Q에 저장
    y_vector = np.zeros((Q, y.shape[1]))                              # (class 개수, y data의 가로방향 길이)의 성분이 0인 matrix 생성

    # y의 길이만큼 반복
    for i in range(y.shape[1]):
        label = int(y[0, i])                                          # y는 정수로 가정
        y_vector[label, i] = 1                                        # label, i번째  자리를 1로 만듦

    return y_vector                                                   # y_vector

#row기반 dummy추가해주는 함수
def add_dummy(x):

    x_dummy = np.ones(x.shape[1])                                     # 입력데이터 x의 길이만큼 dummy 생성
    x = np.row_stack([x, x_dummy])
#row방향으로 쌓음
```

```
    return x                                                    # x
반환


#sigmoid 구현 함수
def sigmoid_function(z):


    return(1/(1 + np.exp(-z)))
#sigmoid 식 구현한 것 바로 반환



#대표값 찾아서 1로 만들어주는 함수
def classification_data_max(y):


    p = np.zeros_like(y)                                        #받
아온 y데이터의 row와 column 크기만큼 요소가 0인 matrix 생성


    y_max = np.argmax(y, axis = 0)                              #y
의 최댓값 index 저장


    #y 데이터 길이만큼 p (i번째 최댓값 index, i)에 1 저장
    for i in range(y.shape[1]):
        p[y_max[i], i] = 1


    return p                                                    #
p값 반환
```

```
#데이터 정확도 함수

def data_accuracy(y_h, y):                                                    #한
데이터에 대한 성분 row로 나열한 데이터기준


    count = 0
#count 기능 이용할 변수 0으로 초기화


    for i in range(y_h.shape[1]):                                             #예측
데이터 column 개수만큼 반복

        if (y_h[:, i] == y[:, i]).all():                                      #받아온 데
이터와 예측 데이터의 같은 column의 row성분 값이 모두 같은지 확인

            count += 1                                                        #
위 조건에 해당할 때 count


    accuracy = count / y_h.shape[1]
#count 된 수를 예측데이터 column 개수만큼 나눠줌


    return accuracy
#accuracy 반환



#batch size 1 forward_propagation 구현 함수

def forward_propagation_1(x_input_added_dummy, v_matrix, w_matrix, L):
#Hidden Layer의 node 수 지정


    alpha = np.dot(v_matrix, x_input_added_dummy)                             #
v와 xinput을 곱해 alpha를 구함

    b_matrix = sigmoid_function(alpha).reshape(-1, 1)                         #
```

batch size 1일 때 sigmoid에 넣으면 형태 깨져서 reshape이용

```
    b_matrix = add_dummy(b_matrix)                                    #
b에 dummy 추가

    beta = np.dot(w_matrix, b_matrix)                                 # w
와 b 곱해서 beta 구함
    y_hat = sigmoid_function(beta)                                    #
beta를 sigmoid function에 넣어 y_hat 구함

    return y_hat, b_matrix                                            #
y_gat, b_matrix 반환
```

#batch size 1 back propagation 구현 함수

```
def Back_Propagation_1(y_hat, y_data, x_matrix_added_dummy, b_matrix, w_prev, L): # w
먼저 weight update시키므로 update 전 w 입력 받음

    # w 기울기 구하는 코드
    delta = 2 * (y_hat - y_data.reshape(-1, 1)) * y_hat * (1 - y_hat)          #delta 구
함, y_data는 (:, 1)로 슬라이스 된 크기

    w_dif = np.dot(delta, b_matrix.T)
#delta와 b를 이용해 w의 기울기 구함

    # v 기울기 구하는 코드
    proc = np.dot(delta.T, w_prev)
```

```
    #dummy data 삭제
    b_matrix_h = np.delete(b_matrix, L, axis = 0)
    proc = np.delete(proc, L, axis = 1 )


    v_dif = np.dot((proc.T * b_matrix_h * (1 - b_matrix_h)),
x_matrix_added_dummy.reshape(1, -1))        # v의 기울기 구하기

    return w_dif, v_dif                                                    # 함
수의 반환값으로 w와 v의 기울기를 반환함


# batch size 1인 Two_Layer_Neural Network
def Two_Layer_Neural_Network_1(x_input, y_data, L, epoch, LR):

    MSE_list = []
#MSE 저장할 list
    ACCURACY_list = []
#accuracy 저장할 list
    x_matrix = add_dummy(x_input)                                          #
입력에 dummy data 추가

    M = ch_count(x_input)
#input 속성 수 체크
    Q = ch_count(y_data)
#ouput class 수 체크
```

```
    # weight 초기화
    v = np.random.rand(L, M + 1) * 2 - 1
    w = np.random.rand(Q, L + 1) * 2 - 1

    # epoch수 만큼 반복
    for i in range(epoch):

        y_hat_all_epoch = []                                                    #한
epoch마다 y_hat 저장하는 list 초기화

        #데이터 길이만큼 반복
        for j in range(y_data.shape[1]):
            w_prev = w.copy()
#update전 weight값 저장

            y_hat, b_matrix = forward_propagation_1(x_matrix[:, j], v, w, L)      #forward
propagation 진행
            y_hat_all_epoch.append(y_hat)
#y_hat 값 list에 저장

            w_dif, v_dif = Back_Propagation_1(y_hat, y_data[:, j], x_matrix[:, j], b_matrix,
w_prev, L)     #back propagation 진행

            #weight update
            w = w - LR * w_dif
            v = v - LR * v_dif
```

```
        y_hat_all = np.hstack(y_hat_all_epoch)
#y_hat을 쌓은 list에 numpy array를 배열로 만들어줌

        error = y_hat_all - y_data
#error 계산

        MSE = np.mean(error ** 2)
#MSE 계산

        MSE_list.append(MSE)
#MSE list에 저장


        P = classification_data_max(y_hat_all)                                  #데이
터 당 최댓값을 1로 만들어주는 분류 함

        accuracy = data_accuracy(P, y_data)
#accuracy 구하기

        ACCURACY_list.append(accuracy)
#accuracy list에 저장


    return MSE_list, ACCURACY_list, v, w
#MSE_list, ACCURACY_list, v, w 반환함


# confusion matrix 구현 함수

def confusion_matrix(y_hat, y_data):


    y_pred_index = np.argmax(y_hat, axis = 0)
#y_hat 데이터당 최댓값 index 가져옴

    y_true_index = np.argmax(y_data, axis = 0)
#y_data 데이터당 최댓값 index 가져옴
```

```
    true_num = 0                                                    #
정확히 예측한 횟수 초기화


    classes_num = ch_count(y_data)
#y_data class 수 체크


    confusion_matrix = np.zeros((classes_num + 1, classes_num + 1))          #정확
도 나타내기 위해 class수 + 1개만큼 정방 행렬 만듦


    #y 길이만큼반복

    for i in range(len(y_pred_index)):

        confusion_matrix[y_true_index[i], y_pred_index[i]] +=  1              #실제
값, 예측값 index에 해당하는 자리에 1 더함


    # class 수만큼 반복

    for i in range(classes_num):


        # row방향으로 더한 값이 0보다 클 때 전체 데이터로 정확히 예측한 값 나눠줌

        if sum(confusion_matrix[i, : classes_num]) > 0:

            confusion_matrix[i, classes_num] = confusion_matrix[i, i] /
np.sum(confusion_matrix[i, : classes_num])


        # column 방향으로 더한 값이 0보다 클 때 전체 데이터로 정확히 예측한 값 나
눠줌

        if sum(confusion_matrix[: classes_num, i]) > 0:

            confusion_matrix[classes_num, i] = confusion_matrix[i, i] /
np.sum(confusion_matrix[: classes_num, i])
```

```
            true_num += confusion_matrix[i, i]                                  #정
확히 예측한 값 세기

        confusion_matrix[classes_num, classes_num] = true_num / len(y_pred_index)    #전체
데이터에 대한 정확도 마지막 index에 저장


        return confusion_matrix
#confusion_matrix 반환



#clustering 이용 함수

def clustering(data):


        K = 10
# 군집 개수

        n, c = data.shape                                                       #
data의 row, column 수 n, c에 저장


        rand_idx = np.random.choice(n, K, replace=False)                        # n, c
값들중 random으로 뽑아 rand_idx에 저장

        m = data[rand_idx].copy()                                               #
data중 rand_idx에 해당하는 값 복사해 중심으로 초기화


        while(1):

            m_prev = m.copy()                                                   #
m값 복사해 m_prev에 저장


            clus = np.zeros(n)                                                  #
```

한 데이터의 특징 수가 크기이고 성분 0인 vector 생성

```
        # 한 데이터의 특징 수만큼 반복
        for i in range(n):
            d = (np.sum(((m - data[i]) ** 2), axis = 1)) ** (1 / 2)                #데이터와
각 중심과의 거리 계산(유클리드 거리: norm 2)
            clus[i] = np.argmin(d)                                                 #거
리중 제일 적은 거리의 index를 clus에 저장

        #군집 개수만큼 반복
        for j in range(K):
            clus_p = data[clus == j]
#clus데이터중 j군집에 속한 값들 clus_p에 저장

            #j군집에 아무것도 없을 경우 방지
            if len(clus_p) > 0:
                m[j] = np.mean(clus_p, axis = 0)                                   #j군
집의 모든 데이터의 평균을 새로운 군집의 중심으로 저장

        #새로운 중심과 이전 중심과 비교(소수점이라 loop 탈출 못할 가능성 존재하여
어느정도 근접하면 멈추게 해줌)
        if np.allclose(m_prev, m):
            break

    return clus, m                                                                 #
clustering label 데이터랑 중심값 반환
```

```
#사과, 복숭아, 토마토 분류 위한 저격 함수

def extract_class_distance(data, labels, clus, m, target_label):

    labels = np.array(labels, dtype=int)                                    #label
numpy array로 변환

    clus = np.array(clus, dtype=int)                                        #clus
numpy array로 변환

    max_count = 0


    #m 길이만큼 반복

    for k in range(m.shape[0]):

        cluster_labels = labels[clus == k]                                  #clus
중 k인 데이터값 모두 저장

        count = np.sum(cluster_labels == target_label)
#cluster label과 target label이 같은 수를 셈

        if count > max_count:
#count가 max_count보다 클 경우

            class_clus = k                                                  #
그 k값을 목표 class 군집이라고 판단, 저장

            max_count = count                                               #
count 값을 max_count에 옮김


    # # class_clus에 아무것도 입력 안됐을 때 0으로 채워 오류 피하기

    # if class_clus is None:

    #     d = np.zeros((data.shape[0], 1))
#data 크기만큼 성분 0인 벡터 생성

    # # peach_clus_idx에 어떤 값 입력 됐을 때

    # else:

    center = m[class_clus]                                                  #그
```

```
index를 목표 class 군집의 중심이라 설정
    d = np.log1p(np.sum(((center - data) ** 2), axis = 1)) ** (1 / 2)          #그 중심과
데이터들의 거리 구하기
    d = d.reshape(-1, 1)                                                        #d
vector 모양 다듬기

    return d                                                                    #
목표 class 군집과 데이터들 간 거리 벡터 d 반환함

#데이터 정규화 함수
def standard_data (input_data):
    mean = np.mean(input_data, axis=0)                                          #
데이터의 평균
    std = np.std(input_data, axis=0)                                            #표준
편차

    # 표준편차가 0인 경우 방지 => 분모 0되면 오류남
    std[std == 0] = 1

    input_data[:, :] = (input_data - mean) / std                                #마지막
label데이터 제외 정규화시킴

    return input_data
#input data 반환

#초록 분류
def No1_feature(input_data):
```

```
    feature = input_data[:, :, 1].mean()                                    #초록
색만 뽑아 평균 취하기, feature에 저장


    return feature
#feature 반환


#빨강 분류

def No2_feature(input_data):

    feature = input_data[:, :, 0].mean()                                    #빨간
색만 뽑아 평균취하기, featrue에 저장


    return feature
#feature 반환


# 가로 분산 => 줄무늬 같은 패턴 분류

def No3_feature(input_data):

    row_mean = np.mean(input_data, axis = (1, 2))                           # 세
로줄 평균 취한 것 저장 => 가로방향 밝기 변화

    var = np.var(row_mean)                                                  #
위에서 구한 값 분산 구하기


    return var                                                              #
var 반환


#청사과, 사과, 복숭아 분류 위함 => 복숭아는 노란색깔 성분 많이 가짐 => 노랑 초록
비율 사과보다 높을 것., 청사과는 빨강 비율 적음

def No4_feature(input_data):

    red_mean = input_data[:, :, 0].mean()                                   #빨간
```

```
색 밝기 평균
    blue_mean = input_data[:, :, 2].mean()                    #파란
색 밝기 평균
    green_mean = input_data[:, :, 1].mean()                   #초록
색 밝기 평균

    feature = (blue_mean + green_mean) / red_mean                  #
초록밝기 + 파랑밝기와 빨강밝기 비율 특징으로 저장

    return feature

#데이터 전처리, 특징추출 함수
def select_features(directory):
    #이미지 파일 directory

    #이미지 파일 directory 안의 파일 이름들 문자열 리스트로 저장
    file_list = os.listdir(directory)

    #특징 저장할 list
    feature_1_list = []
    feature_2_list = []
    feature_3_list = []
    feature_4_list = []

    label = []        #정답 라벨
```

```
# file_list에 있는 값들 반복
for name in file_list:

    #경로 설정함
    path = os.path.join(directory, name)

    #라벨 불러오기(파일명 첫 숫자)
    label.append(int(name.split('_', 1)[0]))

    #이미지 읽고 RGB(red, green, blue)값으로 변환하기
    img_GRB = cv2.imread(path)
    img_RGB = cv2.cvtColor(img_GRB, cv2.COLOR_BGR2RGB)

    #특징추출하기

    # 특징 1: 초록색 분류기
    feature_1 = No1_feature(img_RGB)

    # 특징 3: 빨간색 분류기
    feature_2 = No2_feature(img_RGB)

    # 가로 전체 밝기 분산 ==> 줄무늬같은 패턴 탐지
    feature_3 = No3_feature(img_RGB)
```

```
        # 빨간색밝기 평균에 대한 파랑, 초록 밝기 평균 비율
        feature_4 = No4_feature(img_RGB)

        #각 리스트에 각특징값 append
        feature_1_list.append(feature_1)
        feature_2_list.append(feature_2)
        feature_3_list.append(feature_3)
        feature_4_list.append(feature_4)

    #각 특징 리스트들 numpy array로 변환
    feature_1_list = np.array(feature_1_list)
    feature_2_list = np.array(feature_2_list)
    feature_3_list = np.array(feature_3_list)
    feature_4_list = np.array(feature_4_list)

    #features 한군데에 모으기
    features = np.column_stack([feature_1_list, feature_2_list, feature_3_list, feature_4_list])

    #clustering 이용
    cluster_features, m = clustering(features)
#clustering label, 중심 얻기

    #복숭아 군집 중심에 대한 거리 뽑는 함수
    peach_dist = extract_class_distance(features, label, cluster_features, m, target_label =
6)
```

```
    #토마토 군집 중심에 대한 거리 뽑는 함수
    tomato_dist = extract_class_distance(features, label, cluster_features, m, target_label = 8)

    #복숭아, 토마토, 사과 구분위한 특징까지 포함한 features
    features = np.column_stack([features, peach_dist, tomato_dist])

    # features 정규화
    features = standard_data(features)

    return features, label
#features와 label 반환



directory = "C:₩₩Users₩₩kim07₩₩Desktop₩₩Machinlearning_Workplace₩₩train"
#directory 설정
L = 80
#hidden node 수 설정
LR = 0.01
#learning rate 설정
epoch = 1000
#epoch(학습 횟수) 설정

# training set, validation set, test set 데이터 비율 설정
```

```
Tr_rate = 7
#training data 비율 7 설정

Val_rate = 3
#validation data 비율 3 설정

Te_rate = 0
#test data 비율 0 설정



data, label = select_features(directory)                                    #특징
추출한 데이터와 라벨 받기


total_data = np.column_stack([data, label])                                  #특징
추출 데이터와 라벨 합친 total 데이터 생성


tr_set, val_set, te_set = data_division(total_data, Tr_rate, Val_rate, Te_rate) # training set,
validation set, test set 나누기


x_features, y_label = make_input_output(tr_set)
#training set의 특징추출 데이터와 label 분리하기

x_features_val, y_label_val = make_input_output(val_set)
#validation set의 특징추출 데이터와 label 분리하기


y_data = One_Hot_Encoding(y_label)
#training set label one_hot encoding 하기

y_data_val = One_Hot_Encoding(y_label_val)
#validation set label one_hot encoding 하기
```

#training set을 이용한 신경망 학습

MSE_tr, Accuracy_tr, v_tr, w_tr = Two_Layer_Neural_Network_1(x_features, y_data, L, epoch, LR) #training set에 대한 MSE, ACCURACY, (v, w) weight 저장

y_hat_val_all = [] #test set의 y_hat 저장할 list

x_features_val = add_dummy(x_features_val)
#dummy 추가

training set으로 학습한 weight로 validation set forwoard propagation

#batch size 1이므로 데이터 하나씩 forward propagation

for i in range(x_features_val.shape[1]):

    y_hat_val, _ = forward_propagation_1(x_features_val[:, i], v_tr, w_tr, L) #forward propagation 진행

    y_hat_val_all.append(y_hat_val)
#y_hat_te list에 저장

y_hat_val = np.hstack(y_hat_val_all) #

#validation set에 대한 confusion matrix

confusion_matrix_val = confusion_matrix(y_hat_val, y_data_val)

weight csv 파일로 넘기기

#
==========

```
==================

# df_v = pd.DataFrame(v_tr)

# df_w = pd.DataFrame(w_tr)

#

# df_v.to_csv("w_hidden.csv", index = False, header = False)

# df_w.to_csv("w_output.csv", index = False, header = False)

#
============================================================
==================


parameters = {"axes.labelsize": 20, "axes.titlesize": 30, 'xtick.labelsize': 12, "ytick.labelsize":
12, "legend.fontsize": 12}

plt.rcParams.update(parameters)



plt.figure()

plt.plot(MSE_tr, '-o', markevery = 50, label = 'MSE')

plt.xlabel("epoch")

plt.ylabel("MSE")

plt.title("Training Set MSE")

plt.legend()

plt.grid()


plt.figure()

plt.plot(Accuracy_tr, '-o', markevery = 50, label = 'Accuracy')

plt.xlabel("epoch")
```

```
plt.ylabel("Accuracy")

plt.title("Training Set Accuracy")

plt.legend()

plt.grid()
```