

CNN_Test_Latest.ipynb

File Edit View Insert Runtime Tools Help

Comment Share RAM Disk

```

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

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[2] !pip install segmentation-models-pytorch

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting segmentation-models-pytorch
  Downloading segmentation_models_pytorch-0.3.2-py3-none-any.whl (106 kB)
Requirement already satisfied: torchvision>=0.5.0 in /usr/local/lib/python3.9/dist-packages (from segmentation-models-pytorch) (0.15.1+cu118)
Requirement already satisfied: pillow in /usr/local/lib/python3.9/dist-packages (from segmentation-models-pytorch) (8.4.0)
Collecting efficientnet-pytorch<0.7.1
  Downloading efficientnet_pytorch-0.7.1.tar.gz (21 kB)
    Preparing metadata (setup.py) ... done
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from segmentation-models-pytorch) (4.65.0)
Collecting pretrainedmodels<0.7.4
  Downloading pretrainedmodels-0.7.4.tar.gz (58 kB)
    Preparing metadata (setup.py) ... done
Collecting timm<0.6.12
  Downloading timm-0.6.12-py3-none-any.whl (549 kB)
    549.1/549.1 kB 18.5 MB/s eta 0:00:00
Requirement already satisfied: torch in /usr/local/lib/python3.9/dist-packages (from efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (2.0.0+cu118)
Collecting munch
  Downloading munch-2.5.0-py2.py3-none-any.whl (10 kB)
Collecting huggingface-hub
  Downloading huggingface_hub-0.13.4-py3-none-any.whl (206 kB)
    206.3/206.3 kB 21.6 MB/s eta 0:00:00
Requirement already satisfied: pyyaml in /usr/local/lib/python3.9/dist-packages (from timm==0.6.12->segmentation-models-pytorch) (6.0)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from torchvision==0.5.0->segmentation-models-pytorch) (2.27.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from torchvision==0.5.0->segmentation-models-pytorch) (1.23.4)
Requirement already satisfied: typing_extensions in /usr/local/lib/python3.9/dist-packages (from torch<efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (4.5.0)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.9/dist-packages (from torch<efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (2.0.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.9/dist-packages (from torch<efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (3.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from torch<efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (3.11.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.9/dist-packages (from torch<efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (1.11.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.9/dist-packages (from torch<efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (3.1.2)
Requirement already satisfied: cma in /usr/local/lib/python3.9/dist-packages (from triton==2.0.0->torch<efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (3.25.2)
Requirement already satisfied: lit in /usr/local/lib/python3.9/dist-packages (from triton==2.0.0->torch<efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (16.0.1)
Requirement already satisfied: packaging<=20.9 in /usr/local/lib/python3.9/dist-packages (from huggingface-hub-timm==0.6.12->segmentation-models-pytorch) (23.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from munch->pretrainedmodels==0.7.4->segmentation-models-pytorch) (1.16.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests->torchvision==0.5.0->segmentation-models-pytorch) (2022.12.7)
Requirement already satisfied: charset-normalizer==2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests->torchvision==0.5.0->segmentation-models-pytorch) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->torchvision==0.5.0->segmentation-models-pytorch) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests->torchvision==0.5.0->segmentation-models-pytorch) (1.26.15)
Requirement already satisfied: MarkupSafe==2.0 in /usr/local/lib/python3.9/dist-packages (from jinja2->torch<efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (2.1.2)
Requirement already satisfied: mpmath>0.19 in /usr/local/lib/python3.9/dist-packages (from sympy->torch<efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (1.3.0)
Building wheels for collected packages: efficientnet-pytorch, pretrainedmodels
  Building wheel for efficientnet-pytorch (setup.py) ... done
    Created wheel for efficientnet-pytorch: filename=efficientnet_pytorch-0.7.1-py3-none-any.whl size=16444 sha256=554dd9c769cf76b0e28869c6fb1bef60b1a70bbdb4a628ceceba51b243afbc
  Stored in directory: /root/.cache/pip/wheels/29/16/752e89d88d33af39a288421e64d613b5f652918e39ff1f8e3
  Building wheel for pretrainedmodels (setup.py) ... done
    Created wheel for pretrainedmodels: filename=pretrainedmodels-0.7.4-py3-none-any.whl size=60962 sha256=949df65086217252034c8f303fa999fa8ba40c9328f153659927db8bc0032f2a
  Stored in directory: /root/.cache/pip/wheels/d1/b4/e2/f3015f1ab76f34be28e04c4bcec27e8cafaf70d2caef0bc3b
Successfully built efficientnet-pytorch pretrainedmodels
Installing collected packages: munch, huggingface-hub, timm, pretrainedmodels, efficientnet-pytorch, segmentation-models-pytorch
Successfully installed efficientnet-pytorch-0.7.1 huggingface-hub-0.13.4 munch-2.5.0 pretrainedmodels-0.7.4 segmentation-models-pytorch-0.3.2 timm-0.6.12

[29]
import torch
import math
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
import segmentation_models_pytorch as smp
from sklearn.metrics import balanced_accuracy_score, f1_score, confusion_matrix
import cv2
import timm
import random
from torch.utils.data import WeightedRandomSampler

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.ToTensor(),
    normalize,
])
test_transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),
    transforms.Resize(size=(224, 224)),
    transforms.ToTensor(),
    normalize,
])

[15]
class OCTDataset(Dataset):
    def __init__(self, annot=None, unique_pairs=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 == 'val':
            self.annot = pd.read_csv("/content/drive/MyDrive/FML_Project/df_prime_train.csv")
            self.unique_pairs = unique_pairs
        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[dress] for dress in copy.deepcopy(self.annot['DRSS'].values)]
        self.dress_class = self.annot['Severity_Label']

        if subset == 'train':
            # Create unique pairs of values
            self.unique_pairs = set(zip(self.patient_ids, self.week_nums, self.dress_class))

        # Create a list from the set of unique_pairs
        unique_pairs_list = list(self.unique_pairs)

```

```

# Shuffle the unique_pairs list
random.shuffle(unique_pairs_list)

# Calculate the index at which to split the list
split_index = int(0.8 * len(unique_pairs_list))

# Split the list into training and validation pairs
self.unique_pairs = unique_pairs_list[:split_index]
self.unique_validation_pairs = unique_pairs_list[split_index:]

elif subset == 'test':
    # 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))
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()]
    # Canny edge of each image blended together
    fused_image_path = os.path.dirname(file_paths[0])+"/cn_final.jpg"
    # Canny edge of each image formed as a grid
    # fused_image_path = os.path.dirname(file_paths[0])+"/grid_image_canny.jpg"
    # OCT Images formed as a grid
    # fused_image_path = os.path.dirname(file_paths[0])+"/grid_image.jpg"
    # Canny edge of Alpha blended image
    # fused_image_path = os.path.dirname(file_paths[0])+"/ab_edge_image.jpg"

    img = Image.open(fused_image_path)
    img_gray = img.convert("L")
    if self.transform is not None:
        img_sharpened = self.transform(img_gray)

    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)

# Neural network architecture
class OCTClassifier(torch.nn.Module):
    def __init__(self):
        super(OCTClassifier, self).__init__()
        self.unet = smp.Unet(
            encoder_name="resnet34",   # encoder architecture (e.g., resnet34)
            encoder_name="timm-efficientnet-b7", # encoder architecture (e.g., EfficientNet-B3)
            encoder_weights="imagenet", # pre-trained weights for the encoder
            in_channels=3,           # input channels
            classes=3,               # number of output channels
        )
        self.avg_pool = torch.nn.AdaptiveAvgPool2d((1, 1)) # Pooling layer to get the [32, 3] output shape

    def forward(self, x):
        x = self.unet(x)           # U-Net forward pass
        x = self.avg_pool(x)       # Apply the pooling layer
        x = x.view(x.size(0), -1)  # Reshape the tensor to [batch_size, 3]
        return x

    # Oversampler to use weighted random sampling for data loaders
    def create_oversampler(targets):
        class_sample_counts = np.bincount(targets)
        weights = 1.0 / torch.tensor(class_sample_counts, dtype=torch.float)
        sample_weights = weights[targets]
        sampler = WeightedRandomSampler(weights=sample_weights, num_samples=len(targets), replacement=True)
        return sampler

```

[] #####

```

# Run this code only during Grid Search
#####
def train_model(trainloader, val_loader, testloader, model, criterion, optimizer, device, num_epochs):
    best_val_balanced_accuracy = 0
    best_val_balanced_accuracy_epoch_number = 0
    for epoch in range(num_epochs):
        running_loss = 0.0
        model.train()
        for i, (inputs, labels) in enumerate(trainloader):
            inputs = inputs.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        model.eval()
        val_running_loss = 0.0
        val_preds = []
        val_true_labels = []
        with torch.no_grad():
            for val_inputs, val_labels in val_loader:
                val_inputs = val_inputs.to(device)
                val_labels = val_labels.to(device)
                val_outputs = model(val_inputs)
                valdn_loss = criterion(val_outputs, val_labels)
                val_running_loss += valdn_loss.item()

                _, val_predicted = torch.max(val_outputs.data, 1)
                val_preds.extend(val_predicted.cpu().numpy())
                val_true_labels.extend(val_labels.cpu().numpy())

        train.loss = running_loss / len(trainloader)
        val.loss = val_running_loss / len(val_loader)
        val_balanced_accuracy = balanced_accuracy_score(val_true_labels, val_preds)

        if val_balanced_accuracy > best_val_balanced_accuracy:
            best_val_balanced_accuracy = val_balanced_accuracy
            best_model = model
            best_val_balanced_accuracy_epoch_number = epoch + 1

    return best_model, best_val_balanced_accuracy, best_val_balanced_accuracy_epoch_number

```

```

# After obtaining best model parameters from grid search change the exit condition in below code.
# Use this code for final run.
#####
def train_model(trainloader, val_loader, model, criterion, optimizer, device, num_epochs):
    best_balanced_accuracy = 0
    best_balanced_accuracy_epoch_number = 0
    final_balanced_accuracy = 0
    for epoch in range(num_epochs):
        running_loss = 0.0
        model.train()
        for i, (inputs, labels) in enumerate(trainloader):
            inputs = inputs.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        model.eval()
        val_running_loss = 0.0
        val_balanced_accuracy = 0.0
        val_loss = 0.0
        val_preds = []
        val_true_labels = []
        with torch.no_grad():
            for val_inputs, val_labels in val_loader:
                val_inputs = val_inputs.to(device)
                val_labels = val_labels.to(device)
                val_outputs = model(val_inputs)
                valdn_loss = criterion(val_outputs, val_labels)
                val_running_loss += valdn_loss.item()

                _, val_predicted = torch.max(val_outputs.data, 1)
                val_preds.extend(val_predicted.cpu().numpy())
                val_true_labels.extend(val_labels.cpu().numpy())

        # compute the balanced accuracy
        train_loss = running_loss / len(trainloader)
        val_loss = val_running_loss / len(val_loader)
        val_balanced_accuracy = balanced_accuracy_score(val_true_labels, val_preds)

        print(f'Epoch {epoch + 1} | Train Loss: {train_loss:.3f} | Val Loss: {val_loss:.3f} | Val Balanced Accuracy: {val_balanced_accuracy:.2f}')
    return model, train_loss, val_loss, val_balanced_accuracy

[17] annot_train_prime = "/content/drive/MyDrive/FML_Project/df_prime_train.csv"
annot_test_prime = "/content/drive/MyDrive/FML_Project/df_prime_test.csv"
data_root = "/content/drive/MyDrive/FML_Project/"

#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)
valset = OCTDataset(subset='val', unique_pairs=trainset.unique_validation_pairs, transform=train_transform, device=device)
testset = OCTDataset(subset='test', transform=test_transform, device=device)

oversampler = create_oversampler(trainset._labels)

Found device: cuda:0

[ ] #####
# Run this code only during Grid Search
#####
# Define parameter grid for Grid Search
batch_sizes = [8, 16]
learning_rates = [1e-3, 1e-4]
num_epochs_list = [200]

best_balanced_accuracy = 0
best_params = None
best_model = None

# Perform grid search
for batch_size in batch_sizes:
    for learning_rate in learning_rates:
        for num_epochs in num_epochs_list:
            print(f'Training with batch_size={batch_size}, learning_rate={(learning_rate)}, num_epochs={(num_epochs)}')
            trainloader = DataLoader(trainset, batch_size=batch_size, sampler=oversampler, num_workers=2)
            val_loader = DataLoader(valset, batch_size=batch_size, shuffle=True)
            testloader = DataLoader(testset, batch_size=batch_size, shuffle=False, num_workers=2)
            model = OCTClassifier().to(device)
            optimizer = optim.Adam(model.parameters(), lr=learning_rate)

            #define the loss function
            criterion = nn.CrossEntropyLoss()

            # To handle imbalanced dataset:
            class_counts = np.bincount(trainset._labels)
            print(class_counts)
            total_samples = len(trainset)

            # Adjust weights in proportion to class distribution in trainset
            class_weights = torch.FloatTensor(total_samples / (len(class_counts) * class_counts)).to(device)
            class_weights[0] = class_weights[0] * 2
            class_weights[2] = class_weights[2] * 4

            criterion = nn.CrossEntropyLoss(weight=class_weights)

            model, val_balanced_accuracy, best_balanced_accuracy_epoch_number = train_model(trainloader, val_loader, testloader, model, criterion, optimizer, device, num_epochs)

            if val_balanced_accuracy > best_balanced_accuracy:
                best_balanced_accuracy = val_balanced_accuracy
                best_params = (batch_size, learning_rate, num_epochs, best_balanced_accuracy_epoch_number)
                best_model = model

print(f'Best balanced accuracy: {best_balanced_accuracy}')
print(f'Best parameters: batch_size={best_params[0]}, learning_rate={best_params[1]}, num_epochs={best_params[2]}, epoch_number={best_params[3]}')

[18] # Best model Hyper-parameters obtained from grid search
batch_size = 16
learning_rate = 1e-4
num_epochs = 100

best_balanced_accuracy = 0
best_params = None
best_model = None

# Run this
print(f'Training with batch_size={batch_size}, learning_rate={(learning_rate)}, num_epochs={(num_epochs)}')
trainloader = DataLoader(trainset, batch_size=batch_size, sampler=oversampler, num_workers=2)
val_loader = DataLoader(valset, batch_size=batch_size, shuffle=True)
testloader = DataLoader(testset, batch_size=batch_size, shuffle=False, num_workers=2)
model = OCTClassifier().to(device)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

#define the loss function
criterion = nn.CrossEntropyLoss()

# To handle imbalanced dataset:
class_counts = np.bincount(trainset._labels)
total_samples = len(trainset)

```

```

# Adjust weights in proportion to class distribution in trainset
class_weights = torch.FloatTensor(total_samples / (len(class_counts) * class_counts)).to(device)
class_weights[0] = class_weights[0] * 2
class_weights[2] = class_weights[2] * 8

criterion = nn.CrossEntropyLoss(weight=class_weights)

model, train_loss, val_loss, val_balanced_accuracy = train_model(trainloader, val_loader, model, criterion, optimizer, device, num_epochs)

Epoch 43 | Train Loss: 0.069 | Val Loss: 1.067 | Val Balanced Accuracy: 0.57
Epoch 44 | Train Loss: 0.065 | Val Loss: 1.113 | Val Balanced Accuracy: 0.58
Epoch 45 | Train Loss: 0.145 | Val Loss: 0.924 | Val Balanced Accuracy: 0.57
Epoch 46 | Train Loss: 0.109 | Val Loss: 1.194 | Val Balanced Accuracy: 0.56
Epoch 47 | Train Loss: 0.114 | Val Loss: 1.191 | Val Balanced Accuracy: 0.62
Epoch 48 | Train Loss: 0.087 | Val Loss: 1.111 | Val Balanced Accuracy: 0.52
Epoch 49 | Train Loss: 0.063 | Val Loss: 1.076 | Val Balanced Accuracy: 0.55
Epoch 50 | Train Loss: 0.050 | Val Loss: 1.024 | Val Balanced Accuracy: 0.57
Epoch 51 | Train Loss: 0.122 | Val Loss: 1.309 | Val Balanced Accuracy: 0.57
Epoch 52 | Train Loss: 0.123 | Val Loss: 1.321 | Val Balanced Accuracy: 0.57
Epoch 53 | Train Loss: 0.055 | Val Loss: 1.318 | Val Balanced Accuracy: 0.57
Epoch 54 | Train Loss: 0.052 | Val Loss: 1.287 | Val Balanced Accuracy: 0.63
Epoch 55 | Train Loss: 0.061 | Val Loss: 1.117 | Val Balanced Accuracy: 0.63
Epoch 56 | Train Loss: 0.044 | Val Loss: 1.128 | Val Balanced Accuracy: 0.62
Epoch 57 | Train Loss: 0.043 | Val Loss: 0.971 | Val Balanced Accuracy: 0.65
Epoch 58 | Train Loss: 0.050 | Val Loss: 1.093 | Val Balanced Accuracy: 0.58
Epoch 59 | Train Loss: 0.048 | Val Loss: 1.023 | Val Balanced Accuracy: 0.63
Epoch 60 | Train Loss: 0.036 | Val Loss: 1.062 | Val Balanced Accuracy: 0.60
Epoch 61 | Train Loss: 0.036 | Val Loss: 1.296 | Val Balanced Accuracy: 0.65
Epoch 62 | Train Loss: 0.027 | Val Loss: 1.438 | Val Balanced Accuracy: 0.62
Epoch 63 | Train Loss: 0.049 | Val Loss: 1.495 | Val Balanced Accuracy: 0.64
Epoch 64 | Train Loss: 0.078 | Val Loss: 1.014 | Val Balanced Accuracy: 0.56
Epoch 65 | Train Loss: 0.100 | Val Loss: 1.031 | Val Balanced Accuracy: 0.59
Epoch 66 | Train Loss: 0.040 | Val Loss: 1.117 | Val Balanced Accuracy: 0.60
Epoch 67 | Train Loss: 0.061 | Val Loss: 1.069 | Val Balanced Accuracy: 0.61
Epoch 68 | Train Loss: 0.044 | Val Loss: 1.057 | Val Balanced Accuracy: 0.57
Epoch 69 | Train Loss: 0.063 | Val Loss: 1.481 | Val Balanced Accuracy: 0.54
Epoch 70 | Train Loss: 0.065 | Val Loss: 1.301 | Val Balanced Accuracy: 0.59
Epoch 71 | Train Loss: 0.061 | Val Loss: 1.246 | Val Balanced Accuracy: 0.60
Epoch 72 | Train Loss: 0.057 | Val Loss: 1.334 | Val Balanced Accuracy: 0.59
Epoch 73 | Train Loss: 0.047 | Val Loss: 1.146 | Val Balanced Accuracy: 0.63
Epoch 74 | Train Loss: 0.051 | Val Loss: 1.438 | Val Balanced Accuracy: 0.56
Epoch 75 | Train Loss: 0.056 | Val Loss: 1.071 | Val Balanced Accuracy: 0.57
Epoch 76 | Train Loss: 0.043 | Val Loss: 1.564 | Val Balanced Accuracy: 0.61
Epoch 77 | Train Loss: 0.027 | Val Loss: 1.381 | Val Balanced Accuracy: 0.52
Epoch 78 | Train Loss: 0.018 | Val Loss: 1.446 | Val Balanced Accuracy: 0.57
Epoch 79 | Train Loss: 0.021 | Val Loss: 1.653 | Val Balanced Accuracy: 0.56
Epoch 80 | Train Loss: 0.028 | Val Loss: 1.599 | Val Balanced Accuracy: 0.52
Epoch 81 | Train Loss: 0.029 | Val Loss: 2.206 | Val Balanced Accuracy: 0.55
Epoch 82 | Train Loss: 0.026 | Val Loss: 1.723 | Val Balanced Accuracy: 0.52
Epoch 83 | Train Loss: 0.020 | Val Loss: 1.766 | Val Balanced Accuracy: 0.61
Epoch 84 | Train Loss: 0.027 | Val Loss: 1.648 | Val Balanced Accuracy: 0.57
Epoch 85 | Train Loss: 0.031 | Val Loss: 1.857 | Val Balanced Accuracy: 0.56
Epoch 86 | Train Loss: 0.026 | Val Loss: 1.529 | Val Balanced Accuracy: 0.59
Epoch 87 | Train Loss: 0.016 | Val Loss: 1.628 | Val Balanced Accuracy: 0.52
Epoch 88 | Train Loss: 0.028 | Val Loss: 1.572 | Val Balanced Accuracy: 0.59
Epoch 89 | Train Loss: 0.023 | Val Loss: 1.126 | Val Balanced Accuracy: 0.61
Epoch 90 | Train Loss: 0.022 | Val Loss: 1.608 | Val Balanced Accuracy: 0.64
Epoch 91 | Train Loss: 0.013 | Val Loss: 2.265 | Val Balanced Accuracy: 0.58
Epoch 92 | Train Loss: 0.035 | Val Loss: 1.456 | Val Balanced Accuracy: 0.64
Epoch 93 | Train Loss: 0.018 | Val Loss: 1.870 | Val Balanced Accuracy: 0.64
Epoch 94 | Train Loss: 0.016 | Val Loss: 1.311 | Val Balanced Accuracy: 0.63
Epoch 95 | Train Loss: 0.033 | Val Loss: 1.625 | Val Balanced Accuracy: 0.64
Epoch 96 | Train Loss: 0.094 | Val Loss: 1.051 | Val Balanced Accuracy: 0.64
Epoch 97 | Train Loss: 0.130 | Val Loss: 1.267 | Val Balanced Accuracy: 0.57
Epoch 98 | Train Loss: 0.166 | Val Loss: 0.944 | Val Balanced Accuracy: 0.59
Epoch 99 | Train Loss: 0.072 | Val Loss: 1.291 | Val Balanced Accuracy: 0.56
Epoch 100 | Train Loss: 0.041 | Val Loss: 1.092 | Val Balanced Accuracy: 0.62

```

```
[ ] torch.save(model.state_dict(), "/content/drive/MyDrive/unet_octclassifier_latest.pth")
```

```

testset = OCTDataset(subset='test', transform=test_transform, device=device)
testloader = DataLoader(testset, batch_size=batch_size, shuffle=False, num_workers=2)

#Load model from pth file corresponding to maximum accuracy obtained from all U-Net based experiments.
#This code is commented while trying new methods and only used to lock the model weights once satisfied with trained model.
model.load_state_dict(torch.load("/content/drive/MyDrive/unet_octclassifier_0dot59.pth"))

#evaluate the model on the test set
model.eval()

```

```

# turn off gradients for evaluation
true_labels = []
pred_labels = []
with torch.no_grad():
    for i, (inputs, labels) in enumerate(testloader):
        print('Testing for batch: ',str(i))
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        pred_labels.extend(predicted.cpu().numpy())
        true_labels.extend(labels.cpu().numpy())

# confusion matrix
conf_mat = confusion_matrix(true_labels, pred_labels)

# compute the specificity and sensitivity
sensitivity = []
specificity = []
num_classes = conf_mat.shape[0]
for i in range(num_classes):
    tp = conf_mat[i, i]
    fp = np.sum(conf_mat[:, i]) - tp
    fn = np.sum(conf_mat[i, :]) - tp
    tn = np.sum(conf_mat) - (tp + fp + fn)

    sensitivity.append(tp / (tp + fn))
    specificity.append(tn / (tn + fp))

# compute the balanced accuracy
balanced_accuracy = balanced_accuracy_score(true_labels, pred_labels)
f1 = f1_score(true_labels, pred_labels, average='weighted')

# print the balanced accuracy
print("Balanced accuracy: ", balanced_accuracy)
print("F1 score: ", f1)
for i in range(num_classes):
    print(f'Sensitivity for class {i}: {sensitivity[i]:.2f}')
    print(f'Specificity for class {i}: {specificity[i]:.2f}')

```

```

Testing for batch: 0
Testing for batch: 1
Testing for batch: 2
Testing for batch: 3
Testing for batch: 4
Testing for batch: 5
Testing for batch: 6
Testing for batch: 7
Testing for batch: 8
Testing for batch: 9
Testing for batch: 10
Balanced accuracy: 0.5816583953680728
F1 score: 0.5893705052293798
Sensitivity for class 0: 0.65
Specificity for class 0: 0.72
Sensitivity for class 1: 0.57
Specificity for class 1: 0.70
Sensitivity for class 2: 0.52
Specificity for class 2: 0.63

```

specificity for class 2: 0.92

Colab paid products - Cancel contracts here

✓ 3s completed at 3:09 AM

