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

```
Mounted at /content/drive
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset
from torchvision import transforms
import numpy as np
import pandas as pd
from PIL import Image, ImageEnhance
import argparse
import os
import copy
from torch.utils.data import DataLoader
from torchvision.models import resnet18
from pprint import pprint
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import f1_score
import cv2
from sklearn.naive_bayes import GaussianNB
from torch.utils.data import DataLoader

LABELS_Severity = {35: 0,
                   43: 0,
                   47: 1,
                   53: 1,
                   61: 2,
                   65: 2,
                   71: 2,
                   85: 2}


mean = (.1706)
std = (.2112)
normalize = transforms.Normalize(mean=mean, std=std)
train_transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),
    transforms.Resize(size=(224,224)),
    # transforms.RandomHorizontalFlip(),
    # transforms.RandomRotation(10),
    #transforms.RandomCrop((224,224), padding=4),
    transforms.ToTensor(),
    normalize,
])
test_transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),
    transforms.Resize(size=(224,224)),
    transforms.ToTensor(),
    normalize,
])



class OCTDataset(Dataset):
    def __init__(self, annot=None, subset='train', transform=None, device='cpu'):
        if subset == 'train':
            self.annot = pd.read_csv("/content/drive/MyDrive/FML_Project/df_prime_train.csv")
        elif subset == 'test':
            self.annot = pd.read_csv("/content/drive/MyDrive/FML_Project/df_prime_test.csv")

        # Extract "Patient_ID" and "Week_Num" columns
        # print("Before Pairing ", len(self.annot))
        self.patient_ids = self.annot["Patient_ID"]
        self.week_nums = self.annot["Week_Num"]
        self.patient_ids = self.annot["Patient_ID"]
        self.annot['Severity_Label'] = [LABELS_Severity[drss] for drss in copy.deepcopy(self.annot['DRSS'].values)]
        self.drss_class = self.annot['Severity_Label']

        # Create unique pairs of values
        self.unique_pairs = set(zip(self.patient_ids, self.week_nums, self.drss_class))

        self.root = os.path.expanduser("/content/drive/MyDrive/FML_Project/")
```

```
            self.transform = transform
            self.nb_classes=len(np.unique(list(LABELS_Severity.values())))
            self.path_list = self.annot['File_Path'].values

            self._labels = [pair[2] for pair in self.unique_pairs]
            assert len(self.unique_pairs) == len(self._labels)

            max_samples = int(len(self._labels)) #32 #int(len(self._labels)/2)
            self.max_samples = max_samples
            self.device = device

        def __getitem__(self, index):
            # Get the Patient_ID and Week_Num from the indexed element in unique_pairs
            patient_id, week_num, target = list(self.unique_pairs)[index]

            # Filter the annot DataFrame to select rows that match the Patient_ID and Week_Num
            filtered_df = self.annot[(self.annot['Patient_ID'] == patient_id) & (self.annot['Week_Num'] == week_num)]

            # Extract the file paths from the filtered DataFrame and return them as a list
            file_paths = [self.root + file_path for file_path in filtered_df['File_Path'].values.tolist()]
            fused_image_path = os.path.dirname(file_paths[0])+"/fused_image.jpg"

            img = Image.open(fused_image_path)
            img_gray = img.convert("L")

            # # Apply image sharpening
            # sharpness = ImageEnhance.Sharpness(img_gray)
            # img_sharpened = sharpness.enhance(2.0)  # Adjust the factor (2.0) to control the level of sharpening
            if self.transform is not None:
                img_sharpened = self.transform(img_gray)

            # # Convert the sharpened image back to a NumPy array
            # img_sharpened_np = np.array(img_sharpened)
            # # Flatten the sharpened image array and concatenate it with the HOG features
            # img_sharpened_flat = img_sharpened_np.flatten()

            return img_sharpened, target

        def __len__(self):
            if self.max_samples is not None:
                return min(len(self._labels), self.max_samples)
            else:
                return len(self._labels)


    #set up the device (GPU or CPU)
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    print('Found device:', device)

    trainset = OCTDataset(subset='train', transform=train_transform, device=device)

    #define the hyperparameters
    batch_size = 32

    trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True, num_workers=2)
    print('Train and Test loader complete')

    # for i in range(10):
    #   print(trainset[i][0].shape)
    print(len(trainset))
```

```
     Found device: cuda:0
     Train and Test loader complete
     495
```

```
    # Initialize the Gaussian Naive Bayes classifier
    clf = GaussianNB()

    # Train the classifier
    train_features, train_labels = [], []
    for i, (inputs, labels) in enumerate(trainloader):
        train_features.extend(inputs.view(inputs.shape[0], -1).numpy())
        train_labels.extend(labels.numpy())

    train_features, train_labels = np.array(train_features), np.array(train_labels)
    clf.fit(train_features, train_labels)
```

```
# clf.fit(inputs, labels.numpy())
```

GaussianNB

```
GaussianNB()
```

```
testset = OCTDataset(subset='test', transform=test_transform, device=device)
testloader = DataLoader(testset, batch_size=batch_size, shuffle=True, num_workers=4)
print(len(testloader))

# Test the classifier
test_features, test_labels = [], []
for inputs, labels in testloader:
    test_features.extend(inputs.view(inputs.shape[0], -1).numpy())
    test_labels.extend(labels.numpy())
test_features, test_labels = np.array(test_features), np.array(test_labels)

predicted_labels = clf.predict(test_features)

# true_labels = []
# predicted_labels_list = []

# for i, (inputs, labels) in enumerate(testloader):
#     predicted_labels = clf.predict(inputs)
#     true_labels.extend(labels.numpy())
#     predicted_labels_list.extend(predicted_labels)

balanced_accuracy = balanced_accuracy_score(test_labels, predicted_labels)
f1 = f1_score(test_labels, predicted_labels, average='weighted')  # Use 'weighted' if you have imbalanced classes

print(f'Balanced accuracy: {balanced_accuracy:.4f}')
print(f'F1 score: {f1:.4f}')
```

```
6
/usr/local/lib/python3.9/dist-packages/torch/utils/data/dataloader.py:561: UserWarning: This DataLoader will create 4 worker processes
  warnings.warn(_create_warning_msg(
Balanced accuracy: 0.4955
F1 score: 0.4967
```

```
print(test_labels)
print(predicted_labels)
```

```
[1 1 1 0 1 1 1 0 1 1 1 0 2 1 1 2 1 0 2 2 1 2 2 0 0 1 1 1 2 1 1 1 0 2 0 1 0
 1 0 0 0 0 1 2 0 1 0 1 0 1 1 1 0 1 1 0 1 2 1 1 1 1 2 1 0 1 1 1 1 1 1 1 2 1
 2 0 0 0 0 2 2 2 1 1 1 1 1 0 0 1 2 1 1 1 2 1 0 1 2 1 0 2 2 0 2 2 1 0 1 1 0
 1 1 1 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 1 1 0 2 0 1 1 0 1 0 2 1 2 2
 0 2 0 0 2 1 1 2 1 1 0 2 0 1 0]
[1 1 2 1 1 0 1 1 0 1 1 1 2 2 1 0 1 1 0 2 1 0 0 1 1 1 1 1 2 1 1 1 1 1 0 2 0
 1 1 1 1 1 2 0 0 1 2 1 2 1 0 1 0 2 1 0 0 0 1 2 0 1 2 0 1 2 1 1 1 2 1 1 2 1
 2 1 1 1 1 2 1 0 0 1 0 0 1 1 2 2 2 1 1 1 1 0 0 1 2 1 1 2 2 1 2 2 1 1 2 2 1
 2 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 2 2 1 0 1 1 1 2 1 1 2
 0 2 0 1 1 1 1 2 1 0 1 0 1 2 1]
```

✓ 0s completed at 12:31 AM