REPORT
Polyps Segmentation and Objective Function
Prepared by: Muhammad Ridzuan

Task 1: Polyps Segmentation

Download Polyps detection and segmentation dataset from here. Images from 1 – 428 should be used for training and 429 – 612 for testing.

Code:

```
# DATASET CLASS
class PolypDataset(Dataset):
    def __init__(self, image_paths, label_paths, input_size):

        self.image_paths = image_paths
        self.label_paths = label_paths

        self.input_size = input_size

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        label = io.imread(self.label_paths[idx])
        label = self.preprocess_label()(label)

        img = io.imread(self.image_paths[idx])
        img = self.preprocess()(img)
        return (img, label, idx)

    def preprocess(self):
        return transforms.Compose([
            transforms.ToPILImage(),
            transforms.Resize(self.input_size, Image.BICUBIC),
            transforms.ToTensor()])

    def preprocess_label(self):
        return transforms.Compose([
            transforms.ToPILImage(),
            transforms.Resize(self.input_size, Image.BICUBIC),
            transforms.Grayscale(num_output_channels = 1),
            transforms.ToTensor()])

# SET PARAMETERS
input_size = (128, 128)
batch_size = 8
random_seed = 47

# Create train and testing sets
# Images 1 - 428 should be used for training and 429 - 612 for testing
x_train_path = ["../Original/{}.tif".format(i) for i in range(1, 428+1)]
x_test_path = ["../Original/{}.tif".format(i) for i in range(429, 612+1)]

y_train_path = ["../Ground Truth/{}.tif".format(i) for i in range(1, 428+1)]
y_test_path = ["../Ground Truth/{}.tif".format(i) for i in range(429, 612+1)]

# Initialise Datasets
train_set = PolypDataset(x_train_path, y_train_path, input_size)
val_set = PolypDataset(x_val_path, y_val_path, input_size)
test_set = PolypDataset(x_test_path, y_test_path, input_size
# Initialize Dataloaders
torch.manual_seed(random_seed)
train_loader = DataLoader(train_set, batch_size = batch_size, shuffle = True)
test_loader = DataLoader(test_set, batch_size = 1, shuffle = True)
print("Data Loaded.")
```

Task 1.1. Using this data, train a Convolutional Neural Network (CNN) to segment Polyps while fulfilling the following:

- **Use any architecture and data augmentation techniques.**
- **Use a maximum of 20 epoch to train.**
- **Use random initialization.**
- **Explore 3 different measures of performance (loss) during training while keeping the model parameters fixed.**
- **Report the solutions proposed.**

I chose to use UNet with ResNet50 as the backbone architecture. This combination has been shown to give promising results on a similar dataset in the 2020 Medico Automatic Polyp Segmentation Challenge [1].
The following hyperparameters were chosen: Adam optimizer with a learning rate of 0.0001, random initialization, and training for 20 epochs. The images were resized to 128 by 128 and were randomly flipped horizontally and vertically with a probability of 0. 5 in the training set.

I experimented with the following three objective functions: Dice, Jaccard, and focal losses. A smooth constant was added to ensure the losses continue to be differentiable during backpropagation.

Dice loss:

$$Dice\ loss = 1 - \frac{2|X \cap Y| + smooth}{|X| + |Y| + smooth} = 1 - \frac{2TP + smooth}{2TP + FP + FN + smooth}$$

Jaccard (a.k.a. intersection-over-union, or IoU) loss:

$$Jaccard\ loss = 1 - \frac{|X \cap Y| + smooth}{|X| + |Y| - |X \cap Y| + smooth} = 1 - \frac{TP + smooth}{TP + FP + FN + smooth}$$

where X and Y are the ground truth and predicted segmentation pixels.

Focal loss:

$$Focal\ loss = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$$

where $p_t$is a measurement of prediction accuracy, and α_t and are γ hyperparameters.

Code:

```python
# the code has been adapted from https://www.kaggle.com/balraj98/unet-resnet50-for-
cloth-parsing-pytorch
import segmentation_models_pytorch as smp

# model
ENCODER = 'resnet50'
ENCODER_WEIGHTS = None
CLASSES = ['polyp', 'background']
ACTIVATION = 'sigmoid'
EPOCHS = 20
LEARNING_RATE = 0.0001

# create segmentation model with pretrained encoder
model = smp.Unet(
    encoder_name=ENCODER,
    encoder_weights=ENCODER_WEIGHTS,
    classes=len(CLASSES),
    activation=ACTIVATION,
)

# define optimizer
optimizer = torch.optim.Adam([
    dict(params=model.parameters(), lr=LEARNING_RATE),
])

# define loss functions
dice_loss = smp.utils.losses.DiceLoss()
jaccard_loss = smp.utils.losses.JaccardLoss()
focal_loss = smp.losses.FocalLoss(mode='binary', alpha=0.2, gamma=2)

# TRAIN
train_epoch = smp.utils.train.TrainEpoch(
    model,
    loss=loss,
    metrics=metrics,
    optimizer=optimizer,
    device=DEVICE,
    verbose=True,
)

valid_epoch = smp.utils.train.ValidEpoch(
    model,
    loss=loss,
    metrics=metrics,
    device=DEVICE,
    verbose=True,
)

%time
if TRAINING:

    best_iou_score = 0.0
    train_logs_list, valid_logs_list = [], []

    for i in range(0, EPOCHS):

        # Perform training & validation
        print('\nEpoch: {}'.format(i))
        train_logs = train_epoch.run(train_loader)
        valid_logs = valid_epoch.run(valid_loader)
        train_logs_list.append(train_logs)
        valid_logs_list.append(valid_logs)

        # Save model if a better val IoU score is obtained
        if best_iou_score < valid_logs['iou_score']:
            best_iou_score = valid_logs['iou_score']
            torch.save(model, './best_model.pth')
```

<u>Task 1.2.</u> Apply the trained models on the testing set. Report Dice and Jaccard on the three experiments.

The Dice and Jaccard scores were calculated using the following:

Dice:

$$Dice = \frac{2|X \cap Y| + smooth}{|X| + |Y| + smooth} = \frac{2TP + smooth}{2TP + FP + FN + smooth}$$

Jaccard (a.k.a. IoU, or intersection-over-union):

$$Jaccard = \frac{|X \cap Y| + smooth}{|X| + |Y| - |X \cap Y| + smooth} = \frac{TP + smooth}{TP + FP + FN + smooth}$$

Code:

```
if os.path.exists('./best_model.pth'):
    best_model = torch.load('./best_model.pth', map_location=DEVICE)
    print('Model loaded successfully.')

def Dice(target, pred, smooth=1):
    smooth = 0 #1.
    num = pred.shape[0]
    pred = pred.reshape(num, -1) # Flatten
    target = target.reshape(num, -1) # Flatten
    intersection = (pred * target).sum()

    return (2. * intersection + smooth) / (pred.sum() + target.sum() + smooth)

def Jaccard(target, pred, smooth=1):

    #flatten label and prediction tensors
    pred = pred.view(-1)
    target = target.view(-1)

    intersection = (pred * target).sum()
    total = (pred + target).sum()
    union = total - intersection

    IoU = (intersection + smooth)/(union + smooth)

    return IoU
```

```python
jaccard_total = []
dice_total = []
for idx in range(len(test_dataset)):

    image, gt_mask = test_dataset[idx]
    image = torch.from_numpy(image).to(DEVICE).unsqueeze(0)

    # Predict test image
    pred_mask = best_model(image)
    pred_mask = pred_mask.detach().squeeze().cpu().numpy()
    # Convert pred_mask from `CHW` format to `HWC` format
    pred_mask = np.transpose(pred_mask,(1,2,0))

    # Convert gt_mask from `CHW` format to `HWC` format
    gt_mask = np.transpose(gt_mask,(1,2,0))

    # Calculate Dice score
    dice = Dice(gt_mask[...,0], pred_mask[...,0])
    dice_total.append(dice)

    # Calculate Jaccard score
    jaccard = Jaccard(torch.tensor(gt_mask)[...,0], torch.tensor(pred_mask)[...,0])
    jaccard_total.append(jaccard)


print("Dice average:", np.mean(dice_total))
print("Jaccard average:", np.mean(jaccard_total))
```

Results:

```
# Dice Loss
Dice average: 0.40874836722791563
Jaccard average: 0.3092003647849175

# Jaccard Loss
Dice average: 0.4200998640478806
Jaccard average: 0.31491492730425874

# Focal Loss, alpha 0.2
Dice average: 0.4476101216924698
Jaccard average: 0.3411279028538603
```

Task 1.3. Which metric gave the highest Dice score on the test set?

The focal loss gave the highest Dice score on the test set, followed by the Jaccard and Dice losses. Intuitively, by the principles of risk minimization [2], minimizing the Dice and Jaccard losses during training should correspond to maximizing the Dice and Jaccard scores during testing. Thus, I expected the Dice and Jaccard losses to give higher scores. However, the opposite was found to be true. This is likely because the model was not given enough time to train to optimize the loss functions. It is also possible that the learning rate of 0.0001 may be too low given the size of the architecture (UNet-ResNet50) and the small number of epochs (20).

Task 1.4. Train the network again, but this time with a combined metric (mean) from the three selected. Repeat the experiment while fixing all parameters. Report Dice and Jaccard on the testing set in a table and compare with the results from Task 1.2.

Brief discussion:
The focal loss is a distribution-based loss function, where all pixels (including the background and object of interest) are given consideration, with a focus given to the class with less samples (typically the object of interest) to address class imbalance. Conversely, the Jaccard and Dice losses are region-based loss functions, where the loss is calculated based on the intersection and unions of the true and predicted regions of the object of interest while the background is largely ignored. This makes Jaccard and Dice scale invariant and particularly useful for segmenting small objects.

Given these differences and the fact that the polyps have variable sizes, I would expect a combined loss function to be able to leverage these characteristic differences and give a better result on the final performance metric. However, I found the combined metric to give the lowest scores compared to each of the individual metrics (focal, Dice, and Jaccard). This is likely because the number of epochs is too low for the model to properly optimize the combined metric of three loss functions. From the plots of the losses and Dice and Jaccard scores, it is also evident that the model is still learning since the losses are still decreasing while the Dice and Jaccard scores are still increasing.

Code:

```
from torch.nn.modules.loss import _Loss
from segmentation_models_pytorch.losses import JaccardLoss, DiceLoss, FocalLoss

class CombinedLoss(_Loss):
    def_init_(self):#, y_pred, y_true):
        super(CombinedLoss, self).__init__()

        self.jaccard = JaccardLoss(mode="binary")
        self.dice = DiceLoss(mode="binary")
        self.focal = FocalLoss(mode="binary")

    def forward(self, y_pred, y_true):
        jaccard_loss = self.jaccard(y_pred, y_true)
        dice_loss = self.dice(y_pred, y_true)
        focal_loss = self.focal(y_pred, y_true)

        average_loss = (jaccard_loss + dice_loss + focal_loss)/3

        return average_loss
```

Result:

Loss	Dice Score (on test set)	Jaccard Score (on test set)
Dice	0.4087	0.3092
Jaccard	0.4201	0.3149
Focal	0.4476	0.3411
Combined	0.3944	0.2922

References

[1] Alam, S., Tomar, N. K., Thakur, A., Jha, D., & Rauniyar, A. (2020). Automatic Polyp Segmentation using U-Net-ResNet50. *arXiv preprint arXiv:2012.15247*.

[2] Vapnik, V. (1991), Principles of risk minimization for learning theory, in *NIPS'91: Proceedings of the 4th International Conference on Neural Information Processing Systems,* Morgan Kaufmann Publishers Inc., San Francisco, CA, pp. 831–838.