Deep Learning Homework 6 (Spring 2023)

This code is provided for Deep Learning class (601.482/682) Homework 6. For ease of implementation, we recommend working entire in Google Colaboratory.

Imports

In [ ]:
```
## Mount Google Drive Data (If using Google Colaboratory)
try:
    from google.colab import drive
    drive.mount('/content/gdrive')
except:
    print("Mounting Failed.")
```

```
Mounted at /content/gdrive
```

In [ ]:
```
## Standard Library
import os
import json

## External Libraries
import numpy as np
import torch
import torch.nn as nn
from torchvision import transforms
from torch.autograd import Variable
import torch.nn.functional as functional
from torch.utils.data import Dataset, DataLoader
from skimage import io
import matplotlib.pyplot as plt
```

Problem 1: Unsupervised Pre-training

Training Hyperparameters

These are recommended hyperparameters - please feel free to use what works for you. Batch size can be changed if it does not match your memory, please state your batch step_size in your report.

Dataset is available at: https://livejohnshopkins-my.sharepoint.com/:u:/g/personal/yshen92_jh_edu/EcTxWAXsAhtDiv3vUxCTF8gBgAARCUvvKthb3s-pEExyMg

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

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

In [ ]:
```python
## Batch Size
train_batch_size = 10
validation_batch_size = 10

## Learning Rate
learning_rate = 0.001

# Epochs (Consider setting high and implementing early stopping)
num_epochs = 200
```

In [ ]:
```python
# from google.colab import drive
# drive.flush_and_unmount()
# drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

In [ ]:
```python
import os

# to check if the path already exist
if os.path.exists("/content/drive/MyDrive/HW6_data"):
    print("Directory exists. No need to unzip again.")
else:
    print("Directory not found. You might need to unzip again.")
```

```
Directory exists. No need to unzip again.
```

Data Paths

In [ ]:
```
# import zipfile

# with zipfile.ZipFile(data_dir, 'r') as zip_ref:
#     zip_ref.extractall("/content/gdrive/MyDrive/")
```

In [ ]:
```
# General Data Directory ##TODO: Please fill in the appropriate directory
# data_dir = "/content/gdrive/MyDrive/hw6_data.zip"
data_dir = '/content/drive/MyDrive/HW6_data'

## Segmentation + Colorization Paths
segmentation_data_dir = f"{data_dir}/segmentation/"
colorization_data_dir = f"{data_dir}/colorization/"

# Mask JSON
mask_json = f"{data_dir}/mapping.json"
```

In [ ]:
```
segmentation_data_dir
```

Out[ ]:
```
'/content/drive/MyDrive/HW6_data/segmentation/'
```

Data Loaders

We have provided you with some preprocessing code for the images but you should feel free to modify the class however you please to support your training schema. In the very least, you will have to modify the dataloader to support loading of the colorization dataset.

In [ ]:
```
## Image Transforms
img_transform = transforms.Compose([
        transforms.ToTensor(),
])

## Image Dataloader
class ImageDataset(Dataset):

    """
    ImageDataset
```

```
    """

    def __init__(self,
                 input_dir,
                 op,
                 mask_json_path,
                 transforms=None,
                 is_colorization = False):
        """
        ##TODO: Add support for colorization dataset

        Args:
            input_dir (str): Path to either colorization or segmentation directory
            op (str): One of "train", "val", or "test" signifying the desired split
            mask_json_path (str): Path to mapping.json file
            transforms (list or None): Image transformations to apply upon loading.
        """
        self.transform = transforms
        self.op = op
        # Set the colorization flag
        self.is_colorization = is_colorization
        with open(mask_json_path, 'r') as f:
            self.mask = json.load(f)
        self.mask_num = len(self.mask)  # There are 6 categories: grey, dark grey, and black
        self.mask_value = [int(value) for key, value in self.mask.items() if not key.startswith("_comment"
        self.mask_value.sort()
        if self.is_colorization == False:
            if self.op == 'train':
                self.data_dir = os.path.join(input_dir, 'train')
            elif self.op == 'val':
                self.data_dir = os.path.join(input_dir, 'validation')
            elif self.op == 'test':
                self.data_dir = os.path.join(input_dir, 'test')
        else:
            if self.op == 'train':
                self.data_dir = os.path.join(input_dir, 'train_cor')
            elif self.op == 'val':
                self.data_dir = os.path.join(input_dir, 'validation_cor')


    def __len__(self):
        """
```

```
        """
        return len(next(os.walk(self.data_dir))[1])

    def __getitem__(self,
                    idx):
        """

        """
        ## Load Image and Parse Properties
        if self.is_colorization == False:
            img_name = str(idx) + '_input.jpg'
            mask_name = str(idx) + '_mask.png'
        else:
            img_name = str(idx) + '_gray.jpg'
            mask_name = str(idx) + '_input.jpg'

        img = io.imread(os.path.join(self.data_dir, str(idx), img_name))
        mask = io.imread(os.path.join(self.data_dir, str(idx), mask_name))



        ## determine whether it is gray mask or RGB mask
        # there is no channel(c) for gray mask
        # c = 3 for RGB
        # h: height
        # w: width
        if len(mask.shape) == 2:
            h, w  = mask.shape
        elif len(mask.shape) == 3:
            h, w, c = mask.shape
        ## Convert grey-scale label to one-hot encoding
        if self.is_colorization == False:
            new_mask = np.zeros((h, w, self.mask_num))
            for idx in range(self.mask_num):
                #if the mask has 3 dimension use this code
                new_mask[:, :, idx] = mask[:,:,0] == self.mask_value[idx]
        else:
            new_mask = mask
            #if the mask has 1 dimension use the code below
            #new_mask[:, :, idx] = mask == self.mask_value[idx]
        ## Transform image and mask
```

```
        if self.transform:
            img, new_mask = self.img_transform(img, new_mask)
        # ## Use dictionary to output
        # sample = {'img': img, 'mask': mask}
        # return sample
        # 这里的mask已经转化为one-hot vector
        return img, new_mask

    def img_transform(self,
                      img,
                      mask):
        """

        """
        ## Apply Transformations to Image and Mask
        img = self.transform(img)
        mask = self.transform(mask)
        return img, mask

    def img_transform_single(self, img):
      return self.transform(img)
```

Model Architecture

Finish building the U-net architecture below.

In [ ]:
```
## Functions for adding the convolution layer
def add_conv_stage(dim_in,
                   dim_out,
                   kernel_size=3,
                   stride=1,
                   padding=1,
                   bias=True,
                   useBN=True):
    """

    """
    # Use batch normalization
    if useBN:
        return nn.Sequential(
```

```
          nn.Conv2d(dim_in, dim_out, kernel_size=kernel_size, stride=stride, padding=padding, bias=bias),
          nn.BatchNorm2d(dim_out),
          nn.LeakyReLU(0.1),
          nn.Conv2d(dim_out, dim_out, kernel_size=kernel_size, stride=stride, padding=padding, bias=bias),
          nn.BatchNorm2d(dim_out),
          nn.LeakyReLU(0.1)
        )
    # No batch normalization
    else:
        return nn.Sequential(
          nn.Conv2d(dim_in, dim_out, kernel_size=kernel_size, stride=stride, padding=padding, bias=bias),
          nn.ReLU(),
          nn.Conv2d(dim_out, dim_out, kernel_size=kernel_size, stride=stride, padding=padding, bias=bias),
          nn.ReLU()
        )

## Upsampling
def upsample(ch_coarse,
             ch_fine):
    """

    """
    return nn.Sequential(
                    nn.ConvTranspose2d(ch_coarse, ch_fine, 4, 2, 1, bias=False),
                    nn.ReLU())


# U-Net
class UNET(nn.Module):

    """

    """
    def __init__(self, n_classes, input_channels_num = 3, useBN=True):
        """
        Args:
            n_classes (int): Number of classes
            useBN (bool): Turn Batch Norm on or off. (Hint: Using BatchNorm might help you achieve better p
        """
        super(UNET, self).__init__()
        # Downgrade stages
        self.conv1 = add_conv_stage(input_channels_num, 32, useBN=useBN)
```

```
        self.conv2 = add_conv_stage(32, 64, useBN=useBN)
        self.conv3 = add_conv_stage(64, 128, useBN=useBN)
        self.conv4 = add_conv_stage(128, 256, useBN=useBN)
        # Upgrade stages
        self.conv3m = add_conv_stage(256, 128, useBN=useBN)
        self.conv2m = add_conv_stage(128,  64, useBN=useBN)
        self.conv1m = add_conv_stage( 64,  32, useBN=useBN)
        # Maxpool
        self.max_pool = nn.MaxPool2d(2)
        # Upsample layers
        self.upsample43 = upsample(256, 128)
        self.upsample32 = upsample(128,  64)
        self.upsample21 = upsample(64 ,  32)
        # weight initialization
        # You can have your own weight intialization. This is just an example.
        for m in self.modules():
            if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):
                if m.bias is not None:
                    m.bias.data.zero_()
        #TODO: Design your last layer & activations
        self.final = nn.Conv2d(32, n_classes, 1)

        if n_classes == 3:
          self.final_act = nn.Sigmoid()
        else:
          self.final_act = nn.Softmax(dim = 1)


    def forward(self, x):
        """
        Forward pass
        """
        conv1_out = self.conv1(x)
        conv2_out = self.conv2(self.max_pool(conv1_out))
        conv3_out = self.conv3(self.max_pool(conv2_out))
        conv4_out = self.conv4(self.max_pool(conv3_out))

        conv4m_out_ = torch.cat((self.upsample43(conv4_out), conv3_out), 1)
        conv3m_out  = self.conv3m(conv4m_out_)

        conv3m_out_ = torch.cat((self.upsample32(conv3m_out), conv2_out), 1)
        conv2m_out  = self.conv2m(conv3m_out_)
```

```
        conv2m_out_ = torch.cat((self.upsample21(conv2m_out), conv1_out), 1)
        conv1m_out  = self.conv1m(conv2m_out_)

        #TODO: Design your last layer & activations
        final_out = self.final(conv1m_out)

        return self.final_act(final_out)
```

In [ ]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Current device: {device}")
```

```
Current device: cuda
```

In [ ]:
```
img = io.imread('/content/drive/MyDrive/HW6_data/colorization/validation_cor/1/1_gray.jpg')
print(img.shape)
```

```
(256, 320)
```

DICE Score and DICE Loss

Finish implementing the DICE score function below and then write a Dice Loss function that you can use to update your model weights.

In [ ]:
```
##TODO: Finish implementing the multi-class DICE score function
def dice_score_image(prediction, target, n_classes):
    '''
      computer the mean dice score for a single image

      Reminders: A false positive is a result that indicates a given condition exists, when it does not
               A false negative is a test result that indicates that a condition does not hold, while in fa
      Args:
          prediction (tensor): predictied labels of the image
          target (tensor): ground truth of the image
          n_classes (int): number of classes

      Returns:
          m_dice (float): Mean dice score over classes
    '''
    ## Should test image one by one
    assert prediction.shape[0] == 1 #This line can not be deleted
```

```
    ## TODO: Compute Dice Score for Each Class. Compute Mean Dice Score over Classes.
    dice_scores = np.zeros(n_classes)
    target = torch.argmax(target, dim = 1)
    for cl in range(n_classes):
        pred_cl = (prediction == cl).float()
        true_cl = (target == cl).float()
        TP = torch.sum(pred_cl * true_cl)
        FP = torch.sum(pred_cl*(1 - true_cl))
        FN = torch.sum((1 - pred_cl) * true_cl)
        epsilon = 1e-7
        #When there is no ground truth of the class in this image
        #Give 1 dice score if False Positive pixel number is 0,
        #give 0 dice score if False Positive pixel number is not 0 (> 0).
        # if FP == 0:
        #   dice_scores.append(1.0)
        # else:
        #   dice_scores.append( (2 * TP + epsilon ) / (2 * TP + FP + FN + epsilon ) )

        if TP == 0 or FN == 0:
          dice_scores[cl] = 1.0 if FP == 0 else 0.0
        else:
          dice_scores[cl] = (2 * TP + epsilon) / (2 * TP + FP + FN + epsilon)

    return dice_scores.mean()



def dice_score_dataset(model, dataloader, num_classes, use_gpu=False):
    """
    Compute the mean dice score on a set of data.

    Note that multiclass dice score can be defined as the mean over classes of binary
    dice score. Dice score is computed per image. Mean dice score over the dataset is the dice
    score averaged across all images.

    Reminders: A false positive is a result that indicates a given condition exists, when it does not
               A false negative is a test result that indicates that a condition does not hold, while in fa

    Args:
        model (UNET class): Your trained model
        dataloader (DataLoader): Dataset for evaluation
        num_classes (int): Number of classes
```

```
    Returns:
        m_dice (float): Mean dice score over the input dataset
    """
    ## Number of Batches and Cache over Dataset
    n_batches = len(dataloader)
    scores = np.zeros(n_batches)
    ## Evaluate
    model.eval()
    idx = 0
    with torch.no_grad():
      for data in dataloader:
          ## Format Data
          img, target = data
          if use_gpu:
              img = img.cuda()
              target = target.cuda()
          ## Make Predictions
          out = model(img)
          n_classes = out.shape[1]
          prediction = torch.argmax(out, dim = 1)
          scores[idx] = dice_score_image(prediction, target, n_classes)
          idx += 1
    ## Average Dice Score Over Images
    m_dice = scores.mean()
    return m_dice


## TODO: Implement DICE loss,
#  It should conform to to how we computer the dice score.
class DICELoss(nn.Module):
    def __init__(self, n_classes):
      super(DICELoss, self).__init__()
      self.n_classes = n_classes
    def forward(self, prediction, target):
      #probs = torch.nn.functional.softmax(prediction, dim=1)
      probs = prediction
      dice_loss = 0.0
      for cl in range(self.n_classes):
        #pred_cl = (prediciton_classes == cl).float()
        # true_cl = (target == cl).float()
        pred_cl = probs[:, cl, :, :]
```

```
        true_cl = target[:, cl, :, :]
        error = 1e-7

        TP = torch.sum(pred_cl * true_cl)
        FP = torch.sum(pred_cl * (1 - true_cl))
        FN = torch.sum((1 - pred_cl) * true_cl)

        # Add dice loss for the class to the total dice loss
        dice_score = (2 * TP + error) / (2 * TP + FP + FN + error)
        dice_loss += 1 - dice_score
      return dice_loss / self.n_classes
```

In [ ]:
```
import random

def set_seed(seed_value):
    """Set seed for reproducibility."""
    random.seed(seed_value)
    np.random.seed(seed_value)
    torch.manual_seed(seed_value)
    torch.cuda.manual_seed(seed_value)
    torch.cuda.manual_seed_all(seed_value)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
seed_value = 12345
set_seed(seed_value)
```

Training Procedure (Segmentation)

In [ ]:
```
import copy
```

In [ ]:
```
## Initialize your unet
n_classes = len(json.load(open(mask_json)))
model = UNET(n_classes)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

## Initialize Dataloaders
train_dataset=ImageDataset(input_dir=segmentation_data_dir, op="train", mask_json_path=mask_json, transforr
validation_dataset=ImageDataset(input_dir=segmentation_data_dir, op="val", mask_json_path=mask_json, trans
test_dataset=ImageDataset(input_dir=segmentation_data_dir, op="test", mask_json_path=mask_json, transforms
```

```
train_dataloader = DataLoader(train_dataset, batch_size=train_batch_size, shuffle=True)
validation_dataloader = DataLoader(validation_dataset, batch_size=validation_batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=1, shuffle=False)
## Initialize Optimizer and Learning Rate Scheduler
optimizer = torch.optim.Adam(model.parameters(),lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

# implement early stopping
patience = 10
best_val_loss = float('inf')
counter_early_stop = 0
early_stop = False
training_loss_values = []
validation_loss_values = []
print("Start Training...")
for epoch in range(num_epochs):
    ########################### Training #####################################
    print("\nEPOCH " +str(epoch+1)+" of "+str(num_epochs)+"\n")
    # TODO: Design your own training section
    model.train()
    train_loss = 0.0
    for images, masks in train_dataloader:
      images = images.to(device)
      masks = masks.to(device)

      optimizer.zero_grad()
      outputs = model(images)
      # print(outputs.requires_grad)
      loss = DICELoss(n_classes)(outputs, masks)
      loss.backward()
      optimizer.step()
      train_loss += loss.item()

    train_loss = train_loss / len(train_dataloader)
    training_loss_values.append(train_loss)

    ########################### Validation #####################################
    # TODO: Design your own validation section
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for images, masks in validation_dataloader:
```

```
            images = images.to(device)
            masks = masks.to(device)
            outputs = model(images)
            # probs = torch.softmax(outputs, dim = 1)
            loss = DICELoss(n_classes).forward(outputs, masks)
            val_loss += loss.item()
    val_loss = val_loss / len(validation_dataloader)
    validation_loss_values.append(val_loss)
    print(f"Epoch {epoch+1}, Training loss: {train_loss:.4f}, Validation Loss: {val_loss:.4f}")
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        best_model = copy.deepcopy(model.state_dict())  # Save the best model
        counter_early_stop = 0  # Reset counter
        print("Validation loss decreased, saving model...")
    else:
        counter_early_stop += 1
        print(f"Validation loss did not decrease, counter: {counter_early_stop}/{patience}")

    if counter_early_stop >= patience or epoch == num_epochs - 1:
        early_stop = True
        break


    scheduler.step()

plt.figure(figsize=(10, 5))
plt.plot(training_loss_values, label='Training Loss')
plt.plot(validation_loss_values, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs')
plt.legend()
plt.show()
```

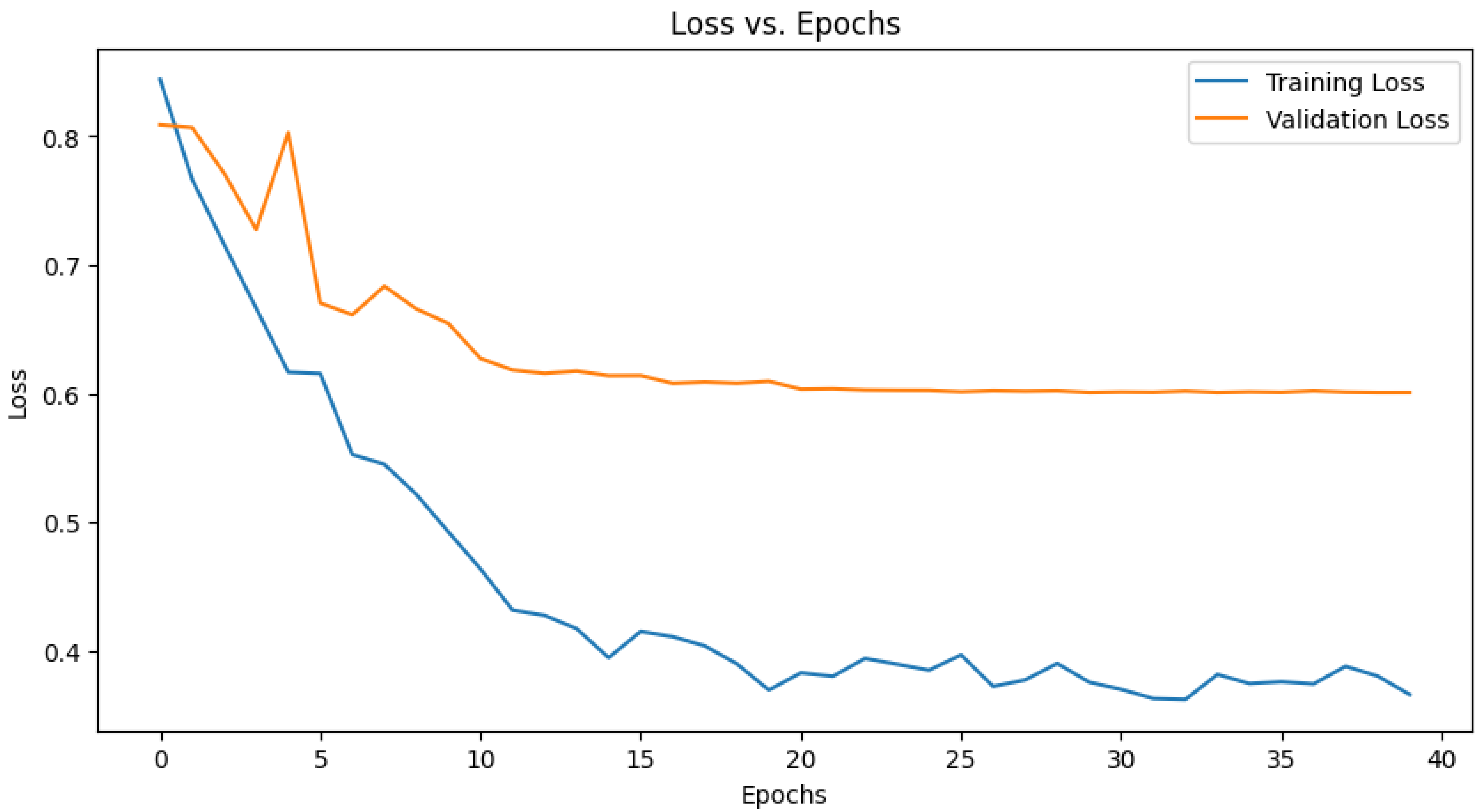
```
Start Training...

EPOCH 1 of 200

Epoch 1, Training loss: 0.8444, Validation Loss: 0.8090
Validation loss decreased, saving model...

EPOCH 2 of 200

Epoch 2, Training loss: 0.7666, Validation Loss: 0.8068
Validation loss decreased, saving model...

EPOCH 3 of 200

Epoch 3, Training loss: 0.7158, Validation Loss: 0.7713
Validation loss decreased, saving model...

EPOCH 4 of 200

Epoch 4, Training loss: 0.6663, Validation Loss: 0.7274
Validation loss decreased, saving model...

EPOCH 5 of 200

Epoch 5, Training loss: 0.6166, Validation Loss: 0.8029
Validation loss did not decrease, counter: 1/10

EPOCH 6 of 200

Epoch 6, Training loss: 0.6155, Validation Loss: 0.6704
Validation loss decreased, saving model...

EPOCH 7 of 200

Epoch 7, Training loss: 0.5525, Validation Loss: 0.6612
Validation loss decreased, saving model...

EPOCH 8 of 200

Epoch 8, Training loss: 0.5449, Validation Loss: 0.6834
Validation loss did not decrease, counter: 1/10
```

```
EPOCH 9 of 200

Epoch 9, Training loss: 0.5215, Validation Loss: 0.6658
Validation loss did not decrease, counter: 2/10

EPOCH 10 of 200

Epoch 10, Training loss: 0.4923, Validation Loss: 0.6545
Validation loss decreased, saving model...

EPOCH 11 of 200

Epoch 11, Training loss: 0.4636, Validation Loss: 0.6272
Validation loss decreased, saving model...

EPOCH 12 of 200

Epoch 12, Training loss: 0.4316, Validation Loss: 0.6183
Validation loss decreased, saving model...

EPOCH 13 of 200

Epoch 13, Training loss: 0.4274, Validation Loss: 0.6157
Validation loss decreased, saving model...

EPOCH 14 of 200

Epoch 14, Training loss: 0.4171, Validation Loss: 0.6175
Validation loss did not decrease, counter: 1/10

EPOCH 15 of 200

Epoch 15, Training loss: 0.3945, Validation Loss: 0.6139
Validation loss decreased, saving model...

EPOCH 16 of 200

Epoch 16, Training loss: 0.4149, Validation Loss: 0.6140
Validation loss did not decrease, counter: 1/10

EPOCH 17 of 200
```

```
Epoch 17, Training loss: 0.4108, Validation Loss: 0.6079
Validation loss decreased, saving model...

EPOCH 18 of 200

Epoch 18, Training loss: 0.4037, Validation Loss: 0.6090
Validation loss did not decrease, counter: 1/10

EPOCH 19 of 200

Epoch 19, Training loss: 0.3898, Validation Loss: 0.6080
Validation loss did not decrease, counter: 2/10

EPOCH 20 of 200

Epoch 20, Training loss: 0.3694, Validation Loss: 0.6095
Validation loss did not decrease, counter: 3/10

EPOCH 21 of 200

Epoch 21, Training loss: 0.3827, Validation Loss: 0.6034
Validation loss decreased, saving model...

EPOCH 22 of 200

Epoch 22, Training loss: 0.3801, Validation Loss: 0.6037
Validation loss did not decrease, counter: 1/10

EPOCH 23 of 200

Epoch 23, Training loss: 0.3939, Validation Loss: 0.6027
Validation loss decreased, saving model...

EPOCH 24 of 200

Epoch 24, Training loss: 0.3894, Validation Loss: 0.6024
Validation loss decreased, saving model...

EPOCH 25 of 200

Epoch 25, Training loss: 0.3848, Validation Loss: 0.6024
Validation loss decreased, saving model...
```

```
EPOCH 26 of 200

Epoch 26, Training loss: 0.3967, Validation Loss: 0.6013
Validation loss decreased, saving model...

EPOCH 27 of 200

Epoch 27, Training loss: 0.3722, Validation Loss: 0.6022
Validation loss did not decrease, counter: 1/10

EPOCH 28 of 200

Epoch 28, Training loss: 0.3772, Validation Loss: 0.6018
Validation loss did not decrease, counter: 2/10

EPOCH 29 of 200

Epoch 29, Training loss: 0.3900, Validation Loss: 0.6022
Validation loss did not decrease, counter: 3/10

EPOCH 30 of 200

Epoch 30, Training loss: 0.3755, Validation Loss: 0.6007
Validation loss decreased, saving model...

EPOCH 31 of 200

Epoch 31, Training loss: 0.3698, Validation Loss: 0.6012
Validation loss did not decrease, counter: 1/10

EPOCH 32 of 200

Epoch 32, Training loss: 0.3628, Validation Loss: 0.6009
Validation loss did not decrease, counter: 2/10

EPOCH 33 of 200

Epoch 33, Training loss: 0.3621, Validation Loss: 0.6019
Validation loss did not decrease, counter: 3/10

EPOCH 34 of 200
```

```
Epoch 34, Training loss: 0.3815, Validation Loss: 0.6007
Validation loss did not decrease, counter: 4/10

EPOCH 35 of 200

Epoch 35, Training loss: 0.3744, Validation Loss: 0.6013
Validation loss did not decrease, counter: 5/10

EPOCH 36 of 200

Epoch 36, Training loss: 0.3759, Validation Loss: 0.6009
Validation loss did not decrease, counter: 6/10

EPOCH 37 of 200

Epoch 37, Training loss: 0.3741, Validation Loss: 0.6021
Validation loss did not decrease, counter: 7/10

EPOCH 38 of 200

Epoch 38, Training loss: 0.3878, Validation Loss: 0.6011
Validation loss did not decrease, counter: 8/10

EPOCH 39 of 200

Epoch 39, Training loss: 0.3802, Validation Loss: 0.6007
Validation loss did not decrease, counter: 9/10

EPOCH 40 of 200

Epoch 40, Training loss: 0.3659, Validation Loss: 0.6007
Validation loss did not decrease, counter: 10/10
```



In [ ]:
```
test_dice_score = dice_score_dataset(model, test_dataloader, n_classes, use_gpu=device.type == 'cuda')
print(f"Epoch {epoch+1}, DICE score on the test set: {test_dice_score:.4f}")
```

```
Epoch 40, DICE score on the test set: 0.5527
```

In [ ]:
```
from torchvision.transforms import functional as F

class RandomHorizontalFlip(object):

  def __init__(self, prob):
      self.prob = prob

  def __call__(self, img):
    if torch.rand(1).item() < self.prob:
```

```
            return torch.flip(img, [2])
        return img


class RandomRotation(object):

    def __init__(self, degrees):
        self.degrees = degrees

    def __call__(self, img):
        angle = torch.FloatTensor(1).uniform_(-self.degrees, self.degrees).item()
        return F.rotate(img, angle)


##prob = 0.25, degrees = 25, dice score = 0.5623



train_img_transform=  transforms.Compose([
      transforms. ToTensor (),
      RandomHorizontalFlip (prob=0.25),
      RandomRotation (degrees=15),
])
```

In [ ]:

```
## Initialize your unet
n_classes = len(json.load(open(mask_json)))
model = UNET(n_classes)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

## Initialize Dataloaders
train_dataset=ImageDataset(input_dir=segmentation_data_dir, op="train", mask_json_path=mask_json, transforn
validation_dataset=ImageDataset(input_dir=segmentation_data_dir, op="val", mask_json_path=mask_json, trans
test_dataset=ImageDataset(input_dir=segmentation_data_dir, op="test", mask_json_path=mask_json, transforms=
train_dataloader = DataLoader(train_dataset, batch_size=train_batch_size, shuffle=True)
validation_dataloader = DataLoader(validation_dataset, batch_size=validation_batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=1, shuffle=False)
## Initialize Optimizer and Learning Rate Scheduler
optimizer = torch.optim.Adam(model.parameters(),lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

# implement early stopping
```

```
patience = 10  # How many epochs to wait after last time validation loss improved.
best_val_loss = float('inf')
counter_early_stop = 0
early_stop = False
training_loss_values = []
validation_loss_values = []
print("Start Training...")
for epoch in range(num_epochs):
    ########################### Training #####################################
    print("\nEPOCH " +str(epoch+1)+" of "+str(num_epochs)+"\n")
    # TODO: Design your own training section
    model.train()
    train_loss = 0.0

    for images, masks in train_dataloader:

      # if torch.rand(1).item() < 0.5:
      #   images = torch.flip(images, [2])
      #   masks = torch.flip(masks, [2])
      # if torch.rand(1).item() < 0.5:
      #   images = torch.flip(images, [3])
      #   masks = torch.flip(masks, [3])


      images = images.to(device)
      masks = masks.to(device)
      optimizer.zero_grad()
      outputs = model(images)
      # print(outputs.requires_grad)
      loss = DICELoss(n_classes)(outputs, masks)
      loss.backward()
      optimizer.step()
      train_loss += loss.item()

    train_loss = train_loss / len(train_dataloader)
    training_loss_values.append(train_loss)

    ########################### Validation ####################################
    # TODO: Design your own validation section
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
```

```
        for images, masks in validation_dataloader:
            images = images.to(device)
            masks = masks.to(device)
            outputs = model(images)
            # probs = torch.softmax(outputs, dim = 1)
            loss = DICELoss(n_classes).forward(outputs, masks)
            val_loss += loss.item()
    val_loss = val_loss / len(validation_dataloader)
    validation_loss_values.append(val_loss)
    print(f"Epoch {epoch+1}, Training loss: {train_loss:.4f}, Validation Loss: {val_loss:.4f}")
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        best_model = copy.deepcopy(model.state_dict())  # Save the best model
        counter_early_stop = 0  # Reset counter
        print("Validation loss decreased, saving model...")
    else:
        counter_early_stop += 1
        print(f"Validation loss did not decrease, counter: {counter_early_stop}/{patience}")

    if counter_early_stop >= patience or epoch == num_epochs -1:
        early_stop = True
        break


    scheduler.step()

plt.figure(figsize=(10, 5))
plt.plot(training_loss_values, label='Training Loss')
plt.plot(validation_loss_values, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs')
plt.legend()
plt.show()
```

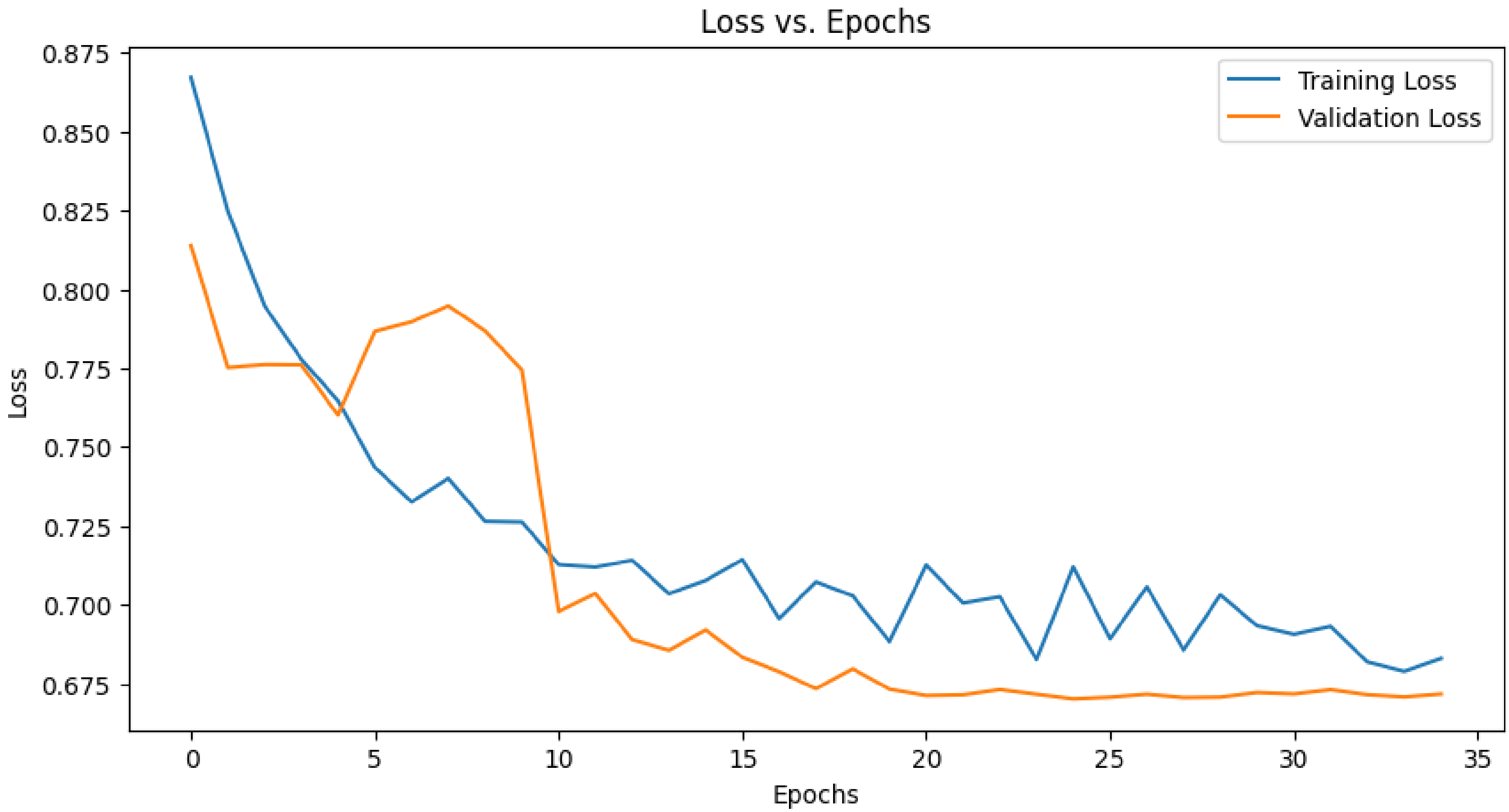
```
Start Training...

EPOCH 1 of 200

Epoch 1, Training loss: 0.8672, Validation Loss: 0.8139
Validation loss decreased, saving model...

EPOCH 2 of 200

Epoch 2, Training loss: 0.8249, Validation Loss: 0.7752
Validation loss decreased, saving model...

EPOCH 3 of 200

Epoch 3, Training loss: 0.7948, Validation Loss: 0.7762
Validation loss did not decrease, counter: 1/10

EPOCH 4 of 200

Epoch 4, Training loss: 0.7779, Validation Loss: 0.7761
Validation loss did not decrease, counter: 2/10

EPOCH 5 of 200

Epoch 5, Training loss: 0.7647, Validation Loss: 0.7601
Validation loss decreased, saving model...

EPOCH 6 of 200

Epoch 6, Training loss: 0.7436, Validation Loss: 0.7867
Validation loss did not decrease, counter: 1/10

EPOCH 7 of 200

Epoch 7, Training loss: 0.7326, Validation Loss: 0.7897
Validation loss did not decrease, counter: 2/10

EPOCH 8 of 200

Epoch 8, Training loss: 0.7401, Validation Loss: 0.7948
Validation loss did not decrease, counter: 3/10
```

```
EPOCH 9 of 200

Epoch 9, Training loss: 0.7265, Validation Loss: 0.7869
Validation loss did not decrease, counter: 4/10

EPOCH 10 of 200

Epoch 10, Training loss: 0.7263, Validation Loss: 0.7745
Validation loss did not decrease, counter: 5/10

EPOCH 11 of 200

Epoch 11, Training loss: 0.7127, Validation Loss: 0.6979
Validation loss decreased, saving model...

EPOCH 12 of 200

Epoch 12, Training loss: 0.7119, Validation Loss: 0.7036
Validation loss did not decrease, counter: 1/10

EPOCH 13 of 200

Epoch 13, Training loss: 0.7141, Validation Loss: 0.6890
Validation loss decreased, saving model...

EPOCH 14 of 200

Epoch 14, Training loss: 0.7035, Validation Loss: 0.6856
Validation loss decreased, saving model...

EPOCH 15 of 200

Epoch 15, Training loss: 0.7077, Validation Loss: 0.6920
Validation loss did not decrease, counter: 1/10

EPOCH 16 of 200

Epoch 16, Training loss: 0.7143, Validation Loss: 0.6833
Validation loss decreased, saving model...

EPOCH 17 of 200
```

```
Epoch 17, Training loss: 0.6956, Validation Loss: 0.6788
Validation loss decreased, saving model...

EPOCH 18 of 200

Epoch 18, Training loss: 0.7073, Validation Loss: 0.6734
Validation loss decreased, saving model...

EPOCH 19 of 200

Epoch 19, Training loss: 0.7028, Validation Loss: 0.6797
Validation loss did not decrease, counter: 1/10

EPOCH 20 of 200

Epoch 20, Training loss: 0.6883, Validation Loss: 0.6733
Validation loss decreased, saving model...

EPOCH 21 of 200

Epoch 21, Training loss: 0.7126, Validation Loss: 0.6712
Validation loss decreased, saving model...

EPOCH 22 of 200

Epoch 22, Training loss: 0.7006, Validation Loss: 0.6714
Validation loss did not decrease, counter: 1/10

EPOCH 23 of 200

Epoch 23, Training loss: 0.7025, Validation Loss: 0.6731
Validation loss did not decrease, counter: 2/10

EPOCH 24 of 200

Epoch 24, Training loss: 0.6826, Validation Loss: 0.6716
Validation loss did not decrease, counter: 3/10

EPOCH 25 of 200

Epoch 25, Training loss: 0.7120, Validation Loss: 0.6702
Validation loss decreased, saving model...
```

```
EPOCH 26 of 200

Epoch 26, Training loss: 0.6892, Validation Loss: 0.6707
Validation loss did not decrease, counter: 1/10

EPOCH 27 of 200

Epoch 27, Training loss: 0.7057, Validation Loss: 0.6716
Validation loss did not decrease, counter: 2/10

EPOCH 28 of 200

Epoch 28, Training loss: 0.6857, Validation Loss: 0.6706
Validation loss did not decrease, counter: 3/10

EPOCH 29 of 200

Epoch 29, Training loss: 0.7031, Validation Loss: 0.6707
Validation loss did not decrease, counter: 4/10

EPOCH 30 of 200

Epoch 30, Training loss: 0.6933, Validation Loss: 0.6722
Validation loss did not decrease, counter: 5/10

EPOCH 31 of 200

Epoch 31, Training loss: 0.6906, Validation Loss: 0.6717
Validation loss did not decrease, counter: 6/10

EPOCH 32 of 200

Epoch 32, Training loss: 0.6931, Validation Loss: 0.6731
Validation loss did not decrease, counter: 7/10

EPOCH 33 of 200

Epoch 33, Training loss: 0.6819, Validation Loss: 0.6715
Validation loss did not decrease, counter: 8/10

EPOCH 34 of 200
```

```
Epoch 34, Training loss: 0.6789, Validation Loss: 0.6708
Validation loss did not decrease, counter: 9/10

EPOCH 35 of 200

Epoch 35, Training loss: 0.6830, Validation Loss: 0.6717
Validation loss did not decrease, counter: 10/10
```



In [ ]:
```python
test_dice_score = dice_score_dataset(model, test_dataloader, n_classes, use_gpu=device.type == 'cuda')
print(f"Epoch {epoch+1}, DICE score on the test set: {test_dice_score:.4f}")
```

```
Epoch 35, DICE score on the test set: 0.4572
```

Training Procedure: Colorization Pre-training

Complete the rest of this problem in the cells below.

In [ ]:
```
## Batch Size
train_batch_size = 30
validation_batch_size = 30

## Learning Rate
learning_rate = 0.001

# Epochs (Consider setting high and implementing early stopping)
num_epochs = 200
```

In [ ]:
```
## Image Transforms
img_transform = transforms.Compose([
        transforms.ToTensor(),
])

## Image Dataloader
class ImageDataset(Dataset):

    """
    ImageDataset
    """

    def __init__(self,
                 input_dir,
                 op,
                 mask_json_path,
                 transforms=None,
                 is_colorization = False):
        """
        ##TODO: Add support for colorization dataset

        Args:
            input_dir (str): Path to either colorization or segmentation directory
            op (str): One of "train", "val", or "test" signifying the desired split
            mask_json_path (str): Path to mapping.json file
```

```
            transforms (list or None): Image transformations to apply upon loading.
        """
        self.transform = transforms
        self.op = op
        # Set the colorization flag
        self.is_colorization = is_colorization
        with open(mask_json_path, 'r') as f:
            self.mask = json.load(f)
        self.mask_num = len(self.mask)  # There are 6 categories: grey, dark grey, and black
        self.mask_value = [int(value) for key, value in self.mask.items() if not key.startswith("_comment"
        self.mask_value.sort()
        if self.is_colorization == False:
            if self.op == 'train':
                self.data_dir = os.path.join(input_dir, 'train')
            elif self.op == 'val':
                self.data_dir = os.path.join(input_dir, 'validation')
            elif self.op == 'test':
                self.data_dir = os.path.join(input_dir, 'test')
        else:
            if self.op == 'train':
                self.data_dir = os.path.join(input_dir, 'train_cor')
            elif self.op == 'val':
                self.data_dir = os.path.join(input_dir, 'validation_cor')

    def __len__(self):
        if self.op == 'train':
            return 1584
        elif self.op == 'val':
            return 50
        else:
            raise ValueError("Invalid split option: {}".format(self.op))


    def __getitem__(self,
                    idx):
        """

        """
        ## Load Image and Parse Properties
        if self.is_colorization == False:
            img_name = str(idx) + '_input.jpg'
            mask_name = str(idx) + '_mask.png'
```

```
        else:
            img_name = str(idx) + '_gray.jpg'
            mask_name = str(idx) + '_input.jpg'

        img = io.imread(os.path.join(self.data_dir, str(idx), img_name))
        mask = io.imread(os.path.join(self.data_dir, str(idx), mask_name))



        ## determine whether it is gray mask or RGB mask
        # there is no channel(c) for gray mask
        # c = 3 for RGB
        # h: height
        # w: width
        if len(mask.shape) == 2:
            h, w  = mask.shape
        elif len(mask.shape) == 3:
            h, w, c = mask.shape
        ## Convert grey-scale label to one-hot encoding
        if self.is_colorization == False:
            new_mask = np.zeros((h, w, self.mask_num))
            for idx in range(self.mask_num):
                #if the mask has 3 dimension use this code
                new_mask[:, :, idx] = mask[:,:,0] == self.mask_value[idx]
        else:
            new_mask = mask
            #if the mask has 1 dimension use the code below
            #new_mask[:, :, idx] = mask == self.mask_value[idx]
        ## Transform image and mask
        if self.transform:
            img, new_mask = self.img_transform(img, new_mask)
        # ## Use dictionary to output
        # sample = {'img': img, 'mask': mask}
        # return sample
        # 这里的mask已经转化为one-hot vector
        return img, new_mask

    def img_transform(self,
                      img,
                      mask):
        """
```

```
        """
        ## Apply Transformations to Image and Mask
        img = self.transform(img)
        mask = self.transform(mask)
        return img, mask

    def img_transform_single(self, img):
      return self.transform(img)
```

In [ ]:
```
from torch.optim.lr_scheduler import StepLR
from torch.nn import MSELoss

# for colorization task, we expect the output channel(n_classes) = 3
model = UNET(input_channels_num= 1,n_classes = 3)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

## Initialize Dataloaders
train_dataset=ImageDataset(input_dir=colorization_data_dir, op="train", mask_json_path=mask_json, transforr
validation_dataset=ImageDataset(input_dir=colorization_data_dir, op="val", mask_json_path=mask_json, trans
train_dataloader = DataLoader(train_dataset, batch_size=train_batch_size, shuffle=True)
validation_dataloader = DataLoader(validation_dataset, batch_size=validation_batch_size, shuffle=False)
## Initialize Optimizer and Learning Rate Scheduler
optimizer = torch.optim.Adam(model.parameters(),lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)
criterion = nn.MSELoss()

patience = 5  # How many epochs to wait after last time validation loss improved.
best_val_loss = float('inf')
counter_early_stop = 0
early_stop = False
training_loss_values = []
validation_loss_values = []
print("Start Training...")
for epoch in range(num_epochs):
    ########################### Training #####################################
    print("\nEPOCH " +str(epoch+1)+" of "+str(num_epochs)+"\n")
    # TODO: Design your own training section
    model.train()
    train_loss = 0.0
    for images, mask in train_dataloader:  # No masks needed for colorization
      images = images.to(device)
```

```
            mask = mask.to(device)

            outputs = model(images)
            loss = criterion(outputs, mask)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        train_loss = train_loss / len(train_dataloader)
        training_loss_values.append(train_loss)

        ########################### Validation #####################################
        # TODO: Design your own validation section
        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            for images, masks in validation_dataloader:
                images = images.to(device)
                masks = masks.to(device)

                outputs = model(images)
                loss = criterion(outputs, masks)

                val_loss += loss.item()

        val_loss = val_loss / len(validation_dataloader)
        validation_loss_values.append(val_loss)
        print(f"Epoch {epoch+1}, Training loss: {train_loss:.4f}, Validation Loss: {val_loss:.4f}")

        scheduler.step()
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_model = copy.deepcopy(model.state_dict())  # Save the best model
            counter_early_stop = 0  # Reset counter
            print("Validation loss decreased, saving model...")
        else:
            counter_early_stop += 1
            print(f"Validation loss did not decrease, counter: {counter_early_stop}/{patience}")

        if counter_early_stop >= patience or epoch == num_epochs -1:
```

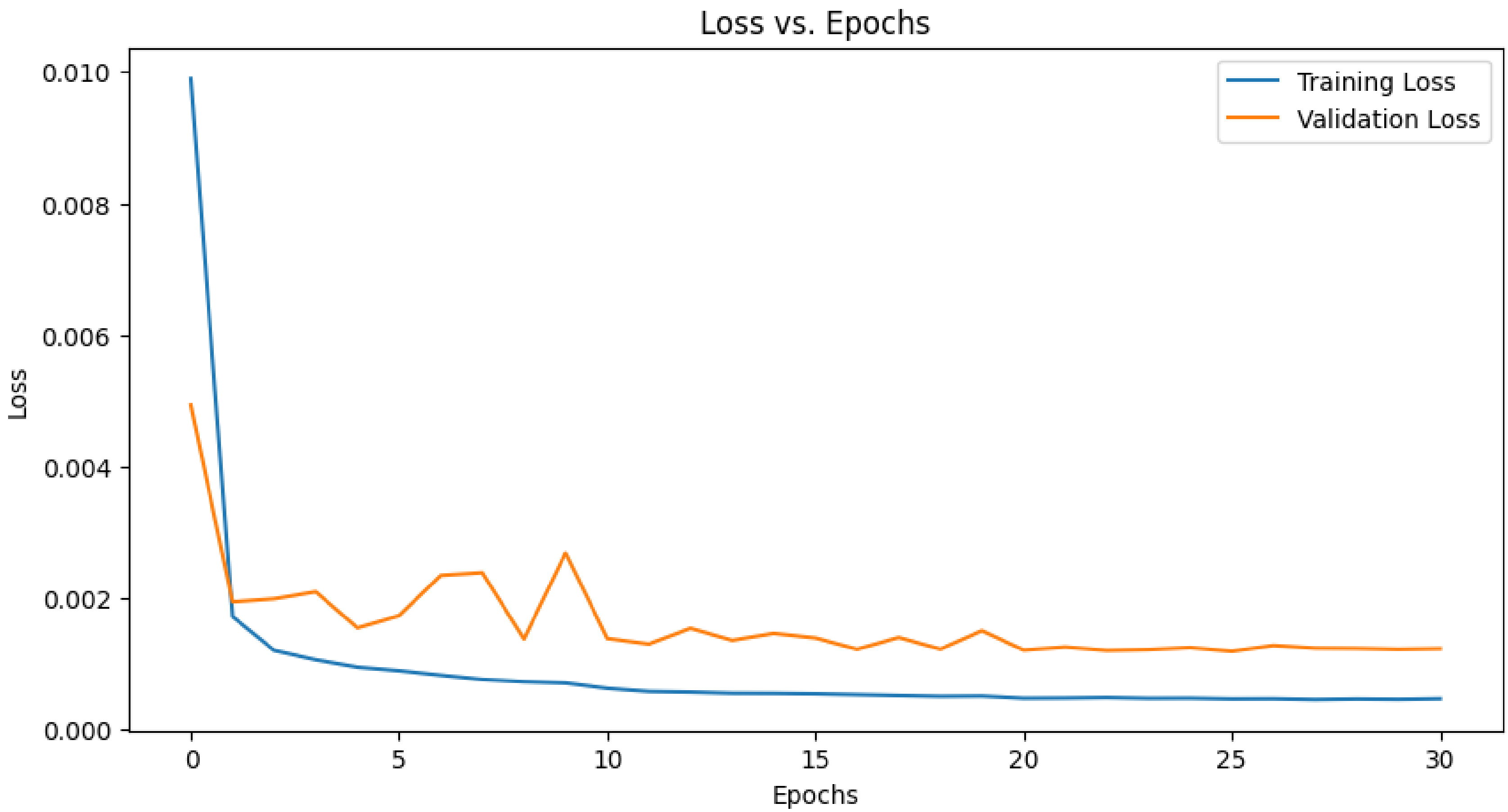
```
        early_stop = True
        break

torch.save(model.state_dict(), data_dir + '/model.pth')

plt.figure(figsize=(10, 5))
plt.plot(training_loss_values, label='Training Loss')
plt.plot(validation_loss_values, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs')
plt.legend()
plt.show()
```

```
Start Training...

EPOCH 1 of 200

Epoch 1, Training loss: 0.0099, Validation Loss: 0.0049
Validation loss decreased, saving model...

EPOCH 2 of 200

Epoch 2, Training loss: 0.0017, Validation Loss: 0.0019
Validation loss decreased, saving model...

EPOCH 3 of 200

Epoch 3, Training loss: 0.0012, Validation Loss: 0.0020
Validation loss did not decrease, counter: 1/5

EPOCH 4 of 200

Epoch 4, Training loss: 0.0011, Validation Loss: 0.0021
Validation loss did not decrease, counter: 2/5

EPOCH 5 of 200

Epoch 5, Training loss: 0.0009, Validation Loss: 0.0015
Validation loss decreased, saving model...

EPOCH 6 of 200

Epoch 6, Training loss: 0.0009, Validation Loss: 0.0017
Validation loss did not decrease, counter: 1/5

EPOCH 7 of 200

Epoch 7, Training loss: 0.0008, Validation Loss: 0.0023
Validation loss did not decrease, counter: 2/5

EPOCH 8 of 200

Epoch 8, Training loss: 0.0008, Validation Loss: 0.0024
Validation loss did not decrease, counter: 3/5
```

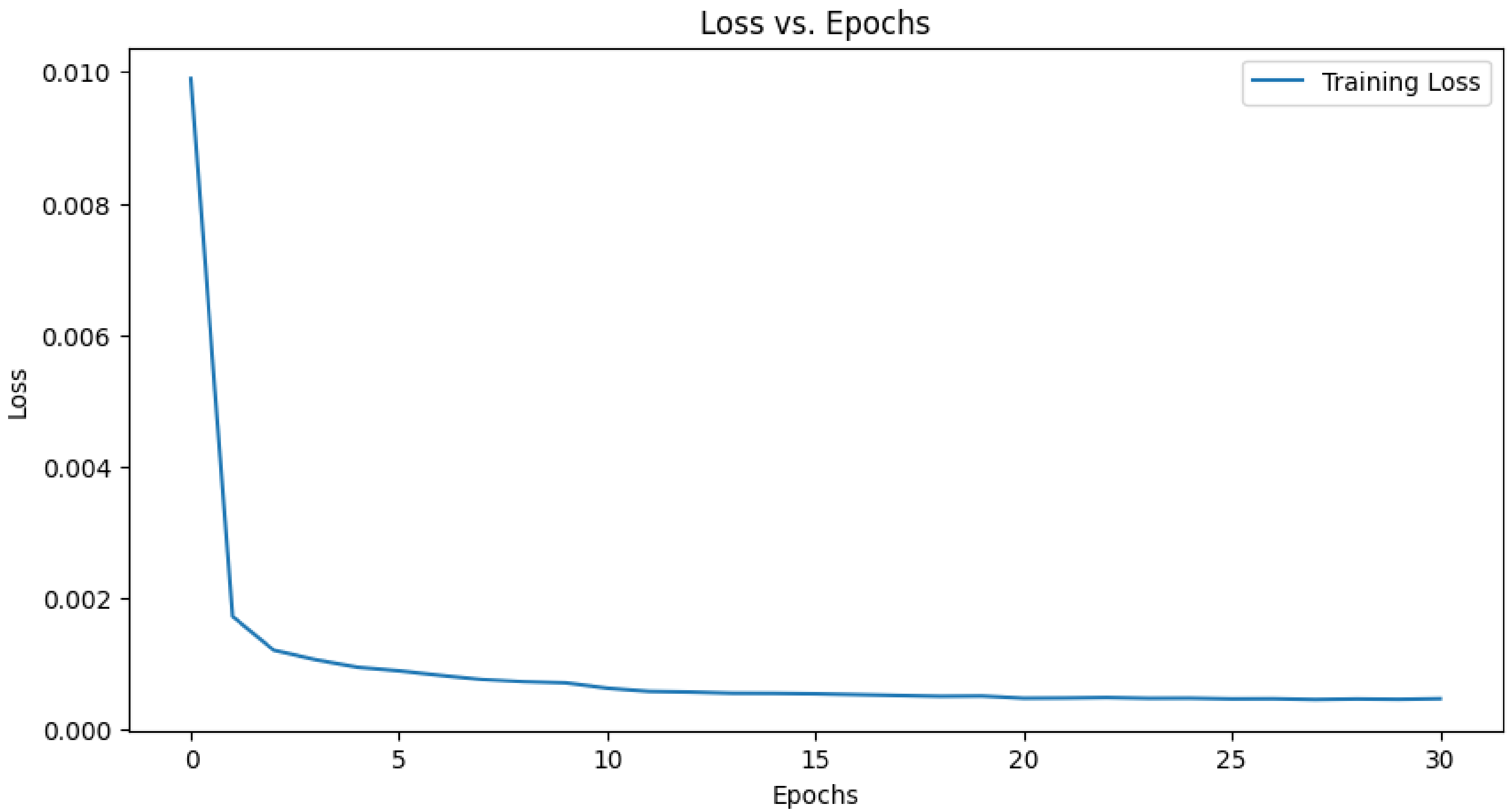
```
EPOCH 9 of 200

Epoch 9, Training loss: 0.0007, Validation Loss: 0.0014
Validation loss decreased, saving model...

EPOCH 10 of 200

Epoch 10, Training loss: 0.0007, Validation Loss: 0.0027
Validation loss did not decrease, counter: 1/5

EPOCH 11 of 200

Epoch 11, Training loss: 0.0006, Validation Loss: 0.0014
Validation loss did not decrease, counter: 2/5

EPOCH 12 of 200

Epoch 12, Training loss: 0.0006, Validation Loss: 0.0013
Validation loss decreased, saving model...

EPOCH 13 of 200

Epoch 13, Training loss: 0.0006, Validation Loss: 0.0015
Validation loss did not decrease, counter: 1/5

EPOCH 14 of 200

Epoch 14, Training loss: 0.0006, Validation Loss: 0.0014
Validation loss did not decrease, counter: 2/5

EPOCH 15 of 200

Epoch 15, Training loss: 0.0006, Validation Loss: 0.0015
Validation loss did not decrease, counter: 3/5

EPOCH 16 of 200

Epoch 16, Training loss: 0.0005, Validation Loss: 0.0014
Validation loss did not decrease, counter: 4/5

EPOCH 17 of 200
```

```
Epoch 17, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss decreased, saving model...

EPOCH 18 of 200

Epoch 18, Training loss: 0.0005, Validation Loss: 0.0014
Validation loss did not decrease, counter: 1/5

EPOCH 19 of 200

Epoch 19, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss did not decrease, counter: 2/5

EPOCH 20 of 200

Epoch 20, Training loss: 0.0005, Validation Loss: 0.0015
Validation loss did not decrease, counter: 3/5

EPOCH 21 of 200

Epoch 21, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss decreased, saving model...

EPOCH 22 of 200

Epoch 22, Training loss: 0.0005, Validation Loss: 0.0013
Validation loss did not decrease, counter: 1/5

EPOCH 23 of 200

Epoch 23, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss decreased, saving model...

EPOCH 24 of 200

Epoch 24, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss did not decrease, counter: 1/5

EPOCH 25 of 200

Epoch 25, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss did not decrease, counter: 2/5
```

```
EPOCH 26 of 200

Epoch 26, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss decreased, saving model...

EPOCH 27 of 200

Epoch 27, Training loss: 0.0005, Validation Loss: 0.0013
Validation loss did not decrease, counter: 1/5

EPOCH 28 of 200

Epoch 28, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss did not decrease, counter: 2/5

EPOCH 29 of 200

Epoch 29, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss did not decrease, counter: 3/5

EPOCH 30 of 200

Epoch 30, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss did not decrease, counter: 4/5

EPOCH 31 of 200

Epoch 31, Training loss: 0.0005, Validation Loss: 0.0012
Validation loss did not decrease, counter: 5/5
```

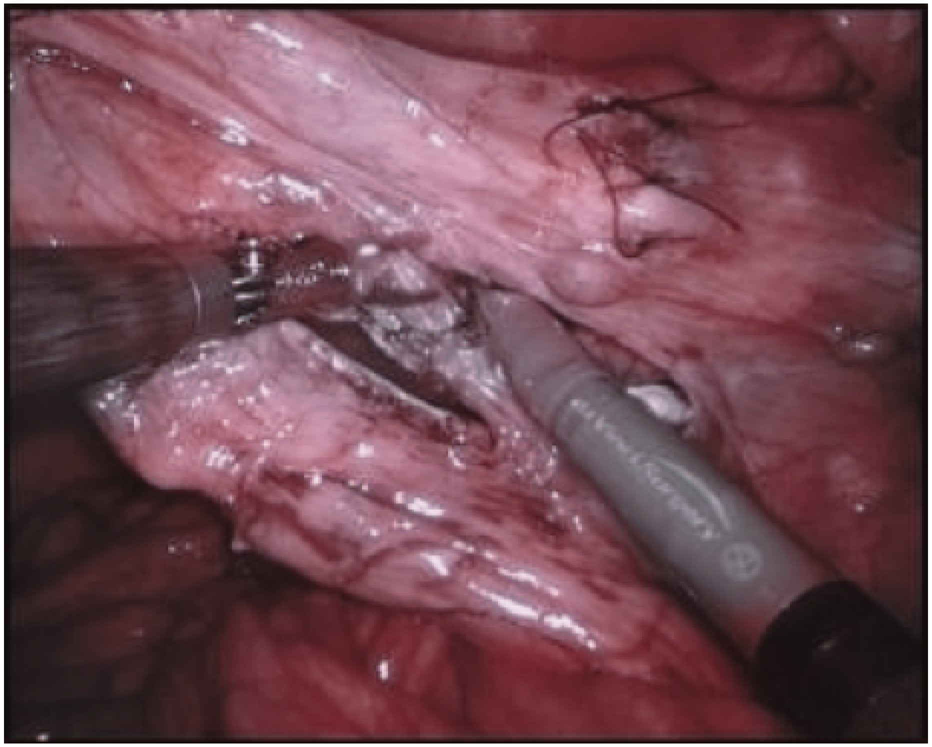


In [ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(training_loss_values, label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs')
plt.legend()
plt.show()
```



In [ ]:
```
train_batch_size = 10
validation_batch_size = 10

train_dataset = ImageDataset(input_dir=colorization_data_dir, op="train", mask_json_path=mask_json, transf
validation_dataset = ImageDataset(input_dir=colorization_data_dir, op="val", mask_json_path=mask_json, tra
train_dataloader = DataLoader(train_dataset, batch_size=train_batch_size, shuffle=True)
validation_dataloader = DataLoader(validation_dataset, batch_size=validation_batch_size, shuffle=False)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
outputs = model(next(iter(validation_dataloader))[0].to(device))

out_np = outputs[0].detach().cpu().numpy()
```

```
out_np = (out_np * 255).astype(np.uint8)

out_np = np.transpose(out_np, (1, 2, 0))

plt.imshow(out_np)
plt.axis('off')
plt.show()
```



In [ ]:
```
## Batch Size
train_batch_size = 10
validation_batch_size = 10

## Learning Rate
learning_rate = 0.001

# Epochs (Consider setting high and implementing early stopping)
num_epochs = 200
```

In [ ]:
```
pretrain_model_path = torch.load(data_dir + '/model.pth')
n_classes = len(json.load(open(mask_json)))
model = UNET(n_classes)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

model_dict = model.state_dict()

pretrain_dict = {k: v for k, v in pretrain_model_path.items() if k in model_dict and model_dict[k].size() =
model_dict.update(pretrain_dict)

model.load_state_dict(model_dict)




## Initialize Dataloaders
train_dataset=ImageDataset(input_dir=segmentation_data_dir, op="train", mask_json_path=mask_json, transform
validation_dataset=ImageDataset(input_dir=segmentation_data_dir, op="val", mask_json_path=mask_json, trans
test_dataset=ImageDataset(input_dir=segmentation_data_dir, op="test", mask_json_path=mask_json, transforms=
train_dataloader = DataLoader(train_dataset, batch_size=train_batch_size, shuffle=True)
validation_dataloader = DataLoader(validation_dataset, batch_size=validation_batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=1, shuffle=False)
## Initialize Optimizer and Learning Rate Scheduler
optimizer = torch.optim.Adam(model.parameters(),lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

# implement early stopping
patience = 10
best_val_loss = float('inf')
counter_early_stop = 0
early_stop = False
training_loss_values = []
validation_loss_values = []
print("Start Training...")
for epoch in range(num_epochs):
    ########################### Training ######################################
    print("\nEPOCH " +str(epoch+1)+" of "+str(num_epochs)+"\n")
    # TODO: Design your own training section
    model.train()
    train_loss = 0.0
    for images, masks in train_dataloader:
```

```
            images = images.to(device)
            masks = masks.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            # print(outputs.requires_grad)
            loss = DICELoss(n_classes)(outputs, masks)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        train_loss = train_loss / len(train_dataloader)
        training_loss_values.append(train_loss)

        ########################### Validation #####################################
        # TODO: Design your own validation section
        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            for images, masks in validation_dataloader:
                images = images.to(device)
                masks = masks.to(device)
                outputs = model(images)
                # probs = torch.softmax(outputs, dim = 1)
                loss = DICELoss(n_classes).forward(outputs, masks)
                val_loss += loss.item()
        val_loss = val_loss / len(validation_dataloader)
        validation_loss_values.append(val_loss)
        print(f"Epoch {epoch+1}, Training loss: {train_loss:.4f}, Validation Loss: {val_loss:.4f}")
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_model = copy.deepcopy(model.state_dict())  # Save the best model
            counter_early_stop = 0  # Reset counter
            print("Validation loss decreased, saving model...")
        else:
            counter_early_stop += 1
            print(f"Validation loss did not decrease, counter: {counter_early_stop}/{patience}")

        if counter_early_stop >= patience or epoch == num_epochs - 1:
            early_stop = True
            break
```

```
    scheduler.step()

plt.figure(figsize=(10, 5))
plt.plot(training_loss_values, label='Training Loss')
plt.plot(validation_loss_values, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs')
plt.legend()
plt.show()
```

```
Start Training...

EPOCH 1 of 200

Epoch 1, Training loss: 0.8570, Validation Loss: 0.7950
Validation loss decreased, saving model...

EPOCH 2 of 200

Epoch 2, Training loss: 0.7780, Validation Loss: 0.7806
Validation loss decreased, saving model...

EPOCH 3 of 200

Epoch 3, Training loss: 0.7267, Validation Loss: 0.7414
Validation loss decreased, saving model...

EPOCH 4 of 200

Epoch 4, Training loss: 0.6772, Validation Loss: 0.7437
Validation loss did not decrease, counter: 1/10

EPOCH 5 of 200

Epoch 5, Training loss: 0.6445, Validation Loss: 0.7653
Validation loss did not decrease, counter: 2/10

EPOCH 6 of 200

Epoch 6, Training loss: 0.6176, Validation Loss: 0.7096
Validation loss decreased, saving model...

EPOCH 7 of 200

Epoch 7, Training loss: 0.5733, Validation Loss: 0.6698
Validation loss decreased, saving model...

EPOCH 8 of 200

Epoch 8, Training loss: 0.5500, Validation Loss: 0.6807
Validation loss did not decrease, counter: 1/10
```

```
EPOCH 9 of 200

Epoch 9, Training loss: 0.5504, Validation Loss: 0.6984
Validation loss did not decrease, counter: 2/10

EPOCH 10 of 200

Epoch 10, Training loss: 0.5344, Validation Loss: 0.6591
Validation loss decreased, saving model...

EPOCH 11 of 200

Epoch 11, Training loss: 0.4741, Validation Loss: 0.6262
Validation loss decreased, saving model...

EPOCH 12 of 200

Epoch 12, Training loss: 0.4698, Validation Loss: 0.6219
Validation loss decreased, saving model...

EPOCH 13 of 200

Epoch 13, Training loss: 0.4476, Validation Loss: 0.6236
Validation loss did not decrease, counter: 1/10

EPOCH 14 of 200

Epoch 14, Training loss: 0.4423, Validation Loss: 0.6176
Validation loss decreased, saving model...

EPOCH 15 of 200

Epoch 15, Training loss: 0.4425, Validation Loss: 0.6187
Validation loss did not decrease, counter: 1/10

EPOCH 16 of 200

Epoch 16, Training loss: 0.4437, Validation Loss: 0.6163
Validation loss decreased, saving model...

EPOCH 17 of 200
```

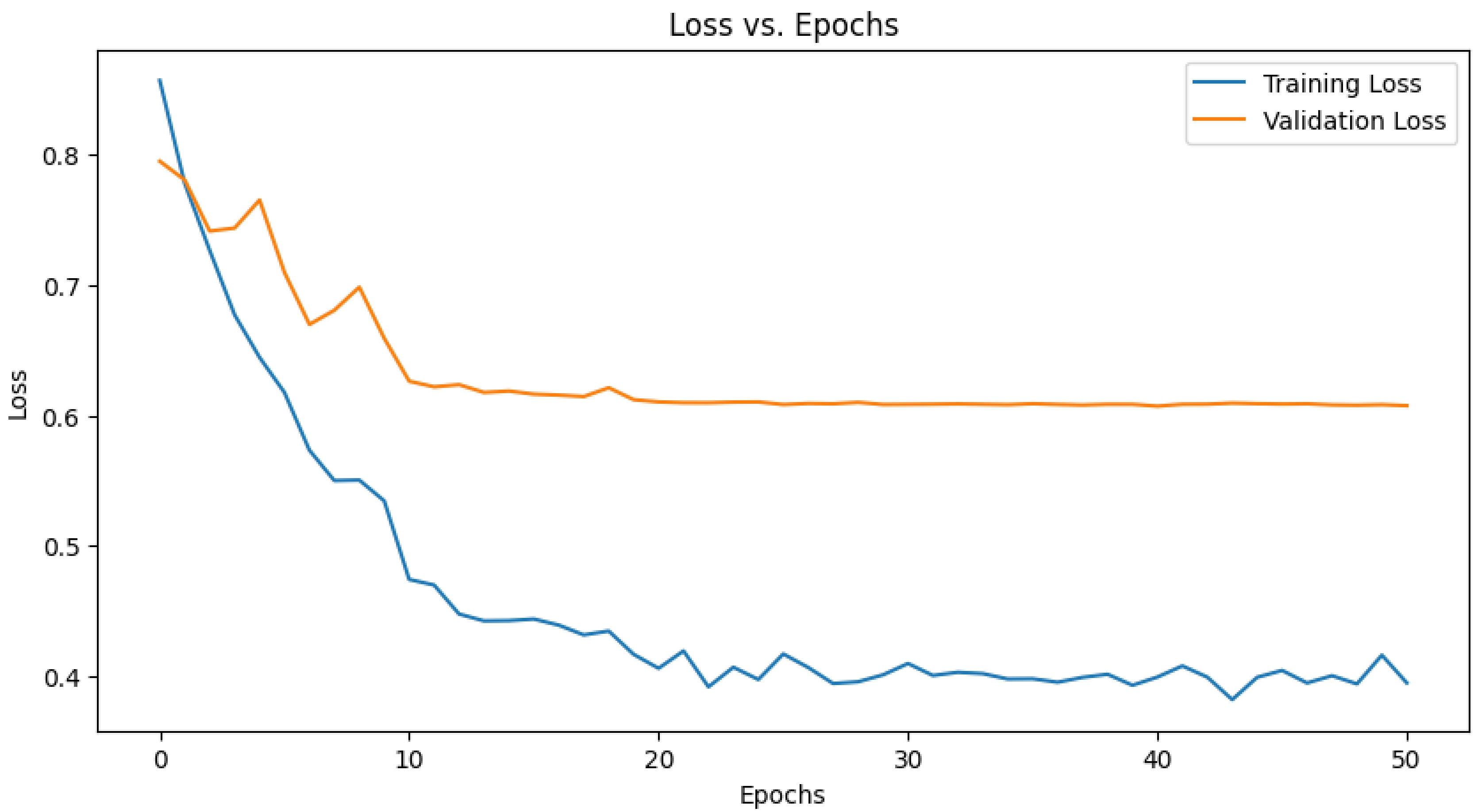
```
Epoch 17, Training loss: 0.4390, Validation Loss: 0.6155
Validation loss decreased, saving model...

EPOCH 18 of 200

Epoch 18, Training loss: 0.4315, Validation Loss: 0.6144
Validation loss decreased, saving model...

EPOCH 19 of 200

Epoch 19, Training loss: 0.4345, Validation Loss: 0.6212
Validation loss did not decrease, counter: 1/10

EPOCH 20 of 200

Epoch 20, Training loss: 0.4166, Validation Loss: 0.6120
Validation loss decreased, saving model...

EPOCH 21 of 200

Epoch 21, Training loss: 0.4059, Validation Loss: 0.6103
Validation loss decreased, saving model...

EPOCH 22 of 200

Epoch 22, Training loss: 0.4193, Validation Loss: 0.6097
Validation loss decreased, saving model...

EPOCH 23 of 200

Epoch 23, Training loss: 0.3917, Validation Loss: 0.6097
Validation loss decreased, saving model...

EPOCH 24 of 200

Epoch 24, Training loss: 0.4067, Validation Loss: 0.6102
Validation loss did not decrease, counter: 1/10

EPOCH 25 of 200

Epoch 25, Training loss: 0.3973, Validation Loss: 0.6103
Validation loss did not decrease, counter: 2/10
```

```
EPOCH 26 of 200

Epoch 26, Training loss: 0.4169, Validation Loss: 0.6083
Validation loss decreased, saving model...

EPOCH 27 of 200

Epoch 27, Training loss: 0.4066, Validation Loss: 0.6091
Validation loss did not decrease, counter: 1/10

EPOCH 28 of 200

Epoch 28, Training loss: 0.3942, Validation Loss: 0.6089
Validation loss did not decrease, counter: 2/10

EPOCH 29 of 200

Epoch 29, Training loss: 0.3956, Validation Loss: 0.6100
Validation loss did not decrease, counter: 3/10

EPOCH 30 of 200

Epoch 30, Training loss: 0.4009, Validation Loss: 0.6083
Validation loss did not decrease, counter: 4/10

EPOCH 31 of 200

Epoch 31, Training loss: 0.4096, Validation Loss: 0.6085
Validation loss did not decrease, counter: 5/10

EPOCH 32 of 200

Epoch 32, Training loss: 0.4004, Validation Loss: 0.6086
Validation loss did not decrease, counter: 6/10

EPOCH 33 of 200

Epoch 33, Training loss: 0.4029, Validation Loss: 0.6088
Validation loss did not decrease, counter: 7/10

EPOCH 34 of 200
```

```
Epoch 34, Training loss: 0.4018, Validation Loss: 0.6085
Validation loss did not decrease, counter: 8/10

EPOCH 35 of 200

Epoch 35, Training loss: 0.3977, Validation Loss: 0.6082
Validation loss decreased, saving model...

EPOCH 36 of 200

Epoch 36, Training loss: 0.3979, Validation Loss: 0.6090
Validation loss did not decrease, counter: 1/10

EPOCH 37 of 200

Epoch 37, Training loss: 0.3952, Validation Loss: 0.6084
Validation loss did not decrease, counter: 2/10

EPOCH 38 of 200

Epoch 38, Training loss: 0.3990, Validation Loss: 0.6079
Validation loss decreased, saving model...

EPOCH 39 of 200

Epoch 39, Training loss: 0.4014, Validation Loss: 0.6085
Validation loss did not decrease, counter: 1/10

EPOCH 40 of 200

Epoch 40, Training loss: 0.3929, Validation Loss: 0.6084
Validation loss did not decrease, counter: 2/10

EPOCH 41 of 200

Epoch 41, Training loss: 0.3992, Validation Loss: 0.6071
Validation loss decreased, saving model...

EPOCH 42 of 200

Epoch 42, Training loss: 0.4078, Validation Loss: 0.6085
```

```
Validation loss did not decrease, counter: 1/10

EPOCH 43 of 200

Epoch 43, Training loss: 0.3991, Validation Loss: 0.6086
Validation loss did not decrease, counter: 2/10

EPOCH 44 of 200

Epoch 44, Training loss: 0.3819, Validation Loss: 0.6094
Validation loss did not decrease, counter: 3/10

EPOCH 45 of 200

Epoch 45, Training loss: 0.3990, Validation Loss: 0.6090
Validation loss did not decrease, counter: 4/10

EPOCH 46 of 200

Epoch 46, Training loss: 0.4043, Validation Loss: 0.6087
Validation loss did not decrease, counter: 5/10

EPOCH 47 of 200

Epoch 47, Training loss: 0.3947, Validation Loss: 0.6089
Validation loss did not decrease, counter: 6/10

EPOCH 48 of 200

Epoch 48, Training loss: 0.4003, Validation Loss: 0.6080
Validation loss did not decrease, counter: 7/10

EPOCH 49 of 200

Epoch 49, Training loss: 0.3939, Validation Loss: 0.6078
Validation loss did not decrease, counter: 8/10

EPOCH 50 of 200

Epoch 50, Training loss: 0.4160, Validation Loss: 0.6082
Validation loss did not decrease, counter: 9/10
```

```
EPOCH 51 of 200

Epoch 51, Training loss: 0.3947, Validation Loss: 0.6074
Validation loss did not decrease, counter: 10/10
```



In [ ]:
```
test_dice_score = dice_score_dataset(model, test_dataloader, n_classes, use_gpu=device.type == 'cuda')
print(f"Epoch {epoch+1}, DICE score on the test set: {test_dice_score:.4f}")
```

```
Epoch 51, DICE score on the test set: 0.4896
```

Problem 2: Transfer Learning

Imports

In [ ]:
```
## Import VGG and FashionMNIST
import torch
from torchvision.models import vgg16
from torchvision.datasets import FashionMNIST
from torchvision import models, datasets, transforms
from torch.utils.data import DataLoader
from torch import nn
```

Data Loading

In [ ]:
```
## Specify Batch Size
train_batch_size = 64
test_batch_size = 64

## Specify Image Transforms
img_transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.Grayscale(num_output_channels= 3),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

## Download Datasets
train_data = FashionMNIST('./data', transform=img_transform, download=True, train=True)
test_data = FashionMNIST('./data', transform=img_transform, download=True, train=False)

## Initialize Dataloaders
training_dataloader = DataLoader(train_data, batch_size=train_batch_size, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=test_batch_size, shuffle=True)
```

Model Initialization and Training/Fine-tuning

Complete the rest of the assignment in the notebook below.

In [ ]:
```
def init_random_vgg16():
    model = models.vgg16()
```

```
    model.to('cuda')
    return model


def finetune_vgg16():
    model = models.vgg16(pretrained=True)

    for param in model.features.parameters():
        param.requires_grad = False

    num_features = model.classifier[6].in_features
    # Freeze all but the last layer: randomly initialize the last layer of your network and fine-tune this.
    model.classifier[6] = nn.Linear(num_features, len(train_data.classes))

    model.to('cuda')

    return model
```

In [ ]:

```
# TODO: Define your training loop, loss function, and optimizer and train your models
def train_model(model, train_dataloader, criterion, optimizer, num_epochs=6):
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct_predictions = 0

        for inputs, labels in train_dataloader:
            inputs, labels = inputs.to('cuda'), labels.to('cuda')
            optimizer.zero_grad()

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            _, preds = torch.max(outputs, 1)

            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            correct_predictions += torch.sum(preds == labels.data)

        epoch_loss = running_loss / len(train_dataloader.dataset)
        epoch_acc = correct_predictions.double() / len(train_dataloader.dataset)
```

```
        print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {epoch_loss:.4f}, Accuracy: {epoch_acc:.4f}')

    print('Training complete')
```

In [ ]:
```
def test_model(model, test_dataloader, criterion, optimizer, num_epochs=6):
        model.eval()
        running_loss = 0.0
        correct_predictions = 0

        for inputs, labels in test_dataloader:
            inputs, labels = inputs.to('cuda'), labels.to('cuda')
            with torch.no_grad():
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                _, preds = torch.max(outputs, 1)

            running_loss += loss.item() * inputs.size(0)
            correct_predictions += torch.sum(preds == labels.data)

        epoch_loss = running_loss / len(test_dataloader.dataset)
        epoch_acc = correct_predictions.double() / len(test_dataloader.dataset)

        print(f'Test Loss: {epoch_loss:.4f}, Test Acc: {epoch_acc:.4f}')
```

In [ ]:
```
random_vgg16 = init_random_vgg16()
finetuned_vgg16 = finetune_vgg16()
criterion = nn.CrossEntropyLoss()
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretr
ained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than
a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The curre
nt behavior is equivalent to passing `weights=VGG16_Weights.IMAGENET1K_V1`. You can also use `weights=VGG16
_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

In [ ]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Current device: {device}")
```

```
Current device: cuda
```

In [ ]:
```
optimizer_random = torch.optim.Adam(random_vgg16.parameters(), lr=0.001)
train_model(random_vgg16, training_dataloader, criterion, optimizer_random, num_epochs=5)
```

```
Epoch 1/5, Loss: 1.2297, Accuracy: 0.6239
Epoch 2/5, Loss: 0.3174, Accuracy: 0.8840
Epoch 3/5, Loss: 0.2643, Accuracy: 0.9042
Epoch 4/5, Loss: 0.2397, Accuracy: 0.9121
Epoch 5/5, Loss: 0.2184, Accuracy: 0.9204
Training complete
```

In [ ]:
```
test_model(random_vgg16, test_dataloader, criterion, optimizer_random, num_epochs=5)
```

```
Test Loss: 0.2340, Test Acc: 0.9156
```

In [ ]:
```
print(random_vgg16(next(iter(test_dataloader))[0].to(device)).shape)
print(next(iter(test_dataloader))[0].shape)
print(next(iter(training_dataloader))[0].shape)
```

```
torch.Size([64, 1000])
torch.Size([64, 3, 224, 224])
torch.Size([64, 3, 224, 224])
```

In [ ]:
```
optimizer_finetuned = torch.optim.Adam(finetuned_vgg16.classifier[6].parameters(), lr=0.001)
train_model(finetuned_vgg16, training_dataloader, criterion, optimizer_finetuned, num_epochs=5)
```

```
Epoch 1/5, Loss: 0.5724, Accuracy: 0.7938
Epoch 2/5, Loss: 0.5141, Accuracy: 0.8149
Epoch 3/5, Loss: 0.5110, Accuracy: 0.8173
Epoch 4/5, Loss: 0.5183, Accuracy: 0.8191
Epoch 5/5, Loss: 0.5127, Accuracy: 0.8191
Training complete
```

In [ ]:
```
test_model(finetuned_vgg16, test_dataloader, criterion, optimizer_finetuned, num_epochs=5)
```

```
Test Loss: 0.4118, Test Acc: 0.8498
```