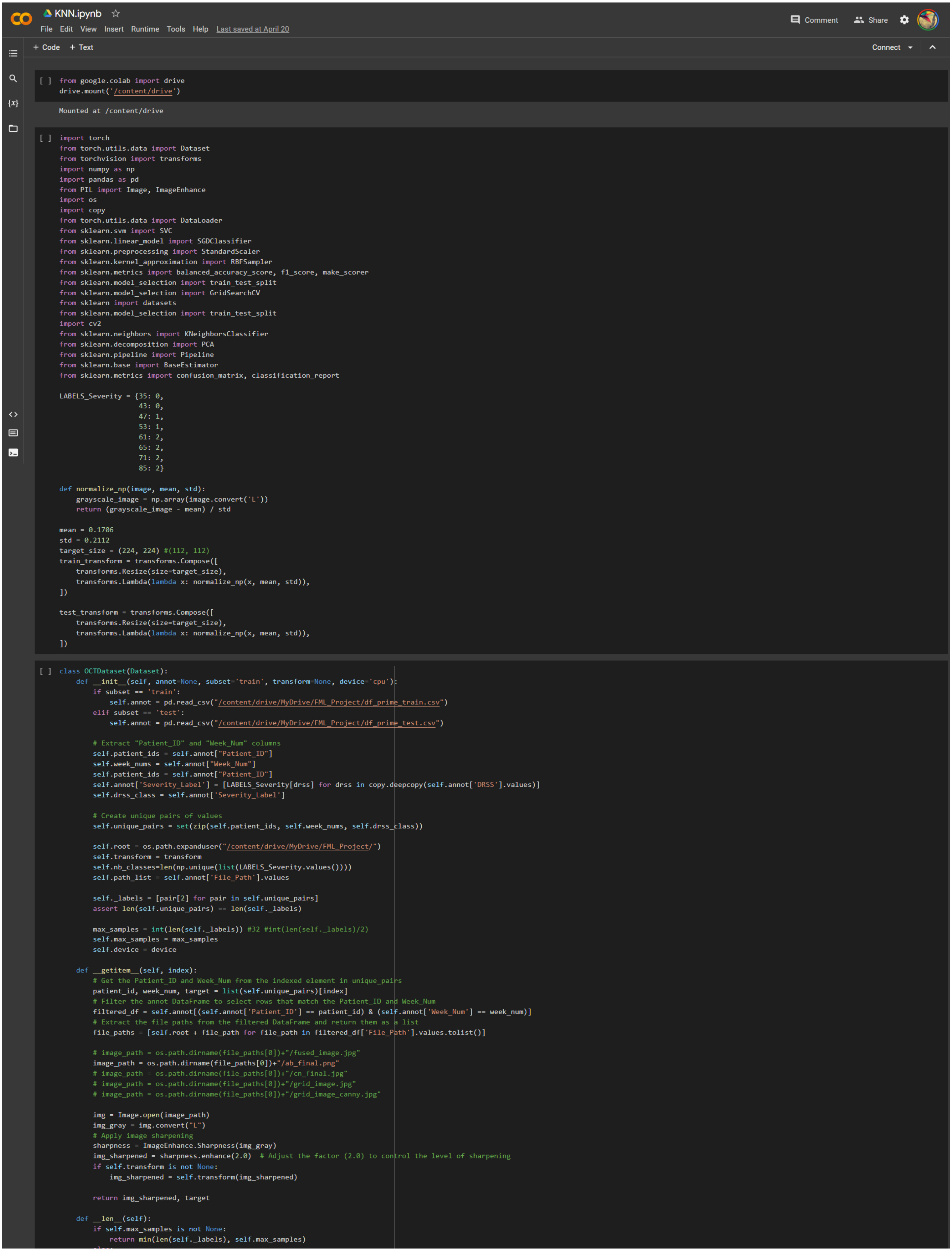
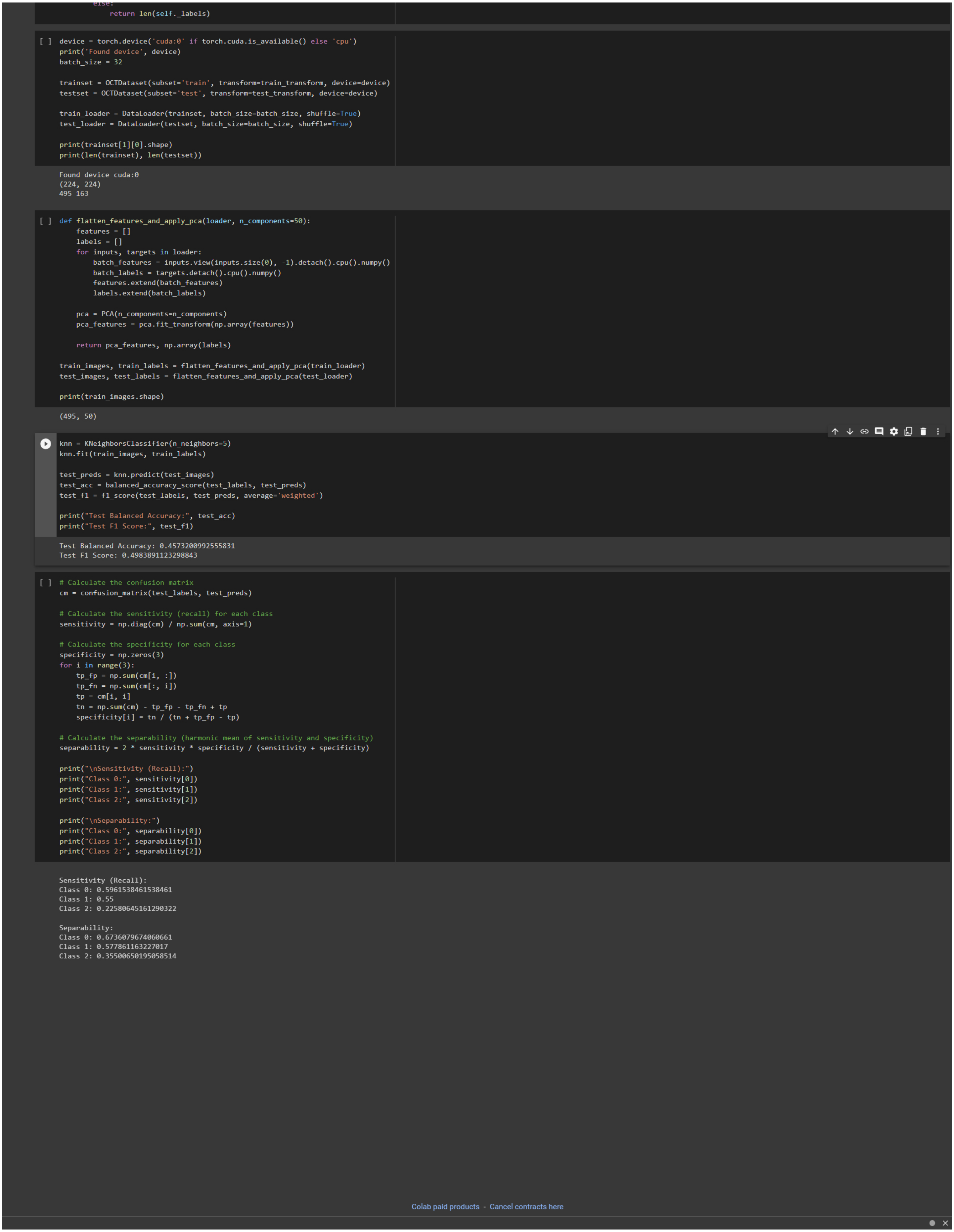

KNN.ipynb
File Edit View Insert Runtime Tools Help Last saved at April 20
Comment
Share
+ Code + Text
Connect

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

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

```
[ ] import torch
    from torch.utils.data import Dataset
    from torchvision import transforms
    import numpy as np
    import pandas as pd
    from PIL import Image, ImageEnhance
    import os
    import copy
    from torch.utils.data import DataLoader
    from sklearn.svm import SVC
    from sklearn.linear_model import SGDClassifier
    from sklearn.preprocessing import StandardScaler
    from sklearn.kernel_approximation import RBFSampler
    from sklearn.metrics import balanced_accuracy_score, f1_score, make_scorer
    from sklearn.model_selection import train_test_split
    from sklearn.model_selection import GridSearchCV
    from sklearn import datasets
    from sklearn.model_selection import train_test_split
    import cv2
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.decomposition import PCA
    from sklearn.pipeline import Pipeline
    from sklearn.base import BaseEstimator
    from sklearn.metrics import confusion_matrix, classification_report

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

    def normalize_np(image, mean, std):
        grayscale_image = np.array(image.convert('L'))
        return (grayscale_image - mean) / std

    mean = 0.1706
    std = 0.2112
    target_size = (224, 224) #(112, 112)
    train_transform = transforms.Compose([
        transforms.Resize(size=target_size),
        transforms.Lambda(lambda x: normalize_np(x, mean, std)),
    ])

    test_transform = transforms.Compose([
        transforms.Resize(size=target_size),
        transforms.Lambda(lambda x: normalize_np(x, mean, std)),
    ])
```

```
[ ] 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
            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()]

            # image_path = os.path.dirname(file_paths[0])+"/fused_image.jpg"
            image_path = os.path.dirname(file_paths[0])+"/ab_final.png"
            # image_path = os.path.dirname(file_paths[0])+"/cn_final.jpg"
            # image_path = os.path.dirname(file_paths[0])+"/grid_image.jpg"
            # image_path = os.path.dirname(file_paths[0])+"/grid_image_canny.jpg"

            img = Image.open(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_sharpened)

            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)
```

```
[ ] device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    print('Found device', device)
    batch_size = 32

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

    train_loader = DataLoader(trainset, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(testset, batch_size=batch_size, shuffle=True)

    print(trainset[1][0].shape)
    print(len(trainset), len(testset))
```

```
Found device cuda:0
(224, 224)
495 163
```

```
[ ] def flatten_features_and_apply_pca(loader, n_components=50):
        features = []
        labels = []
        for inputs, targets in loader:
            batch_features = inputs.view(inputs.size(0), -1).detach().cpu().numpy()
            batch_labels = targets.detach().cpu().numpy()
            features.extend(batch_features)
            labels.extend(batch_labels)

        pca = PCA(n_components=n_components)
        pca_features = pca.fit_transform(np.array(features))

        return pca_features, np.array(labels)

    train_images, train_labels = flatten_features_and_apply_pca(train_loader)
    test_images, test_labels = flatten_features_and_apply_pca(test_loader)

    print(train_images.shape)
```

```
(495, 50)
```

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(train_images, train_labels)

test_preds = knn.predict(test_images)
test_acc = balanced_accuracy_score(test_labels, test_preds)
test_f1 = f1_score(test_labels, test_preds, average='weighted')

print("Test Balanced Accuracy:", test_acc)
print("Test F1 Score:", test_f1)
```

```
Test Balanced Accuracy: 0.4573200992555831
Test F1 Score: 0.4983891123298843
```

```
[ ] # Calculate the confusion matrix
    cm = confusion_matrix(test_labels, test_preds)

    # Calculate the sensitivity (recall) for each class
    sensitivity = np.diag(cm) / np.sum(cm, axis=1)

    # Calculate the specificity for each class
    specificity = np.zeros(3)
    for i in range(3):
        tp_fp = np.sum(cm[i, :])
        tp_fn = np.sum(cm[:, i])
        tp = cm[i, i]
        tn = np.sum(cm) - tp_fp - tp_fn + tp
        specificity[i] = tn / (tn + tp_fp - tp)

    # Calculate the separability (harmonic mean of sensitivity and specificity)
    separability = 2 * sensitivity * specificity / (sensitivity + specificity)

    print("\nSensitivity (Recall):")
    print("Class 0:", sensitivity[0])
    print("Class 1:", sensitivity[1])
    print("Class 2:", sensitivity[2])

    print("\nSeparability:")
    print("Class 0:", separability[0])
    print("Class 1:", separability[1])
    print("Class 2:", separability[2])
```

```

Sensitivity (Recall):
Class 0: 0.5961538461538461
Class 1: 0.55
Class 2: 0.22580645161290322

Separability:
Class 0: 0.6736079674060661
Class 1: 0.577861163227017
Class 2: 0.35500650195058514
```

Colab paid products - Cancel contracts here