109550156 HW1 Report

Part 1

(一) 圖像分類任務及資料集介紹

在 part 1 中，我選擇 Kaggle 的 Garbage Classification Dataset 作為資料集，此圖像分類任務目標是把圖像中的物體分類成 6 個類別，分別是紙板、玻璃、金屬、紙張、塑料和垃圾。(https://www.kaggle.com/datasets/asdasdasasdas/garbage-classification)

Training dataset 中，包含 287 張紙板、354 張玻璃、286 張金屬、403 張紙張、347 張塑料和 91 張垃圾的圖像。共 1768 張 512 * 384 的圖像。Testing dataset 中，包含 70 張紙板、82 張玻璃、68 張金屬、108 張紙張、74 張塑料和 29 張垃圾的圖像。共 431 張 512 * 384 的圖像。因為此資料集已提供測試資料集，所以 Part1 將不使用 Cross-validation 評估，直接用測試集進行表現評估。

在此圖像分類的任務中，我選擇使用簡單層數少的 MLP 以及 CNN 作為我的分類模型。

MLP 結構:

```python
class MLP(nn.Module):

    def __init__(self):
        super(MLP, self).__init__()
        self.input_layer = nn.Linear(48 * 48 * 3, 4096)
        self.output_layer = nn.Linear(4096, 6)

    def forward(self, x):
        x = x.view(-1, 48 * 48 * 3)
        x = self.input_layer(x)
        x = F.relu(x)
        x = self.output_layer(x)
        x = torch.softmax(x, dim=1)
        return x
```

CNN 結構:

```python
class CNN(nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=2),
                    nn.ReLU(),
                    nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=2),
                    nn.ReLU(),
                    nn.MaxPool2d(kernel_size=2),
        )
        self.out = nn.Linear(32 * 13 * 13, 6)
```

(二) 使用不同訓練資料量來比較分類的表現

MLP 和 CNN 中 batch size, learning rate 等參數皆一致，訓練 300 個 epoch。

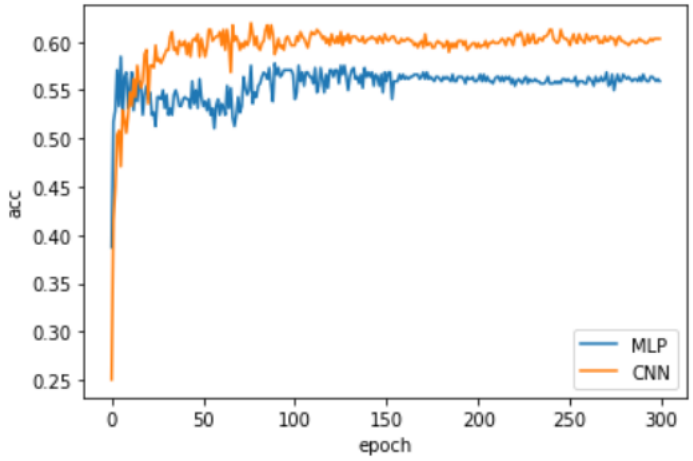
	MLP	CNN	MLP	CNN	MLP	CNN
訓練資料量	**precision**		**recall**		**F1-score**	
25% (442)	0.51	0.57	0.51	0.52	0.50	0.50
50% (884)	0.53	0.62	0.52	0.58	0.52	0.56
75% (1326)	0.54	0.60	0.53	0.57	0.53	0.55
100% (1768)	0.55	0.63	0.55	0.60	0.55	0.58

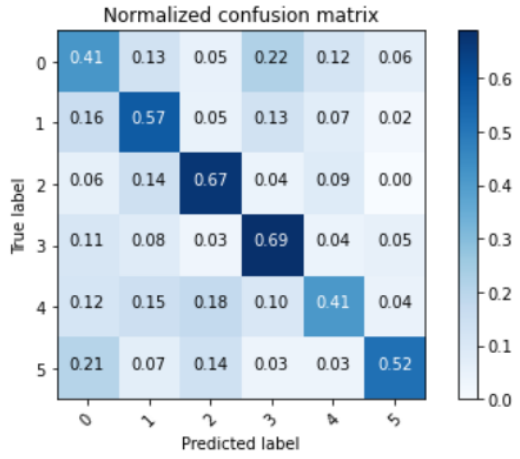
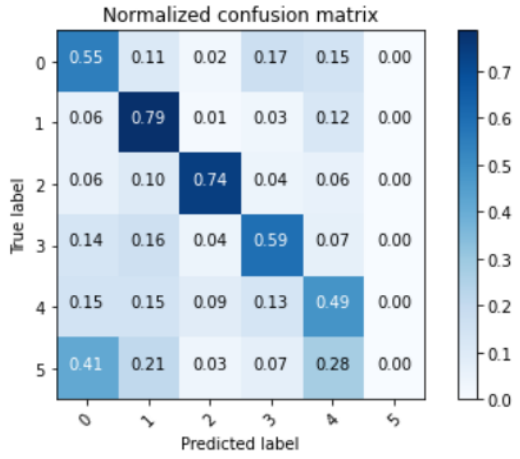
觀察分析:

透過比較在不同訓練資料量下的表現，發現 MLP 和 CNN 這兩個模型都會隨著訓練資料量越小而表現越差，但是，在只有 25%資料量時，兩者仍有 0.5 的 F1-score，模型對於 6 個類別還是有一定的分類能力。而相較於 CNN，MLP 在這個實驗設置下被資料量影響的程度較小。

(三) 比較 MLP 和 CNN 的分類表現

此實驗將使用整個訓練資料集訓練 300 個 epoch 評估 MLP 及 CNN 的分類表現，使用圖表呈現實驗數據。



MLP	CNN
	
Classification Report: precision recall f1-score support 0 0.49 0.46 0.48 82 1 0.63 0.56 0.60 108 2 0.62 0.64 0.63 70 3 0.51 0.68 0.58 74 4 0.49 0.41 0.45 68 5 0.52 0.52 0.52 29 accuracy 0.55 431 macro avg 0.54 0.55 0.54 431 weighted avg 0.55 0.55 0.55 431	Classification Report: precision recall f1-score support 0 0.52 0.55 0.53 82 1 0.66 0.79 0.72 108 2 0.80 0.74 0.77 70 3 0.59 0.59 0.59 74 4 0.44 0.49 0.46 68 5 1.00 0.00 0.00 29 accuracy 0.60 431 macro avg 0.67 0.53 0.51 431 weighted avg 0.63 0.60 0.58 431

觀察分析:

評估的表現如上面的圖表呈現，上圖 1 表現出了 MLP 和 CNN 這兩個模型在此任務上都易於收斂，約於第 100 個 epoch 就會出現收斂的情況，且 CNN 整體的表現會比 MLP 較好。

從兩個模型的 Classification Report 來看，CNN 在整體的 f1-score 的表現較佳，但其中較值得注意的是，CNN 在這個設置下不傾向預測類別 5 (金屬類)，因此對於類別 5 的分類能力較差，而 MLP 對於各類別的分類表現較於平均。

(四) 比較各個模型調整超參數後的表現

此實驗將調整 MLP 及 CNN 的 Learning rate 作為實驗的變因。

	MLP	CNN	MLP	CNN	MLP	CNN
Learning rate	**Precision**		**Recall**		**F1-score**	
0.0001	0.60	0.51	0.55	0.49	0.54	0.45
0.0005	0.59	0.61	0.55	0.58	0.54	0.55
0.001	0.57	0.61	0.54	0.57	0.52	0.55
0.005	0.55	0.64	0.55	0.61	0.55	0.59
0.01	0.55	0.63	0.55	0.60	0.55	0.58
0.05	0.64	0.70	0.49	0.16	0.43	0.01
0.1	0.82	0.32	0.25	0.23	0.15	0.12

觀察分析:

從此實驗結果來看，可以看到不管是 MLP 還是 CNN，learning rate 越大會造成模型表現變差，更有可能沒有學到任何東西。另一方面，learning rate 變小也造成了模型表現稍微下降的趨勢，例如 CNN 在 learning rate 從 0.0005 變成 0.0001 時，f1-score 下降幅度增大，因此從此實驗得知，對於這兩種模型，太大或太小的 learning rate 都將使分類任務的表現變差。

Part 2

(一) 非圖像分類任務及資料集介紹

在 part 2 中，我選擇 Kaggle 的 Wine Quality Dataset 作為資料集，此分類任務目標是把葡萄酒分類成 6 個品質類別 (3 到 8)，3 的意思為品質最差，8 為最好。(https://www.kaggle.com/datasets/yasserh/wine-quality-dataset)

資料集中，包含 1143 個 instances，每個 instance 具有 11 個數值特徵，例如固定酸度、揮發性酸度、pH 值等等。且因為此資料集未提供測試資料集，所以

Part2 的每個實驗將使用 10-fold cross-validation 進行評估。

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	5
7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	5
11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	6
7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7.4	0.66	0	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	5
7.9	0.6	0.06	1.6	0.069	15	59	0.9964	3.3	0.46	9.4	5
7.3	0.65	0	1.2	0.065	15	21	0.9946	3.39	0.47	10	7

在此葡萄酒品質分類任務中，我選擇使用 K-NN 和 Logistic Regression 作為我的分類模型。

(二) 使用不同訓練資料量來比較分類的表現

K-NN: K = 10

Logistic Regression (LR): solver = L-BFGS

	K-NN	**LR**	**K-NN**	**LR**	**K-NN**	**LR**	**K-NN**	**LR**
訓練資料量	**Accuracy**		**Precision**		**Recall**		**F1-score**	
25% (285)	0.55	0.60	0.59	0.64	0.55	0.60	0.50	0.56
50% (571)	0.58	0.60	0.60	0.63	0.58	0.60	0.55	0.57
75% (756)	0.57	0.60	0.59	0.62	0.57	0.60	0.55	0.58
100% (1143)	0.59	0.59	0.60	0.60	0.59	0.59	0.57	0.57

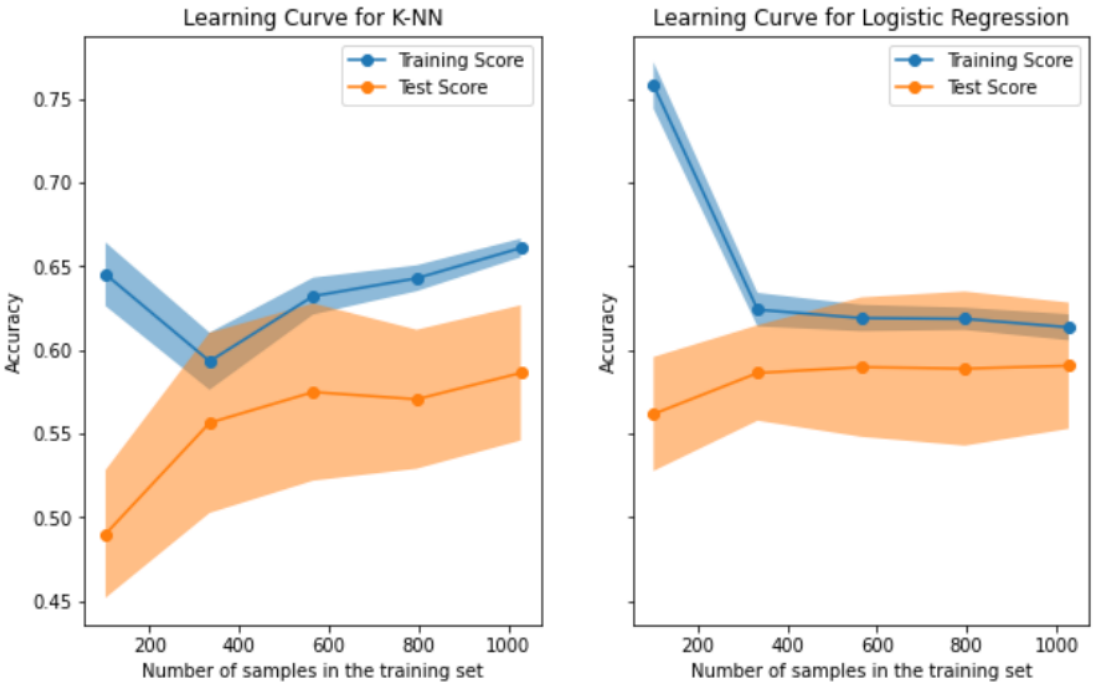
觀察分析:

透過比較在不同訓練資料量下的表現，發現 K-NN 在訓練資料量越多時表現會越來越好；而對於 Logistic Regression 來說，在此任務下，資料量的大小變化對於分類表現差異不大。

(三) 比較 K-NN 和 Logistic Regression 的分類表現

K-NN: K = 10

Logistic Regression (LR): solver = L-BFGS



觀察分析:

從此實驗結果和上個使用不同資料量訓練的實驗來看，可以發現到 K-NN 的分類能力會隨著資料量越多而有所提升；Logistic Regression (LR)在此任務下則只需要少量的資料即可收斂，如果比較這兩個演算法，可以發現 LR 和 K-NN 最後的各項評估表現是差不多的，但是，LR 擁有著比 K-NN 需要更少資料量就能達到好表現的優勢。

(四) 比較各個模型調整超參數或設定後的表現

K-NN: 實驗將把 K 作為變因調整，比較不同的 K 值下 K-NN 對此任務的分類表現。

K-value	Accuracy	Precision	Recall	F1-score
1	0.90	0.91	0.90	0.90
5	0.91	0.93	0.91	0.91
10	0.89	0.91	0.89	0.89
15	0.89	0.91	0.89	0.89
20	0.66	0.80	0.66	0.63

25	0.44	0.45	0.44	0.35

觀察分析:

從此實驗結果來看，可以發現到當 K-NN 的 K 值於 15 以下時，仍能維持約有 0.9 F1-score 的表現，但是當 K 值越來越大時，很明顯的可以觀察到模型表現下降。

Logistic Regression (LR): 此實驗變因為使用不同的 Solver 來進行分類表現比較，因為是多類別分類任務，所以僅包含 Newton-CG、SAG、SAGA 和 L-BFGS。

Penalty	**Accuracy**	**Precision**	**Recall**	**F1-score**
L-BFGS	0.59	0.60	0.59	0.57
Newton-CG	0.59	0.60	0.59	0.57
SAG	0.59	0.60	0.59	0.57
SAGA	0.59	0.60	0.59	0.57

觀察分析:

從此實驗結果來看，可以發現到對於這個資料集，LR 使用不同的優化器都得到一樣的表現，所以可以推測，在做此分類任務時，solver 的不同並不會大幅影響分類表現，因為訓練資料集並無偏好任何一種演算法。

Part 3

(一) 自製分類任務及自製資料集介紹

在 part 3 中，我與林哲安(109550088)共同收集了來自大賣場、商店所販賣的各種飲料、餅乾、糖果、泡麵的營養成分表作為資料集，並以各自的類別標註它們，共 4 種類別。分類任務的目標是分類某一個商品屬於 4 個類別中的哪一類。

資料集收集了 70 個品項，每個品項具有 5 個數值特徵，分別為熱量、蛋白質、飽和脂肪、碳水化合物和鈉。此資料集未收集測試資料集，所以 Part3 的每個實驗將使用 10-fold cross-validation 進行評估。

Name	Calories	Protein	Saturated Fat	Total Carbohydrate	Sodium	Class
多力多滋三角玉米片－超濃起	522	8	13	64	669	0
品客洋芋片-原味	520	4.7	12.7	61.3	500	0
樂事九州岩燒海苔味洋芋片	564	7.2	17.5	55	585	0
奧利奧原味夾心餅乾	489	4.7	9.9	71.3	520	0
旺旺仙貝	477	4.2	8.8	74.8	778	0
旺仔小饅頭	381	2.9	0.4	90.2	61	0
奇多隨口脆岩燒海苔口味	584	4.6	20.2	54.7	539	0
樂天小熊餅-草莓	533	3.5	21.5	61.8	213	0
盛香珍夾心酥-花生	535	6.2	12.7	62.5	162	0
真魷味-紅燒口味	432.3	6	6.4	74.4	1133	0
多力多滋三角玉米片－黃金起	521	8.1	13.5	62.5	739	0
麥香綠茶	38	0	0	9.5	14	1
美祿巧克力麥芽飲品減糖配方	70	2.2	1.3	10.3	35	1
貝納頌-曼特寧	58	1.7	1.4	8.1	49	1
立頓英式奶茶	36.9	0.5	0.4	6.9	17.3	1
飲冰室綠奶茶	56.9	0.2	1.7	10.2	12	1

在此商品分類任務中，我選擇使用 Random Forest 和 AdaBoost 作為我的分類模型。

(二) 使用不同訓練資料量來比較分類的表現

Random Forest (RF): Criterion = Gini

AdaBoost (AB): learning_rate = 1.0

	RF	**AB**	**RF**	**AB**	**RF**	**AB**	**RF**	**AB**
訓練資料量	Accuracy		Precision		Recall		F1-score	
25% (18)	0.85	0.75	0.95	0.93	0.85	0.75	0.83	0.75
50% (38)	0.78	0.60	0.95	0.92	0.78	0.60	0.79	0.57
75% (54)	0.88	0.64	0.93	0.77	0.88	0.64	0.87	0.58
100% (70)	0.9	0.70	0.94	0.83	0.90	0.70	0.90	0.62

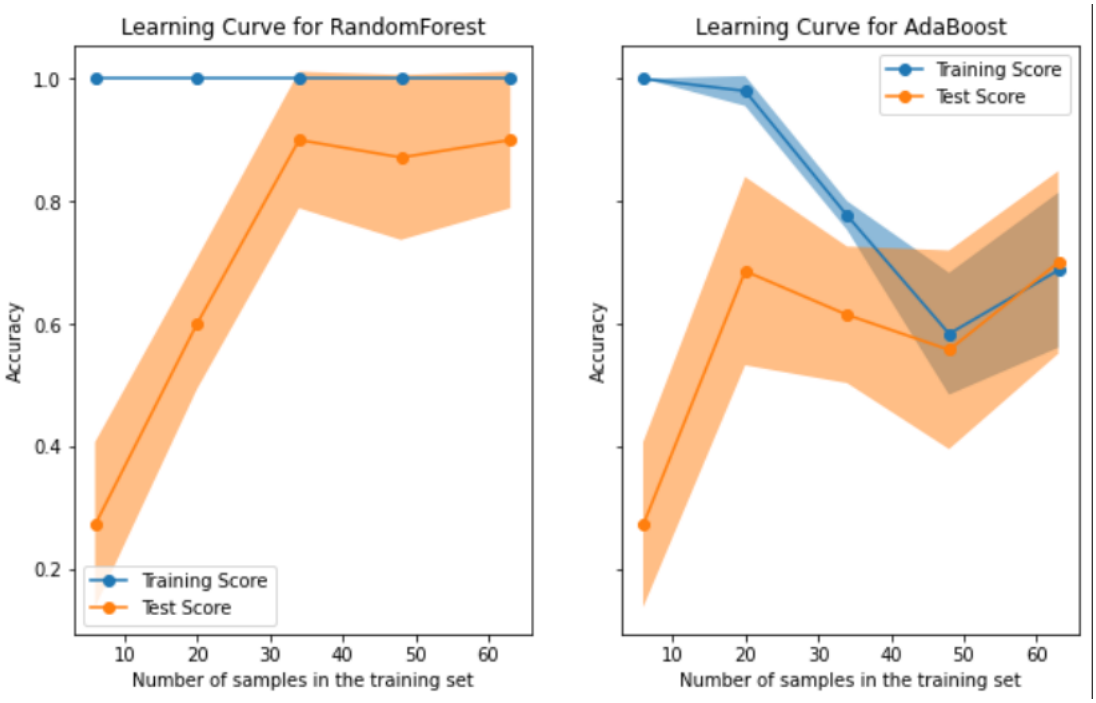
觀察分析:

透過比較在不同訓練資料量下的表現，發現 Random Forest 在訓練資料越多的情況下表現會越好，但 Adaboost 在不同訓練資料量下表現卻不一定。

(三) 比較 Random Forest 和 AdaBoost 的分類表現

Random Forest (RF): Criterion = Gini

AdaBoost (AB): learning_rate = 1.0



觀察分析:

從此實驗結果和上個使用不同資料量訓練的實驗來看，可以發現到對於此分類任務，Random Forest 整體的表現大於 AdaBoost，Random Forest 表現好的原因可能是因為資料集各項類別的特徵足夠明顯，容易被基於 Decision Tree 的 Random Forest 所分類，因此有很好的效果。

(四) 比較各個模型調整超參數或設定後的表現

Random Forest: 實驗將把最大深度作為變因調整，比較不同的最大深度下 Random Forest 對此任務的分類表現。

Max depth	Accuracy	Precision	Recall	F1-score
1	0.86	0.92	0.86	0.84
2	0.89	0.93	0.89	0.89
3	0.89	0.93	0.89	0.89
4	0.89	0.93	0.89	0.89
5	0.90	0.94	0.9	0.90

觀察分析:

從此實驗結果來看，可以發現到在此較小的資料集上，當最大深度越大，模型能夠小幅度的增加表現，並不會像大資料集一樣，深度太大會造成 overfitting。

AdaBoost: 此實驗將調整 Learning rate 作為實驗的變因，並比較使用不同的 Learning rate 的分類表現。

Learning rate	**Accuracy**	**Precision**	**Recall**	**F1-score**
0.001	0.71	0.89	0.71	0.70
0.005	0.86	0.93	0.86	0.84
0.01	0.89	0.94	0.89	0.88
0.05	0.73	0.85	0.73	0.68
0.1	0.8	0.89	0.8	0.77
0.5	0.79	0.86	0.79	0.76
1.0	0.7	0.83	0.7	0.62

觀察分析:

從此實驗結果來看，可以發現到對於這個資料集，AdaBoost 使用不同 learning rate 會產生分類表現的浮動，當 learning rate 太小或太大時，都會造成分類表現不佳，因此對於不同的資料集，AdaBoost 的 learning rate 會有各自適合的選擇。

Appendix

Dataset files & Source codes:

Google Drive:

https://drive.google.com/drive/folders/1b0ZZBuya8thYCEg2f7G4ne7HXX85_U7b?usp=sharing

Code:

For part 1:

```python
import torch
import torch.nn as nn
import os
import numpy as np
import torch.nn.functional as F
import itertools
from torch import Tensor
from torch.optim import SGD
from torch.nn import CrossEntropyLoss
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from tqdm import tqdm
import matplotlib.pyplot as plt
from google.colab import drive

drive.mount('/content/drive')
```

0.0s Python

```python
def train_model(train_dl, model, test_dl, gpu_mode, draw_plt, device):
    # define the optimization
    criterion = CrossEntropyLoss()
    optimizer = SGD(model.parameters(), lr=0.1, momentum=0.9)
    # enumerate epochs
    performance = []
    for _ in tqdm(range(300)):
        if draw_plt:
            model.train()
        # enumerate mini batches
        for i, (inputs, targets) in enumerate(train_dl):
            if gpu_mode:
              inputs = inputs.to(device)
              targets = targets.to(device)
            targets_one_hot = F.one_hot(targets, num_classes=6)

            targets_one_hot = targets_one_hot.view(-1, 6).to(torch.float)
            # clear the gradients
            optimizer.zero_grad()
            # compute the model output
            pred = model(inputs)
            pred = pred.view(-1, 6)

            # calculate loss
            loss = criterion(pred, targets_one_hot)
            # credit assignment
            loss.backward()
            # update model weights
            optimizer.step()

        if draw_plt:
            model.eval()
            performance.append(evaluate_model(test_dl, model, gpu_mode))

    return performance
```

```python
# evaluate the model
def evaluate_model(test_dl, model, gpu_mode, device):
    predictions, actuals = [], []
    for i, (inputs, targets) in enumerate(test_dl):
        # evaluate the model on the test set
        if gpu_mode:
          inputs = inputs.to(device)
          targets = targets.to(device)
        yhat = model(inputs)
        # retrieve numpy array
        actual = targets.cpu().numpy()
        actual = actual.reshape((-1, 1))
        # round to class values
        yhat = torch.argmax(yhat, dim=1).cpu().numpy()
        yhat = yhat.reshape((-1, 1))
        predictions.append(yhat)
        actuals.append(actual)
    predictions = np.vstack(predictions)
    actuals = np.vstack(actuals)

    return accuracy_score(actuals, predictions)
```

```python
# make a class prediction for one row of data
def predict(row, model):
    # convert row to data
    row = Tensor([row])
    # make prediction
    pred = model(row)
    # retrieve numpy array
    pred = np.argmax(pred.detach().numpy())
    return pred
```

✓ 0.0s Python

```python
class Dataset(torch.utils.data.Dataset):
    def __init__(self, train_X, train_y):
        self.X = torch.from_numpy(train_X)
        self.labels = torch.from_numpy(train_y.astype(int)).view(-1, 1) - 1

    def __len__(self):
        return len(self.X)

    def __getitem__(self, index):
        return self.X[index], self.labels[index]
class MLP(nn.Module):

    def __init__(self):
        super(MLP, self).__init__()
        self.input_layer = nn.Linear(48 * 48 * 3, 4096)
        self.output_layer = nn.Linear(4096, 6)

    def forward(self, x):
        x = x.view(-1, 48 * 48 * 3)
        x = self.input_layer(x)
        x = F.relu(x)
        x = self.output_layer(x)
        x = torch.softmax(x, dim=1)
        return x
```

```python
class CNN(nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=3,
                out_channels=16,
                kernel_size=3,
                stride=1,
                padding=2,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 3, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        self.out = nn.Linear(32 * 13 * 13, 6)
```

```python
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)
        x = self.out(x)
        x = torch.softmax(x, dim=1)
        return x
```

✓ 0.0s Python

```python
is_colab = False
draw_plt = True
gpu_mode = True

batch_size = 64

if torch.cuda.is_available():
    dev = "cuda"
else:
    dev = "cpu"
device = torch.device(dev)
```

✓ 0.0s Python

```python
part = 'part1'
dataset_path = os.path.join('datasets', part)
if is_colab:
    dataset_path = os.path.join('/content/drive/MyDrive/AI_hw/hw1', dataset_path)

train_X_path  = os.path.join(dataset_path, 'train_X.npy')
train_y_path  = os.path.join(dataset_path, 'train_y.npy')
test_X_path = os.path.join(dataset_path, 'test_X.npy')
test_X_path = os.path.join(dataset_path, 'test_y.npy')

train_X = np.load(os.path.join(dataset_path, 'train_X.npy'))
train_y = np.load(os.path.join(dataset_path, 'train_y.npy'))
test_X = np.load(os.path.join(dataset_path, 'test_X.npy'))
test_y = np.load(os.path.join(dataset_path, 'test_y.npy'))
```

✓ 0.0s Python

```python
train_dataset = Dataset(train_X, train_y)
test_dataset = Dataset(test_X, test_y)
```

✓ 0.0s Python

```python
train_indices = list(range(len(train_X)))
test_indices = list(range(len(test_X)))

train_subsampler = torch.utils.data.SubsetRandomSampler(train_indices)
test_subsampler = torch.utils.data.SubsetRandomSampler(test_indices)
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, sampler=train_subsampler)
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, sampler=test_subsampler)

CNN = CNN()
MLP = MLP()
models = [(MLP, 'MLP'), (CNN, 'CNN')]
plt.cla()
plt.xlabel('epoch')
plt.ylabel('acc')
for model, model_name in models:
    if gpu_mode:
        model.to(device)
    performance_per_epoch = np.array(train_model(trainloader, model, testloader, gpu_mode, draw_plt, device))
    with torch.no_grad():
        acc = evaluate_report(testloader, model, gpu_mode, device)

    if draw_plt:
        plt.plot(performance_per_epoch, label = model_name)
        plt.legend(loc = 'lower right')
```

For part 2 & part 3:

```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import itertools
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold, LearningCurveDisplay, cross_validate
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
from sklearn.feature_selection import VarianceThreshold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.pipeline import make_pipeline
from google.colab import drive


drive.mount('/content/drive')
```

3.5s Python

```python
def evaluate_model(data_x, data_y, model):
    k_fold = KFold(10, shuffle=True, random_state=1)

    predicted_targets = np.array([])
    actual_targets = np.array([])

    for train_ix, test_ix in k_fold.split(data_x):
        train_x, train_y, test_x, test_y = data_x[train_ix], data_y[train_ix], data_x[test_ix], data_y[test_ix]

        # Fit the classifier
        classifier = model.fit(train_x, train_y)

        # Predict the labels of the test set samples
        predicted_labels = classifier.predict(test_x)

        predicted_targets = np.append(predicted_targets, predicted_labels)
        actual_targets = np.append(actual_targets, test_y)

    return predicted_targets, actual_targets
```

```python
def load_dataset(part):
    if part == 2:
        path = '/content/drive/MyDrive/AI_hw/hw1/datasets/part2'
        dataset = pd.read_csv(os.path.join(path, "WineQT.csv")).drop(columns=["Id"]).values
        X_train = dataset[:,:-1].astype(float)
        y_train = dataset[:,-1].astype(float)
    elif part == 3:
        path = '/content/drive/MyDrive/AI_hw/hw1/datasets/part3'
        dataset = pd.read_csv(os.path.join(path, "dataset.csv")).values
        X_train = dataset[:,1:-1].astype(float)
        y_train = dataset[:,-1].astype(float)
    else:
        return None, None
    return X_train, y_train
```

Python

```python
# load dataset
part = 3
X_train, y_train = load_dataset(part)
class_names = np.unique(y_train)
selector = VarianceThreshold()
X_train = selector.fit_transform(X_train)
y_train = y_train.ravel()

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)

scoring = {'accuracy' : make_scorer(accuracy_score),
           'precision' : make_scorer(precision_score, average='weighted', zero_division=1),
           'recall' : make_scorer(recall_score, average='weighted', zero_division=1),
           'f1_score' : make_scorer(f1_score, average='weighted', zero_division=1)}
```

Python

```python
# Evaluate different amount of training data
compare_amounts = [18, 38, 54, 70]
neigh = KNeighborsClassifier(n_neighbors=10)
logist = LogisticRegression(random_state=0, solver='lbfgs', max_iter=1000)
RF = RandomForestClassifier(random_state=0)
AdaBoost = AdaBoostClassifier(random_state=0)
models = [(RF,'RandomForest'),
        (AdaBoost, 'AdaBoost')]

for model, model_name in models:
    print(f"{model_name}")
    kfold = KFold(n_splits=10, random_state=0, shuffle=True)
    for compare_amount in compare_amounts:
        X_train_subset = X_train[:compare_amount]
        y_train_subset = y_train[:compare_amount]
        cv_results = cross_validate(model, X_train_subset, y_train_subset, cv=kfold, scoring=scoring)
        print(f"data amount: {compare_amount}")
        print("acc\tprecision\trecall\tf1-score")
        print(f"{np.mean(cv_results['test_accuracy'])}\t{np.mean(cv_results['test_precision'])}\t{np.mean(cv_r
        print("-------------------------")
```

```python
# evaluate different models

neigh = KNeighborsClassifier(n_neighbors=10)
logist = LogisticRegression(random_state=0, solver='lbfgs', max_iter=1000)
RF = RandomForestClassifier(random_state=0)
AdaBoost = AdaBoostClassifier(random_state=0)
# models = [(neigh,'K-NN'),
#         (logist,'Logistic Regression')]
models = [(RF,'RandomForest'),
        (AdaBoost, 'AdaBoost')]
for model, model_name in models:
    print(f"model: {model_name}")
    predictions, labels = evaluate_model(X_train, y_train, neigh)
    plot_confusion_matrix(predictions, labels, class_names)
    report(predictions, labels)
```

Python

```python
# Evaluate different K values for K-NN
k_range = list(range(26))[1:]

for k in k_range:
    kfold = KFold(n_splits=10, random_state=0, shuffle=True)
    neigh = KNeighborsClassifier(n_neighbors=k)
    cv_results = cross_validate(neigh, X_train, y_train, cv=kfold, scoring=scoring)
    print(f'k = {k}')
    print("acc\t\t\tprecision\t\trecall\t\t\tf1-score")
    print(f"{np.mean(cv_results['test_accuracy'])}\t{np.mean(cv_results['test_precision'])}\t{np.mean(cv_resul
    print("-------------------------")
```

```python
# Evaluate different solver for Logistic Regression
solvers = ['newton-cg', 'sag','saga', 'lbfgs']

for solver in solvers:
    kfold = KFold(n_splits=10, random_state=0, shuffle=True)
    logist = LogisticRegression(random_state=0, solver=solver, max_iter=1000)
    cv_results = cross_validate(logist, X_train, y_train, cv=kfold, scoring=scoring)
    print(f'solver = {solver}')
    print("acc\t\t\tprecision\t\trecall\t\t\tf1-score")
    print(f"{np.mean(cv_results['test_accuracy'])}\t{np.mean(cv_results['test_precision'])}\t{np.mean(cv_resu
    print("---------------------------")
```

```python
# Evaluate different max_depth for Random Forest
max_depths = [2, 3, 4, 5, 6]

for max_depth in max_depths:
    kfold = KFold(n_splits=10, random_state=0, shuffle=True)
    RF = RandomForestClassifier(random_state=0, max_depth=max_depth)
    cv_results = cross_validate(RF, X_train, y_train, cv=kfold, scoring=scoring)
    print(f'max_depth = {max_depth}')
    print("acc\tprecision\trecall\tf1-score")
    print(f"{np.mean(cv_results['test_accuracy'])}\t{np.mean(cv_results['test_precision'])}\t{np.mean(cv_results['test_recall'])}
    print("-------------------------")
```

```python
# Evaluate different learning rate for Random Forest
learning_rates = [0.001, 0.005, 0.01 ,0.05, 0.1, 0.5, 1.0]

for learning_rate in learning_rates:
    kfold = KFold(n_splits=10, random_state=0, shuffle=True)
    AdaBoost = AdaBoostClassifier(random_state=0, learning_rate=learning_rate)
    cv_results = cross_validate(AdaBoost, X_train, y_train, cv=kfold, scoring=scoring)
    print(f'learning rate = {learning_rate}')
    print("acc\tprecision\trecall\tf1-score")
    print(f"{np.mean(cv_results['test_accuracy'])}\t{np.mean(cv_results['test_precision'])}\t{np.mean(cv_results['test_recall'])}
    print("-------------------------")
```