AI capstone project 1

109350008 張詠哲

Research question

Because I have taking a course about autonomous driving this semester. Therefore I want to do some traffic object image classification. And current research about autonomous driving and image classification are often use deep learning model, and the machine learning model are barely used. Therefore, the research question is the deep learning model are better than machine learning model in image classification? And beside the mention research question, I also want to try some data preprocessing and augmentation I've learn to make the result get better and compare them. I will take the traffic object image as dataset.

Dataset

I collaborate with 沈昱宏 to collect this dataset.

This dataset comprises 500 PNG images categorized into five classes for image classification related to traffic objects. I think this maybe applicable to self-driving car systems. Classes include double yellow lines, zebra crossings, traffic lights, buses, and a miscellaneous class labeled "nothing" representing non-classified traffic objects. Images were collected exclusively through screenshots from Google Maps, ensuring diverse locations. Each class contains 100 images cropped and resized to a uniform dimension of 64x64 pixels. No specific weather or lighting conditions were enforced during data collection. Manual labeling was performed to assign class labels to each image. There are some example:

Double yellow lines	Bus	Nothing	Traffic light	Zebra crossing

Algorithm

There is the description about supervise and unsupervised method.

Supervised Learning

Supervised learning means each input data point is associated with a corresponding target label or output. During training, the algorithm learns the mapping between input features and target labels by optimizing a predefined objective function.

Unsupervised Learning

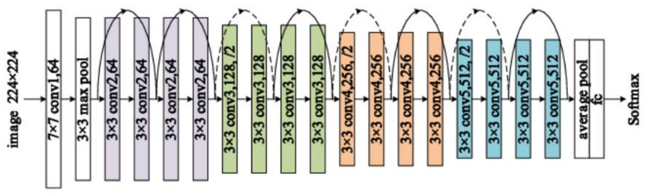
Unsupervised learning means there are no target labels or outputs provided during training. The algorithm explores the structure and patterns in the input data without explicit guidance, such as clustering similar data points together or discovering latent features.

For the research question I mention above and the project requirement. I choose 1 deep learning model and 2 machine learning models , ResNet (supervised) 、Randomforest (supervised) 、Kmean (unsupervised). And For the data preprocessing I will try 2 methods, which are HOG to extract feature and PCA to reduce the feature. Following are description about the above algorithm, and the reference I use.

ResNet

(reference: https://pytorch.org/vision/main/models/resnet.html)

ResNet is a deep learning architecture designed to address the vanishing gradient problem in very deep neural networks. To address this problem ResNet introduces the concept of residual learning, where instead of learning the desired mapping directly, the network learns residual functions. As a result, ResNet can effectively train very deep neural networks with hundreds or even thousands of layers, leading to improved performance on various computer vision tasks.



(ResNet18 architecture)

Random Forest

(reference:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

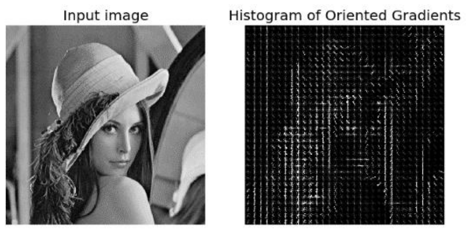
Random Forest is a machine learning algorithm. It belongs to the ensemble learning category. It refers to a collection of decision trees. Each decision tree is trained on a random subset of the training data and a random subset of features from the input data. During training, the algorithm constructs multiple decision trees with different random subsets, and predictions are made by aggregating the predictions of these trees (e.g., averaging for regression or voting for classification).

Kmeans
(reference:
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)

K-means is a unsupervised method. It's designed to partition a dataset into K distinct, non-overlapping clusters. The algorithm works by iteratively assigning each data point to the nearest cluster center and then recalculating the cluster centers based on the mean of the data points assigned to each cluster. This process continues until the cluster assignments stabilize.

HOG (Histogram of Oriented Gradients)
(reference:
https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_hog.html)

Histogram of Oriented Gradients (HOG) is a feature descriptor used in computer vision and image processing for object detection and recognition tasks. It works by capturing the distribution of gradient orientations in an image, providing a representation of the local structure and texture within the image.



Example figure of HOG

PCA (Principal components analysis)
(reference:
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)

Principal Component Analysis (PCA) is a dimensionality reduction technique commonly used in image processing and classification tasks. It aims to reduce the complexity of high-dimensional data while preserving the most important information and patterns.

Experiment

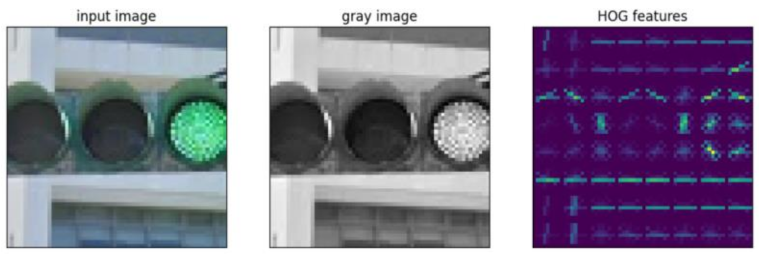
For experiment, I will implement the 2 supervised and 1 unsupervised method I mention above. For data part I will try to use some data preprocessing method, such HOG and PCA. And some data augmentation in ResNet50, such as normalize, resize, autoaugment (policy = CIFAR10). For weights and hyperparameter part. I will use pretrain weights in ResNet, and use cross validation to find the best hyperparameter in machine learning method to get the result better.. I also want to experiment and see if deep learning really outperforms machine learning in image classification.

Result

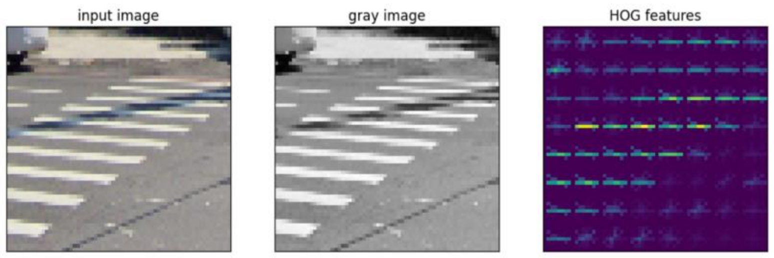
HOG result

Following figures are the HOG processing result visualization.

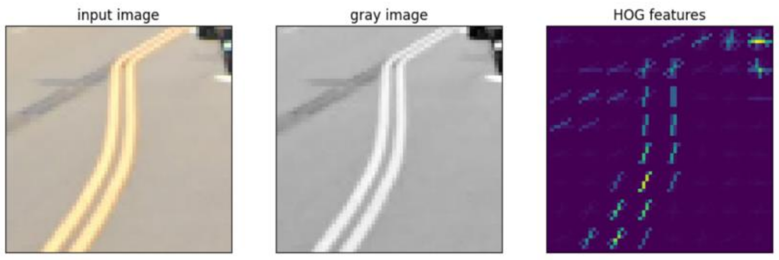
- **Traffic light**



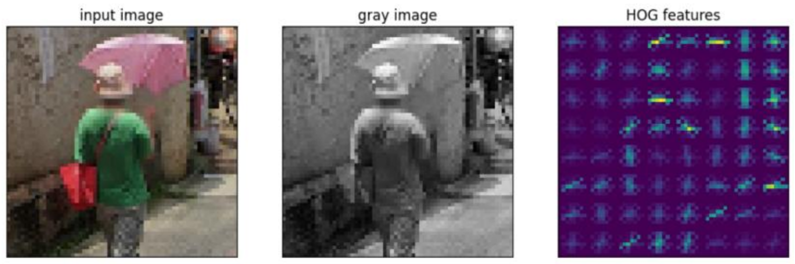
- **Zebra crossing**



- **Double yellow lines**



- **Nothing**



- **Bus**

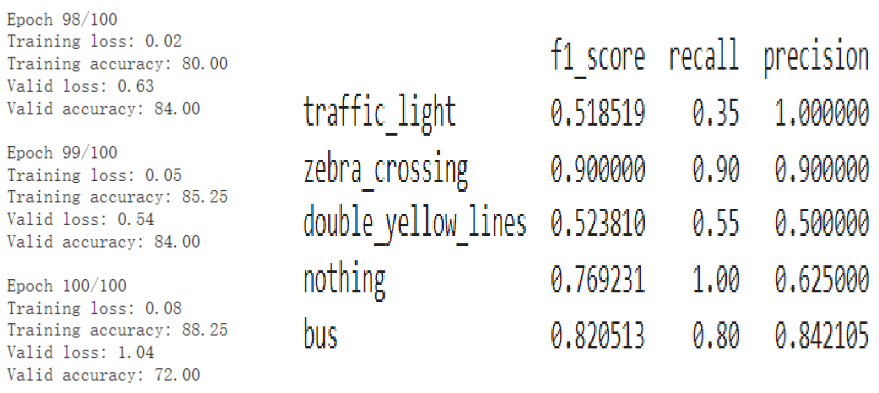
ResNet

- **Without pretrain**
  - Hyperparameter & optimizer

```
lr = 1e-4
weight_decay = 5e-4
epochs = 100
criterion = torch.nn.CrossEntropyLoss()
pretrain = False
optimizer = torch.optim.Adam
```

  - Training 100 epoch

Last epoch test metrics: accuracy = 72%, best epoch accuracy = 84%

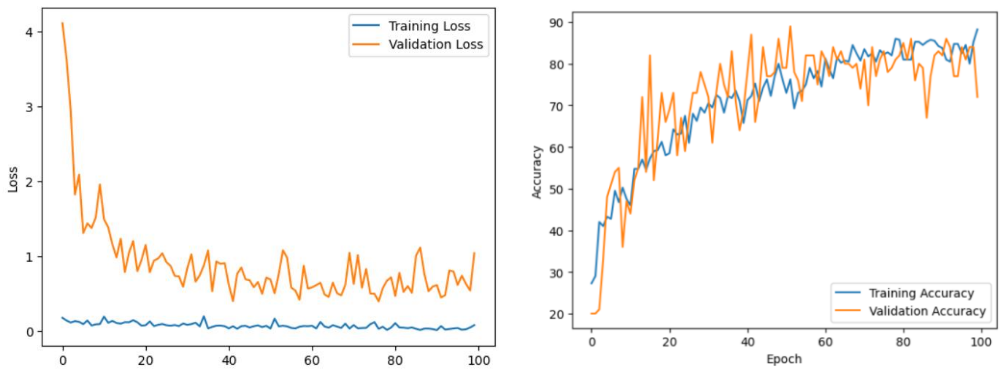


```
Epoch 98/100
Training loss: 0.02
Training accuracy: 80.00
Valid loss: 0.63
Valid accuracy: 84.00

Epoch 99/100
Training loss: 0.05
Training accuracy: 85.25
Valid loss: 0.54
Valid accuracy: 84.00

Epoch 100/100
Training loss: 0.08
Training accuracy: 88.25
Valid loss: 1.04
Valid accuracy: 72.00
```

	f1_score	recall	precision
traffic_light	0.518519	0.35	1.000000
zebra_crossing	0.900000	0.90	0.900000
double_yellow_lines	0.523810	0.55	0.500000
nothing	0.769231	1.00	0.625000
bus	0.820513	0.80	0.842105



- **With pretrain weights**

(https://download.pytorch.org/models/resnet50-0676ba61.pth)

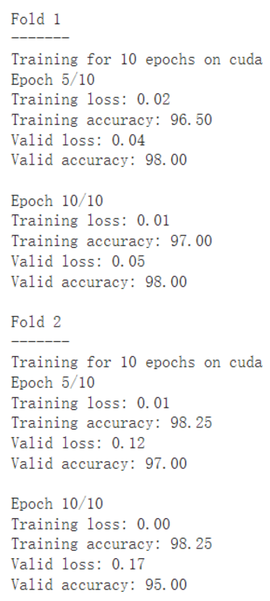
For this part I only use accuracy as metric. I also use 5-fold cross validation in this part (because it can get great result in few epoch).

- Hyperparameter & optimizer

```
lr = 1e-4
weight_decay = 5e-4
epochs = 10
criterion = torch.nn.CrossEntropyLoss()
pretrain = True
optimizer = torch.optim.Adam
```

- Train 10 epoch each fold

  Best accuracy on test data: 98%



```
Fold 1
-------
Training for 10 epochs on cuda
Epoch 5/10
Training loss: 0.02
Training accuracy: 96.50
Valid loss: 0.04
Valid accuracy: 98.00

Epoch 10/10
Training loss: 0.01
Training accuracy: 97.00
Valid loss: 0.05
Valid accuracy: 98.00

Fold 2
-------
Training for 10 epochs on cuda
Epoch 5/10
Training loss: 0.01
Training accuracy: 98.25
Valid loss: 0.12
Valid accuracy: 97.00

Epoch 10/10
Training loss: 0.00
Training accuracy: 98.25
Valid loss: 0.17
Valid accuracy: 95.00
```

- **Analysis**

This result quiet intuition. The highest accuracy achieved without using pre-trained weights with ResNet was 84%, whereas with pre-trained weights, the accuracy reached as high as 98%. But as the training and test accuracy and loss curve present in without pretrain. It seems some unstable behavior. I think it possibly due to a small batch size or a large learning rate.

Random Forest

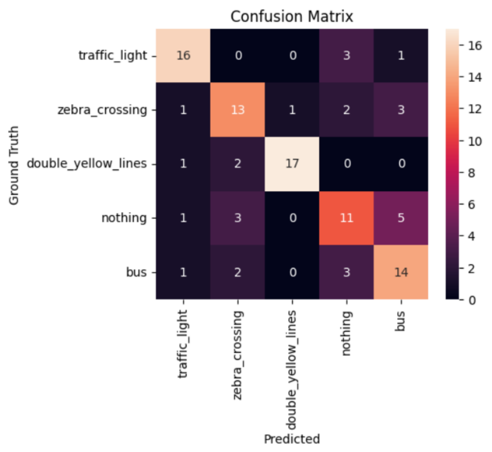
- **Origin data**
  - Hyperparameter

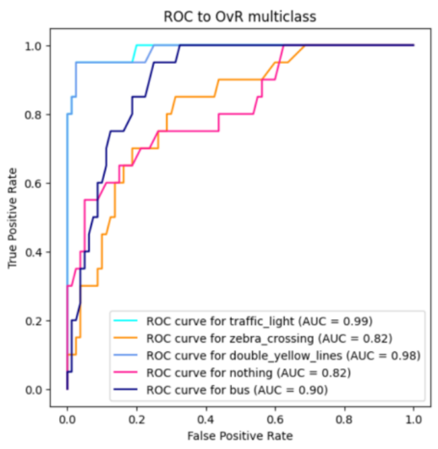
```
RandomForestClassifier(max_depth= 50, min_samples_leaf=1,
min_samples_split=2, n_estimators=250)
```

■ Result

```
The accuracy is 71.0

                   f1_score  recall  precision
traffic_light      0.800000    0.80   0.800000
zebra_crossing     0.650000    0.65   0.650000
double_yellow_lines 0.894737   0.85   0.944444
nothing            0.564103    0.55   0.578947
bus                0.651163    0.70   0.608696
```





● PCA

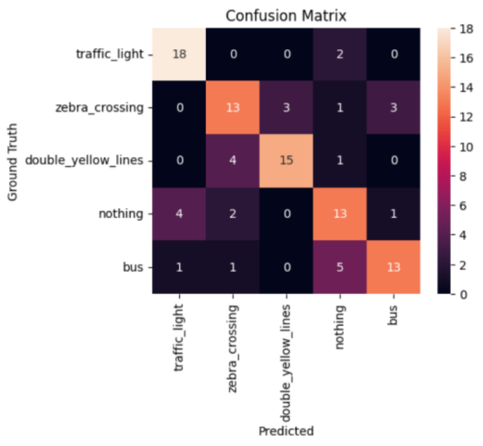
■ Hyperparameter

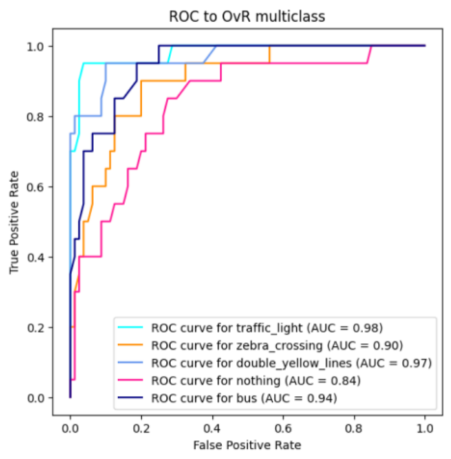
```
RandomForestClassifier(max_depth= 30, min_samples_leaf=1,
min_samples_split=2, n_estimators=350)
```

■ Result

```
The accuracy is 72.0

                    f1_score  recall  precision
traffic_light       0.837209    0.90   0.782609
zebra_crossing      0.650000    0.65   0.650000
double_yellow_lines 0.789474    0.75   0.833333
nothing             0.619048    0.65   0.590909
bus                 0.702703    0.65   0.764706
```





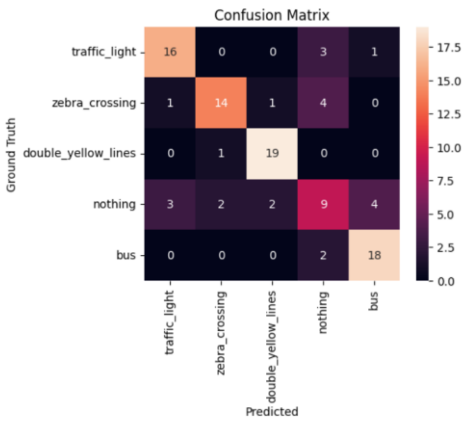
- **HOG**
  - Hyperparameter

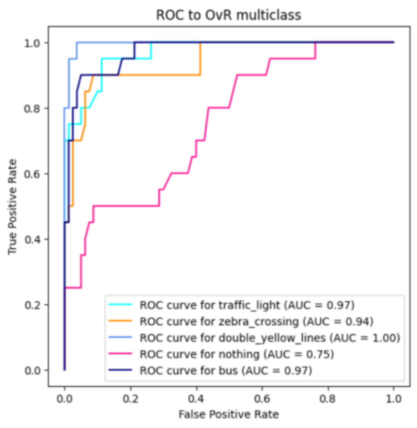
```
RandomForestClassifier(max_depth= 50, min_samples_leaf=1,
min_samples_split=2, n_estimators=450)
```

  - Result

```
The accuracy is 76.0

                     f1_score  recall  precision
traffic_light        0.800000    0.80   0.800000
zebra_crossing       0.756757    0.70   0.823529
double_yellow_lines  0.904762    0.95   0.863636
nothing              0.473684    0.45   0.500000
bus                  0.837209    0.90   0.782609
```





- **Analysis**

  The results revealed varying levels of accuracy. Using the original data yielded an accuracy of 71%, indicating a decent performance without preprocessing. Introducing PCA slightly improved the accuracy to 72%, showcasing the benefit of dimensionality reduction in capturing more relevant features. However, the most significant improvement came with the HOG preprocessing, achieving an accuracy of 76%. This substantial increase suggests that HOG's feature extraction capability, focusing on local structure and texture information in images, greatly enhanced the Random Forest model's performance.

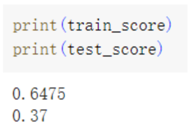
Kmeans

In this part I only use accuracy as metric.

- **Origin data**
  - Hyperparameter

```
KMeans(n_clusters = 120)
```

- Result



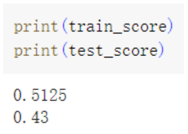
```
print(train_score)
print(test_score)

0.6475
0.37
```

- **PCA**
  - Hyperparameter

```
KMeans(n_clusters = 20)
```

  - Result



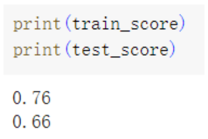
```
print(train_score)
print(test_score)

0.5125
0.43
```

- **HOG**
  - Hyperparameter

```
KMeans(algorithm = "elkan", n_clusters = 70)
```

  - Result



```
print(train_score)
print(test_score)

0.76
0.66
```

- **Analysis**

  After conducting experiments using K-means clustering with different data preprocessing methods, significant variations in accuracy were observed. The initial accuracy with the original data stood at 37%, indicating challenges in effectively separating and classifying data points without preprocessing. And PCA led to a moderate improvement in accuracy to 43%, showcasing the benefits of dimensionality reduction in performance. However, HOG result in an impressive accuracy of 66%. HOG's ability to capture local structure and texture information in images may played a crucial role in facilitating better classification.

Discussion

Accuracy (%)	ResNet	Random Forest	Kmeans
Origin	84	71	37
PCA		72	43
HOG		76	66
pretrain	98		

The table above summarizes the accuracy results of all test sets. As the table shows I want to discuss in 3 part: model (random forest vs Kmeans), data processing (Origin vs HOG vs PCA), DL vs ML (ResNet vs RandomForst & Kmeans).

In the table shows that random Forest outperforms K-means in every condition. I think is due to its ability to capture non-linear relationships. Random Forest is non-linear relationships between input features and output labels. It combines multiple decision trees, each trained on a random subset of data and features, resulting in more accurate and robust predictions through a voting mechanism. In contrast to K-means , which is a cluster algorithm. Maybe is more suitable in image segmentation.

In both random forest and kmeans HOG outperforms PCA . I think maybe in image classification due to HOG focused feature extraction, capturing local structure and texture information critical for distinguishing between objects or regions. HOG's discriminative power and robustness to variations and noise make it more effective in creating distinct and separable clusters, leading to improved classification accuracy. Additionally, HOG's domain-specific design tailored for image processing tasks contributes to its superior performance compared to PCA, which may not always capture the most relevant features for image classification.

ResNet outperform traditional machine learning methods such as Random Forest and K-means in image classification. I think it's due to their capacity for hierarchical feature learning, end-to-end learning from raw data, non-linearity and flexibility in capturing complex relationships, superior representation learning capabilities. These factors collectively enable deep learning models to automatically extract intricate patterns and meaningful representations from data, leading to higher accuracy and better generalization compared to traditional machine learning approaches. But the disadvantage is DL model needs huge data and lots of calculation time and resource to get high performanse like 98% accuracy (pretrain).

Reference:

https://pytorch.org/vision/main/models/resnet.html
https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
https://ithelp.ithome.com.tw/articles/10206249
https://www.kaggle.com/code/akhileshrai/pca-for-visualisation-classification
https://medium.com/@joel_34096/k-means-clustering-for-image-classification-a648f28bdc47
https://saturncloud.io/blog/how-to-use-kfold-cross-validation-with-dataloaders-in-pytorch/

Appendix

I push it on github too (https://github.com/zyz-2299mod10/AIcap/tree/main).

ResNet

```
"""AIcapstone-P1-ResNet.ipynb
# Package
"""

import torchvision
import torch
import os
import pandas as pd
import numpy as np

from sklearn.model_selection import KFold

import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score

from google.colab import drive
drive.mount('/content/drive')

"""# DataLoader"""

class PrepareDataLoader():
    def __init__(self, train_path, test_path):
        self.train_path = train_path
        self.test_path = test_path

    def calculate_mean_std(self):
        train_dataset = torchvision.datasets.ImageFolder(
        self.train_path,
        transform=torchvision.transforms.Compose([
            # Resize step is required as we will use a ResNet model, which accepts at
leats 224x224 images
            torchvision.transforms.Resize((224,224)),
            torchvision.transforms.ToTensor(),
          ])
        )
        train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=32, 
shuffle=False, num_workers=2, pin_memory=True)

        means = []
        stdevs = []
        for X, _ in train_dataloader:
            # Dimensions 0,2,3 are respectively the batch, height and width dimensions
            means.append(X.mean(dim=(0,2,3)))
            stdevs.append(X.std(dim=(0,2,3)))
        mean = torch.stack(means, dim=0).mean(dim=0)
        stdev = torch.stack(stdevs, dim=0).mean(dim=0)

        return mean, stdev

    def get_dataset(self):
        mean, stdev = self.calculate_mean_std()

        train_transforms = torchvision.transforms.Compose([
                torchvision.transforms.Resize((224,224)),

torchvision.transforms.AutoAugment(policy=torchvision.transforms.AutoAugmentPo
licy.CIFAR10),
                torchvision.transforms.ToTensor(),
                torchvision.transforms.Normalize(mean, stdev)
            ])



        test_transforms = torchvision.transforms.Compose([
                torchvision.transforms.Resize((224,224)),
                torchvision.transforms.ToTensor(),
                torchvision.transforms.Normalize(mean, stdev)
            ])

        train_dataset = torchvision.datasets.ImageFolder(self.train_path, 
transform=train_transforms)
```

```
        test_dataset = torchvision.datasets.ImageFolder(self.test_path,
transform=test_transforms)

        return train_dataset, test_dataset

    def prepare_dataloader(self, get_dataset = False):
        train_dataset, test_dataset = self.get_dataset()

        train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=32,
shuffle=True, num_workers=0, pin_memory=True)
        test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=32,
shuffle=False, num_workers=0, pin_memory=True)


        return train_dataloader, test_dataloader

DataLoader = PrepareDataLoader(train_path='/content/drive/MyDrive/dataset/train',
test_path='/content/drive/MyDrive/dataset/test')

train_dataloader, test_dataloader = DataLoader.prepare_dataloader()

"""# Build model"""

class_name_path = "/content/drive/MyDrive/dataset/train"
ClassName = os.listdir(class_name_path)

def get_net(pretrain = False):
    if pretrain:
        resnet =
torchvision.models.resnet50(weights="ResNet50_Weights.IMAGENET1K_V1")
    else:
        resnet = torchvision.models.resnet50(weights=None)

    # Substitute the FC output layer
    resnet.fc = torch.nn.Linear(resnet.fc.in_features, len(ClassName))
    torch.nn.init.xavier_uniform_(resnet.fc.weight)
    return resnet
```

```
def train(net, train_dataloader, valid_dataloader, criterion, optimizer, scheduler=None,
epochs=10, device='cpu', checkpoint_epochs=10, is_cv = False):
    print(f'Training for {epochs} epochs on {device}')
    train_losses = []
    train_accuracies = []
    val_losses = []
    val_accuracies = []

    for epoch in range(1,epochs+1):
        net.train()    # put network in train mode for Dropout and Batch
Normalization
        train_loss = torch.tensor(0., device=device)    # loss and accuracy tensors are
on the GPU to avoid data transfers
        train_accuracy = torch.tensor(0., device=device)
        for X, y in train_dataloader:
            X = X.to(device)
            y = y.to(device)
            preds = net(X)
            train_loss = criterion(preds, y)

            optimizer.zero_grad()
            train_loss.backward()
            optimizer.step()

            with torch.no_grad():
                train_loss += train_loss * train_dataloader.batch_size
                train_accuracy += (torch.argmax(preds, dim=1) == y).sum()

        train_losses.append((train_loss/len(train_dataloader.dataset)).item())

train_accuracies.append((100*train_accuracy/len(train_dataloader.dataset)).item())

        if valid_dataloader is not None:
            net.eval()    # put network in train mode for Dropout and Batch
Normalization
            valid_loss = torch.tensor(0., device=device)
            valid_accuracy = torch.tensor(0., device=device)
            with torch.no_grad():
                for X, y in valid_dataloader:
                    X = X.to(device)
                    y = y.to(device)
                    preds = net(X)
                    val_loss = criterion(preds, y)

                    valid_loss += val_loss * valid_dataloader.batch_size
                    valid_accuracy += (torch.argmax(preds, dim=1) == y).sum()

            val_losses.append((valid_loss/len(valid_dataloader.dataset)).item())

val_accuracies.append((100*valid_accuracy/len(valid_dataloader.dataset)).item())

            if scheduler is not None:
                scheduler.step()

            # saving & print result
            if is_cv: # cross validation
              if epoch%checkpoint_epochs==0:
                print(f"Epoch {epoch}/{epochs}")
                print(f'Training loss: {train_loss/len(train_dataloader.dataset):.2f}')
                print(f'Training                                        accuracy:
{100*train_accuracy/len(train_dataloader.dataset):.2f}')
                print(f'Valid loss: {valid_loss/len(valid_dataloader.dataset):.2f}')
                print(f'Valid                                           accuracy:
{100*valid_accuracy/len(valid_dataloader.dataset):.2f}')
                print()

            else: # training
              print(f"Epoch {epoch}/{epochs}")
              print(f'Training loss: {train_loss/len(train_dataloader.dataset):.2f}')
              print(f'Training                                          accuracy:
{100*train_accuracy/len(train_dataloader.dataset):.2f}')

              if valid_dataloader is not None:
                  print(f'Valid loss: {valid_loss/len(valid_dataloader.dataset):.2f}')
                  print(f'Valid                                         accuracy:
{100*valid_accuracy/len(valid_dataloader.dataset):.2f}')
```

```
            print()

            if epoch%checkpoint_epochs==0:
               torch.save({
                     'epoch': epoch,
                     'state_dict': net.state_dict(),
                     'optimizer': optimizer.state_dict(),
               }, './checkpoint.resnet50_.pth')

      return net, train_losses, train_accuracies, val_losses, val_accuracies

"""# Train

### Hyperparameter
"""

lr = 1e-4
weight_decay = 5e-4
epochs = 100
criterion = torch.nn.CrossEntropyLoss()
pretrain = False

# params_1x are the parameters of the network body, i.e., of all layers except the FC layers
device = 'cuda' if torch.cuda.is_available() else 'cpu'
net = get_net(pretrain=pretrain).to(device)
params_1x = [param for name, param in net.named_parameters() if 'fc' not in str(name)]
optimizer = torch.optim.Adam([{'params':params_1x}, {'params': net.fc.parameters(), 'lr': lr*10}], lr = lr, weight_decay=weight_decay)

"""### training"""

net, train_losses, train_accuracies, test_losses, test_accuracies = train(net, train_dataloader, test_dataloader, criterion, optimizer, None, epochs, device, is_cv = False)

"""### visualize"""
```

```
import matplotlib.pyplot as plt

class Result():
    def __init__(self, y_pred, y_test, ClassName):
        self.y_pred = y_pred
        self.y_test = y_test
        self.ClassName = ClassName

    def get_accuracy(self):
        return accuracy_score(self.y_pred, self.y_test)*100

    def get_cm(self):
        return confusion_matrix(self.y_test, self.y_pred)

    def plot_cm(self):
        cm = self.get_cm()
        cm_df = pd.DataFrame(cm, index = self.ClassName, columns = self.ClassName)

        plt.figure(figsize=(5,4))
        sns.heatmap(cm_df, annot=True)
        plt.title('Confusion Matrix')
        plt.ylabel('Ground Truth')
        plt.xlabel('Predicted')
        plt.show()

    def get_recall_f1score_precision(self):
        f1_Score = f1_score(self.y_test, self.y_pred, average = None).reshape(len(ClassName), -1)
        recall = recall_score(self.y_test, self.y_pred, average = None).reshape(len(ClassName), -1)
        precision = precision_score(self.y_test, self.y_pred, average = None).reshape(len(ClassName), -1)

        result = np.concatenate((f1_Score, recall, precision), axis = 1)
        df_result = pd.DataFrame(result)
        df_result.columns = ["f1_score", "recall", "precision"]
        df_result.index = self.ClassName
        print(df_result)
    def plot_every_metrix(self):
        print(f"The accuracy is {self.get_accuracy()}")
        print()

        self.get_recall_f1score_precision()
        print()

        self.plot_cm()
        print()

def plot_loss(train_losses, test_losses):
        plt.figure()
        plt.plot(range(len(train_losses)), train_losses, label='Training Loss')
        plt.plot(range(len(test_losses)), test_losses, label='Validation Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()

def plot_accuracy(train_accuracies, test_accuracies):
        plt.figure()
        plt.plot(range(len(train_accuracies)), train_accuracies, label='Training Accuracy')
        plt.plot(range(len(test_accuracies)), test_accuracies, label='Validation Accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.legend()
        plt.show()

plot_loss(train_losses, test_losses)
plot_accuracy(train_accuracies, test_accuracies)

model = get_net(pretrain=pretrain).to(device)
model.load_state_dict(torch.load("/content/checkpoint.resnet50_.pth")['state_dict'])

y_test = []
y_pred = []
```

```
model.eval()
with torch.no_grad():
    for x, y in test_dataloader:
        x = x.to(device)
        preds = model(x)
        pred = torch.argmax(preds, dim=1)

        y_test += y.tolist()
        y_pred += pred.tolist()

result = Result(y_pred, y_test, ClassName)
result.plot_every_metrix()

"""# Cross Validation"""

DataLoader = PrepareDataLoader(train_path='/content/drive/MyDrive/dataset/train',
test_path='/content/drive/MyDrive/dataset/test')

train_dataset, test_dataset = DataLoader.get_dataset()

"""### hyperparameter"""

lr = 1e-4
weight_decay = 5e-4
epochs = 10
criterion = torch.nn.CrossEntropyLoss()
pretrain = True

# params_1x are the parameters of the network body, i.e., of all layers except the FC
layers
device = 'cuda' if torch.cuda.is_available() else 'cpu'
net = get_net(pretrain=pretrain).to(device)
params_1x = [param for name, param in net.named_parameters() if 'fc' not in str(name)]
optimizer = torch.optim.Adam([{'params':params_1x}, {'params': net.fc.parameters(),
'lr': lr*10}], lr = lr, weight_decay=weight_decay)

"""### cross validation"""
```

```
k_folds = 5
batch_size = 32
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

kf = KFold(n_splits=k_folds, shuffle=True)
for fold, (train_idx, test_idx) in enumerate(kf.split(train_dataset)):
    print(f"Fold {fold + 1}")
    print("-------")

    # Define the data loaders for the current fold
    train_loader = torch.utils.data.DataLoader(
        dataset=train_dataset,
        batch_size=batch_size,
        sampler=torch.utils.data.SubsetRandomSampler(train_idx),
    )
    test_loader = torch.utils.data.DataLoader(
        dataset=train_dataset,
        batch_size=batch_size,
        sampler=torch.utils.data.SubsetRandomSampler(test_idx),
    )

    net, train_losses, train_accuracies, test_losses, test_accuracies = train(net,
train_dataloader, test_dataloader, criterion, optimizer, None, epochs, device,
checkpoint_epochs = 5, is_cv=True)
```

Kmeans

```
"""AIcapstone-P1-Kmeans.ipynb
Original file is located at
    https://colab.research.google.com/drive/1Ixeu-LS_9hE0RoR_u-
HNFAr_4jiVbKRK

# Package
"""

import pandas as pd
import os
from skimage.transform import resize
from skimage.io import imread
```

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import color

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA
from skimage.feature import hog

from sklearn.metrics import accuracy_score

from google.colab import drive
drive.mount('/content/drive')

"""# Process Data"""

class_name_path = "/content/drive/MyDrive/dataset/train"
ClassName = os.listdir(class_name_path)

flat_data_arr_train=[] # training input array
flat_data_arr_test=[] # test input array

target_arr_train=[] # train output array
target_arr_test=[] # test output array

datadir="/content/drive/MyDrive/dataset/"
# path which contains all the categories of images
for i in ClassName:
    print(f'loading... category : {i}')
    # training
    training_path=os.path.join(datadir, 'train', i)
    for img in os.listdir(training_path):
        img_array=imread(os.path.join(training_path,img))
        flat_data_arr_train.append(img_array.flatten())
        target_arr_train.append(ClassName.index(i))

    # test
    test_path=os.path.join(datadir, 'test', i)
    for img in os.listdir(test_path):
        img_array=imread(os.path.join(test_path,img))
        flat_data_arr_test.append(img_array.flatten())
        target_arr_test.append(ClassName.index(i))
    print(f'loaded category:{i} successfully')
    print()

flat_data_train=np.array(flat_data_arr_train)
target_train=np.array(target_arr_train)
df_train=pd.DataFrame(flat_data_train) #dataframe
df_train['Target']=target_train
x_train=df_train.iloc[:,:-1] # training input data
y_train=df_train.iloc[:,-1] # training output data

flat_data_test=np.array(flat_data_arr_test)
target_test=np.array(target_arr_test)
df_test=pd.DataFrame(flat_data_test)
df_test["Target"]=target_test
x_test=df_test.iloc[:,:-1]
y_test=df_test.iloc[:,-1]

# print(df_train)
# print(df_test)

"""# common function"""

def retrieve_info(cluster_labels,y_train):
    # Initializing
    reference_labels = {}
    # For loop to run through each label of cluster label
    for i in range(len(np.unique(cluster_labels))):
        index = np.where(cluster_labels==i, 1, 0)
        num = np.bincount(y_train[index==1]).argmax()
        reference_labels[i] = num

    return reference_labels

def compute_kmeans_score(model, kmean_labels, y_train, x_test, y_test):
    # get reference label
    reference_labels = retrieve_info(kmeans.labels_,y_train)

    # training score
    train_labels = np.random.rand(len(kmeans.labels_))
    for i in range(len(kmeans.labels_)):
        train_labels[i] = reference_labels[kmeans.labels_[i]]

    train_score = accuracy_score(train_labels, y_train)

    # test score
    pre = kmeans.predict(x_test)
    test_labels = []
    for i in pre:
        test_labels.append(reference_labels[i])

    test_labels = np.array(test_labels)
    test_score = accuracy_score(test_labels,y_test)

    return train_score, test_score

def cross_validation(model, x_train, y_train, cv = 5):

    cvs = 0
    record = []
    num_val_samples = len(x_train)//cv

    for i in range(cv):
        val_data = x_train[i*num_val_samples : (i+1)*num_val_samples]
        val_targets = y_train[i*num_val_samples : (i+1)*num_val_samples]

        remaining_data = np.concatenate(
                                  [x_train[: i*num_val_samples],
                                  x_train[(i+1)*num_val_samples :]],
                                  axis = 0)
        remaining_targets = np.concatenate(
                                  [y_train[: i*num_val_samples],
                                  y_train[(i+1)*num_val_samples :]],
                                        axis = 0)
            # print(i)
            # print(remaining_data.shape)
            # print(remaining_targets.shape)
            model.fit(remaining_data, remaining_targets)
            train_score, test_score = compute_kmeans_score(model, model.labels_,
                                                     remaining_targets, val_data, val_targets)
            record.append(test_score)

    for i in record:
        cvs += (i/cv)

    return test_score

"""# origion"""

k = range(10, 350, 10)
llo_score = []
elk_score = []
best_score = 0
best_k = 0
best_algorithm = ""

for a in ["lloyd", "elkan"]:
    for i in k:
        kmeans = KMeans(algorithm = a, n_init = 'auto', n_clusters = i)
        now_score = cross_validation(kmeans, x_train, y_train, cv = 5)
        if(a == "lloyd"): llo_score.append(now_score)
        else: elk_score.append(now_score)

        if(now_score > best_score):
            best_algorithm = a
            best_score = now_score
            best_k = i

print(f"best_algorithm: {best_algorithm}")
print(f"best_k: {best_k}")
print(f"best_score: {best_score}")
```

```
plt.plot()
lloyd, = plt.plot(k, llo_score, label = 'lloyd')
elkan, = plt.plot(k, elk_score, label = 'elkan')
plt.xlabel('k')
plt.ylabel('mean_score')
plt.legend(handles = [lloyd, elkan], loc='upper right')
plt.show()

kmeans = KMeans(n_clusters = 120)
kmeans.fit(x_train, y_train)

train_score, test_score = compute_kmeans_score(kmeans, kmeans.labels_, y_train,
x_test, y_test)

print(train_score)
print(test_score)

"""# PCA"""

pca = PCA(n_components=0.85)
pca.fit(x_train)

x_train_pca = pca.transform(x_train)
x_test_pca = pca.transform(x_test)

k = range(10, 350, 10)
llo_score = []
elk_score = []
best_score = 0
best_k = 0
best_algorithm = ""

for a in ["lloyd", "elkan"]:
    for i in k:
        kmeans = KMeans(algorithm = a, n_init = 'auto', n_clusters = i)
        now_score = cross_validation(kmeans, x_train_pca, y_train, cv = 5)
        if(a == "lloyd"): llo_score.append(now_score)
        else: elk_score.append(now_score)

        if(now_score > best_score):
            best_algorithm = a
            best_score = now_score
            best_k = i

print(f"best_algorithm: {best_algorithm}")
print(f"best_k: {best_k}")
print(f"best_score: {best_score}")

plt.plot()
lloyd, = plt.plot(k, llo_score, label = 'lloyd')
elkan, = plt.plot(k, elk_score, label = 'elkan')
plt.xlabel('k')
plt.ylabel('mean_score')
plt.legend(handles = [lloyd, elkan], loc='upper right')
plt.show()

kmeans = KMeans(n_clusters = 20)
kmeans.fit(x_train_pca, y_train)

train_score, test_score = compute_kmeans_score(kmeans, kmeans.labels_, y_train,
x_test_pca, y_test)

print(train_score)
print(test_score)

"""# HOG"""

imgs = []
gray_img = []
hog_img = []
hog_feature_train = []
hog_feature_test = []
target_arr_train=[] # train output array
target_arr_test=[] # test output array
```

```
datadir="/content/drive/MyDrive/dataset/"
# path which contains all the categories of images
for i in ClassName:
    print(f'loading... category : {i}')
    # training
    training_path=os.path.join(datadir, 'train', i)
    for img_ in os.listdir(training_path):
        img = imread(os.path.join(training_path,img_))
        img_gray = color.rgb2gray(img)
        hog_vec, hog_vis = hog(img_gray, visualize=True)

        imgs.append(img)
        gray_img.append(img_gray)
        hog_img.append(hog_vis)
        hog_feature_train.append(hog_vec)
        target_arr_train.append(ClassName.index(i))

    # test
    test_path=os.path.join(datadir, 'test', i)
    for img_ in os.listdir(test_path):
        img = imread(os.path.join(test_path,img_))
        img_gray = color.rgb2gray(img)
        hog_vec, hog_vis = hog(img_gray, visualize=True)

        imgs.append(img)
        gray_img.append(img_gray)
        hog_img.append(hog_vis)
        hog_feature_test.append(hog_vec)
        target_arr_test.append(ClassName.index(i))
    print(f'loaded category:{i} successfully')
    print()

hog_feature_train = np.array(hog_feature_train)
target_train = np.array(target_arr_train)
df_train_hog = pd.DataFrame(hog_feature_train) #dataframe
df_train_hog['Target'] = target_train
x_train_hog=df_train_hog.iloc[:,:-1] # training input data
y_train=df_train_hog.iloc[:,-1] # training output data
```

```
hog_feature_test = np.array(hog_feature_test)
target_test=np.array(target_arr_test)
df_test_hog=pd.DataFrame(hog_feature_test)
df_test_hog["Target"]=target_test
x_test_hog=df_test_hog.iloc[:,:-1]
y_test=df_test_hog.iloc[:,-1]

k = range(10, 350, 10)
llo_score = []
elk_score = []
best_score = 0
best_k = 0
best_algorithm = ""

for a in ["lloyd", "elkan"]:
    for i in k:
        kmeans = KMeans(algorithm = a, n_init = 'auto', n_clusters = i)
        now_score = cross_validation(kmeans, x_train_hog, y_train, cv = 5)
        if(a == "lloyd"): llo_score.append(now_score)
        else: elk_score.append(now_score)

        if(now_score > best_score):
            best_algorithm = a
            best_score = now_score
            best_k = i

print(f"best_algorithm: {best_algorithm}")
print(f"best_k: {best_k}")
print(f"best_score: {best_score}")

plt.plot()
lloyd, = plt.plot(k, llo_score, label = 'lloyd')
elkan, = plt.plot(k, elk_score, label = 'elkan')
plt.xlabel('k')
plt.ylabel('mean_score')
plt.legend(handles = [lloyd, elkan], loc='upper right')
plt.show()
```

```
kmeans = KMeans(algorithm = "elkan", n_clusters = 70)
kmeans.fit(x_train_hog, y_train)

train_score, test_score = compute_kmeans_score(kmeans, kmeans.labels_, y_train,
x_test_hog, y_test)

print(train_score)
print(test_score)

"""# HOG + PCA"""

pca = PCA(n_components=0.85)
pca.fit(x_train_hog)

x_train_hog_pca = pca.transform(x_train_hog)
x_test_hog_pca = pca.transform(x_test_hog)

k = range(10, 350, 10)
llo_score = []
elk_score = []
best_score = 0
best_k = 0
best_algorithm = ""

for a in ["lloyd", "elkan"]:
  for i in k:
    kmeans = KMeans(algorithm = a, n_init = 'auto', n_clusters = i)
    now_score = cross_validation(kmeans, x_train_hog_pca, y_train, cv = 5)
    if(a == "lloyd"): llo_score.append(now_score)
    else: elk_score.append(now_score)

    if(now_score > best_score):
      best_algorithm = a
      best_score = now_score
      best_k = i

print(f"best_algorithm: {best_algorithm}")
print(f"best_k: {best_k}")
print(f"best_score: {best_score}")

plt.plot()
lloyd, = plt.plot(k, llo_score, label = 'lloyd')
elkan, = plt.plot(k, elk_score, label = 'elkan')
plt.xlabel('k')
plt.ylabel('mean_score')
plt.legend(handles = [lloyd, elkan], loc='upper right')
plt.show()

kmeans = KMeans(algorithm = "elkan", n_clusters = 60)
kmeans.fit(x_train_hog_pca, y_train)

train_score, test_score = compute_kmeans_score(kmeans, kmeans.labels_, y_train,
x_test_hog_pca, y_test)

print(train_score)
print(test_score)
```

RandomForest

```
"""AIcapstone-P1-RF.ipynb

# Package
"""

import pandas as pd
import os
from skimage.transform import resize
from skimage.io import imread
import numpy as np
import matplotlib.pyplot as plt
import math

from skimage import color
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from skimage.feature import hog
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelBinarizer
from itertools import cycle
from sklearn.metrics import auc, roc_curve, accuracy_score, f1_score, recall_score,
precision_score
from sklearn.metrics import RocCurveDisplay

from google.colab import drive
drive.mount('/content/drive')

"""# Process Data"""

class_name_path = "/content/drive/MyDrive/dataset/train"
ClassName = os.listdir(class_name_path)

flat_data_arr_train=[] # training input array
flat_data_arr_test=[] # test input array

target_arr_train=[] # train output array
target_arr_test=[] # test output array

datadir="/content/drive/MyDrive/dataset/"
# path which contains all the categories of images
for i in ClassName:
    print(f'loading... category : {i}')
    # training
    training_path=os.path.join(datadir, 'train', i)
    for img in os.listdir(training_path):
        img_array=imread(os.path.join(training_path,img))
        flat_data_arr_train.append(img_array.flatten())
        target_arr_train.append(ClassName.index(i))
    # test
    test_path=os.path.join(datadir, 'test', i)
    for img in os.listdir(test_path):
        img_array=imread(os.path.join(test_path,img))
        flat_data_arr_test.append(img_array.flatten())
        target_arr_test.append(ClassName.index(i))
    print(f'loaded category:{i} successfully')
    print()

flat_data_train=np.array(flat_data_arr_train)
target_train=np.array(target_arr_train)
df_train=pd.DataFrame(flat_data_train) #dataframe
df_train['Target']=target_train
x_train=df_train.iloc[:,:-1] # training input data
y_train=df_train.iloc[:,-1] # training output data

flat_data_test=np.array(flat_data_arr_test)
target_test=np.array(target_arr_test)
df_test=pd.DataFrame(flat_data_test)
df_test["Target"]=target_test
x_test=df_test.iloc[:,:-1]
y_test=df_test.iloc[:,-1]

# print(df_train)
# print(df_test)

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

"""# common function"""

def grid_search(model, param_grid_, x_train, y_train_):
  optimal_params = GridSearchCV(
      model,
      param_grid_,
      cv = 5,
        scoring = 'accuracy',
        verbose = 1,
        n_jobs = -1,
    )

optimal_params.fit(x_train, y_train_)
print(optimal_params.best_params_)

class Result():
def __init__(self, x_test, y_train, y_test, model, ClassName):
    self.x_test = x_test
    self.y_train = y_train
    self.y_test = y_test
    self.model = model
    self.ClassName = ClassName

    self.y_pred = self.model.predict(self.x_test)
    self.y_score = self.model.predict_proba(self.x_test)

def get_accuracy(self):
    return accuracy_score(self.y_pred, self.y_test)*100

def get_cm(self):
    return confusion_matrix(self.y_test, self.y_pred)

def get_onehot_label(self):
    label_binarizer = LabelBinarizer().fit(self.y_train)
    y_onehot_test = label_binarizer.transform(self.y_test)

    return y_onehot_test

def plot_cm(self):
    cm = self.get_cm()
    cm_df = pd.DataFrame(cm, index = self.ClassName, columns = self.ClassName)

    plt.figure(figsize=(5,4))
    sns.heatmap(cm_df, annot=True)
    plt.title('Confusion Matrix')
    plt.ylabel('Ground Truth')
    plt.xlabel('Predicted')
    plt.show()

def plot_ROC_AUC(self):
    y_onehot_test = self.get_onehot_label()
    n_classes = len(self.ClassName)

    fig, ax = plt.subplots(figsize=(6, 6))
    colors = cycle(["aqua", "darkorange", "cornflowerblue", "deeppink", "navy"])
    for class_id, color in zip(range(n_classes), colors):
        RocCurveDisplay.from_predictions(
            y_onehot_test[:, class_id],
            self.y_score[:, class_id],
            name=f"ROC curve for {ClassName[class_id]}",
            color=color,
            ax=ax,
        )

    _ = ax.set(
        xlabel="False Positive Rate",
        ylabel="True Positive Rate",
        title="ROC to OvR multiclass",
    )

def get_recall_f1score_precision(self):
    f1_Score = f1_score(self.y_test, self.y_pred, average =
None).reshape(len(ClassName), -1)
    recall = recall_score(self.y_test, self.y_pred, average =
None).reshape(len(ClassName), -1)
    precision = precision_score(self.y_test, self.y_pred, average =
None).reshape(len(ClassName), -1)

    result = np.concatenate((f1_Score, recall, precision), axis = 1)
    df_result = pd.DataFrame(result)
    df_result.columns = ["f1_score", "recall", "precision"]
    df_result.index = self.ClassName
    print(df_result)
def plot_every_metrix(self):
    print(f"The accuracy is {self.get_accuracy()}")
    print()

    self.get_recall_f1score_precision()
    print()

    self.plot_cm()
    print()

    self.plot_ROC_AUC()

"""# origion"""

param_grid = {
    'max_depth': [10, 20, 30, 40, 50],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 4, 8],
    'n_estimators': [100, 150, 200, 250]
}

model = RandomForestClassifier()
grid_search(model, param_grid, x_train, y_train)

rf_model = RandomForestClassifier(max_depth= 50, min_samples_leaf=1,
min_samples_split=2, n_estimators=250)
rf_model.fit(x_train, y_train)

ori_result = Result(x_test, y_train, y_test, rf_model, ClassName)
ori_result.plot_every_metrix()

"""# PCA"""

pca = PCA(n_components=0.85)
pca.fit(x_train)

x_train_pca = pca.transform(x_train)
```

```
x_test_pca = pca.transform(x_test)

print(x_train_pca.shape)

param_grid = {
    'max_depth': [10, 20, 30, 40],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 4, 8],
    'n_estimators': [100, 150, 200, 350, 450]
}

model = RandomForestClassifier()
grid_search(model, param_grid, x_train_pca, y_train)

rf_model = RandomForestClassifier(max_depth= 30, min_samples_leaf=1,
min_samples_split=2, n_estimators=350)
rf_model.fit(x_train_pca, y_train)

pca_result = Result(x_test_pca, y_train, y_test, rf_model, ClassName)
pca_result.plot_every_metrix()

"""# HOG"""

imgs = []
gray_img = []
hog_img = []
hog_feature_train = []
hog_feature_test = []
target_arr_train=[] # train output array
target_arr_test=[] # test output array

datadir="/content/drive/MyDrive/dataset/"
# path which contains all the categories of images
for i in ClassName:
    print(f'loading... category : {i}')
    # training
    training_path=os.path.join(datadir, 'train', i)
    for img_ in os.listdir(training_path):
            img = imread(os.path.join(training_path,img_))
            img_gray = color.rgb2gray(img)
            hog_vec, hog_vis = hog(img_gray, visualize=True)

            imgs.append(img)
            gray_img.append(img_gray)
            hog_img.append(hog_vis)
            hog_feature_train.append(hog_vec)
            target_arr_train.append(ClassName.index(i))

        # test
        test_path=os.path.join(datadir, 'test', i)
        for img_ in os.listdir(test_path):
            img = imread(os.path.join(test_path,img_))
            img_gray = color.rgb2gray(img)
            hog_vec, hog_vis = hog(img_gray, visualize=True)

            imgs.append(img)
            gray_img.append(img_gray)
            hog_img.append(hog_vis)
            hog_feature_test.append(hog_vec)
            target_arr_test.append(ClassName.index(i))
        print(f'loaded category:{i} successfully')
        print()

    hog_feature_train = np.array(hog_feature_train)
    target_train = np.array(target_arr_train)
    df_train_hog = pd.DataFrame(hog_feature_train) #dataframe
    df_train_hog['Target'] = target_train
    x_train_hog=df_train_hog.iloc[:,:-1] # training input data
    y_train=df_train_hog.iloc[:,-1] # training output data

    hog_feature_test = np.array(hog_feature_test)
    target_test=np.array(target_arr_test)
    df_test_hog=pd.DataFrame(hog_feature_test)
    df_test_hog["Target"]=target_test
    x_test_hog=df_test_hog.iloc[:,:-1]
    y_test=df_test_hog.iloc[:,-1]
```

```
fig = plt.figure(figsize=(12, 12))
k = 0
for i in range(len(imgs)):
    fig, ax = plt.subplots(1, 3, figsize=(12, 6), subplot_kw=dict(xticks=[], yticks=[]))
    ax[0].imshow(imgs[i])
    ax[0].set_title('input image')

    ax[1].imshow(gray_img[i], cmap='gray')
    ax[1].set_title('gray image')

    ax[2].imshow(hog_img[i])
    ax[2].set_title('HOG features');
plt.show()

param_grid = {
    'max_depth': [10, 20, 30, 40, 50],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 4, 8],
    'n_estimators': [100, 150, 200, 350, 450]
}

model = RandomForestClassifier()
grid_search(model, param_grid, x_train_hog, y_train)

rf_model = RandomForestClassifier(max_depth= 50, min_samples_leaf=1,
min_samples_split=2, n_estimators=450)
rf_model.fit(x_train_hog, y_train)

hog_result = Result(x_test_hog, y_train, y_test, rf_model, ClassName)
hog_result.plot_every_metrix()

"""### HOG + PCA"""

pca = PCA(n_components=0.85)
pca.fit(x_train_hog)

x_train_hog_pca = pca.transform(x_train_hog)
```

```
x_test_hog_pca = pca.transform(x_test_hog)

x_train_hog_pca.shape

param_grid = {
        'max_depth': [10, 20, 30, 40, 50],
        'min_samples_leaf': [1, 2, 4],
        'min_samples_split': [2, 4, 8],
        'n_estimators': [100, 150, 200, 350, 450]
}

model = RandomForestClassifier()
grid_search(model, param_grid, x_train_hog_pca, y_train)

rf_model = RandomForestClassifier(max_depth= 50, min_samples_leaf=1,
min_samples_split=2, n_estimators=350)
rf_model.fit(x_train_hog_pca, y_train)

hog_pca_result = Result(x_test_hog_pca, y_train, y_test, rf_model, ClassName)
hog_pca_result.plot_every_metrix()
```