seg

October 27, 2023

1 CS6476 - Segmentation Assignment

2 Question 1 (All Parts)

[1]:
```
import os
import albumentations as A
from dataset import SegDataset
from torch.utils.data import DataLoader
from albumentations.pytorch import ToTensorV2
from PIL import Image
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import torch.nn as nn
import torchvision.transforms.functional as TF
from tqdm import tqdm
from utils import *
from train import train_epoch
import torch.optim as optim
import cv2 as cv
from matplotlib import pyplot as plt
import numpy as np
train_dir="Data/train" #Enter the path to your train data
test_dir="Data/test"  #Enter the path to your test data
save_path="Data/saves" #Enter path to save model ouputs
###START YOUR CODE
training_image_sample_count = 0
training_mask_sample_count = 0
testing_image_sample_count = 0
testing_mask_sample_count = 0
a = 1
training_image_blue_channel_data = None
training_image_green_channel_data = None
training_image_red_channel_data = None
training_mask_blue_channel_data = None
training_mask_green_channel_data = None
training_mask_red_channel_data = None
testing_image_blue_channel_data = None
```

```
testing_image_green_channel_data = None
testing_image_red_channel_data = None
testing_mask_blue_channel_data = None
testing_mask_green_channel_data = None
testing_mask_red_channel_data = None
training_image_dimensions = None
training_mask_dimensions = None
testing_image_dimensions = None
testing_mask_dimensions = None

def add_to_array(data, data_to_add):
    if np.any(data):
        data = np.concatenate((data, data_to_add), axis=0)
    else:
        data = data_to_add
    return data

for images in os.listdir(train_dir+'/image'):
    # check if the image ends with png or jpg or jpeg
    if (images.endswith(".png") or images.endswith(".jpg")\
        or images.endswith(".jpeg")):
        image = cv.imread(train_dir+'/image/'+images)
        if a < 9:
            plt.subplot(5, 2, a)
            plt.imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
            a += 2
        b = image[:,:,0] # get blue channel
        g = image[:,:,1] # get green channel
        r = image[:,:,2] # get red channel
        training_image_blue_channel_data =␣
 ↪add_to_array(training_image_blue_channel_data, b.flatten())
        training_image_green_channel_data =␣
 ↪add_to_array(training_image_green_channel_data, g.flatten())
        training_image_red_channel_data =␣
 ↪add_to_array(training_image_red_channel_data, r.flatten())
        training_image_dimensions = add_to_array(training_image_dimensions,␣
 ↪image.shape[0:2])
        training_image_sample_count += 1
a = 2
for images in os.listdir(train_dir+'/mask'):
    # check if the image ends with png or jpg or jpeg
    if (images.endswith(".png") or images.endswith(".jpg")\
        or images.endswith(".jpeg")):
        mask = cv.imread(train_dir+'/mask/'+images)
        if a < 10:
            plt.subplot(5, 2, a)
            plt.imshow(cv.cvtColor(mask, cv.COLOR_BGR2RGB))
```

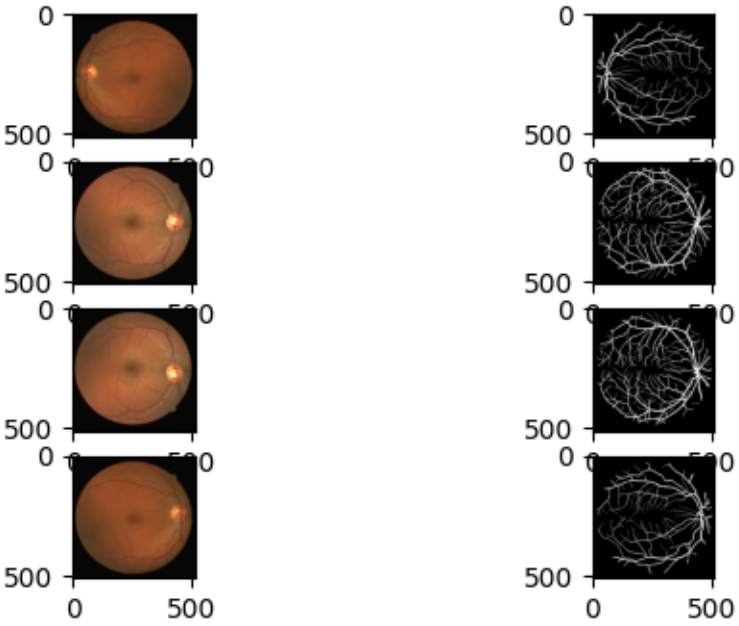
```
            a += 2
        b = mask[:,:,0] # get blue channel
        g = mask[:,:,1] # get green channel
        r = mask[:,:,2] # get red channel
        training_mask_blue_channel_data =␣
 ↪add_to_array(training_mask_blue_channel_data, b.flatten())
        training_mask_green_channel_data =␣
 ↪add_to_array(training_mask_green_channel_data, g.flatten())
        training_mask_red_channel_data =␣
 ↪add_to_array(training_mask_red_channel_data, r.flatten())
        training_mask_dimensions = add_to_array(training_mask_dimensions, mask.
 ↪shape[0:2])
        training_mask_sample_count += 1
for images in os.listdir(test_dir+'/image'):
    # check if the image ends with png or jpg or jpeg
    if (images.endswith(".png") or images.endswith(".jpg")\
        or images.endswith(".jpeg")):
        image = cv.imread(test_dir+'/image/'+images)
        b = image[:,:,0] # get blue channel
        g = image[:,:,1] # get green channel
        r = image[:,:,2] # get red channel
        testing_image_blue_channel_data =␣
 ↪add_to_array(testing_image_blue_channel_data, b.flatten())
        testing_image_green_channel_data =␣
 ↪add_to_array(testing_image_green_channel_data, g.flatten())
        testing_image_red_channel_data =␣
 ↪add_to_array(testing_image_red_channel_data, r.flatten())
        testing_image_dimensions = add_to_array(testing_image_dimensions, image.
 ↪shape[0:2])
        testing_image_sample_count += 1
for images in os.listdir(test_dir+'/mask'):
    # check if the image ends with png or jpg or jpeg
    if (images.endswith(".png") or images.endswith(".jpg")\
        or images.endswith(".jpeg")):
        mask = cv.imread(test_dir+'/mask/'+images)
        b = mask[:,:,0] # get blue channel
        g = mask[:,:,1] # get green channel
        r = mask[:,:,2] # get red channel
        testing_mask_blue_channel_data =␣
 ↪add_to_array(testing_mask_blue_channel_data, b.flatten())
        testing_mask_green_channel_data =␣
 ↪add_to_array(testing_mask_green_channel_data, g.flatten())
        testing_mask_red_channel_data =␣
 ↪add_to_array(testing_mask_red_channel_data, r.flatten())
        testing_mask_dimensions = add_to_array(testing_mask_dimensions, mask.
 ↪shape[0:2])
```

```
        testing_mask_sample_count += 1
print("Training Image Sample Size:",training_image_sample_count)
print("Training Mask Sample Count:",training_mask_sample_count)
print("Testing Image Sample Count:",testing_image_sample_count)
print("Testing Mask Sample Count:",testing_mask_sample_count)
print("Training Image Pixel Mean Blue Channel:", np.
 ↪mean(training_image_blue_channel_data))
print("Training Image Pixel Mean Green Channel:", np.
 ↪mean(training_image_green_channel_data))
print("Training Image Pixel Mean Red Channel:", np.
 ↪mean(training_image_red_channel_data))
print("Training Mask Pixel Mean Blue Channel:", np.
 ↪mean(training_mask_blue_channel_data))
print("Training Mask Pixel Mean Green Channel:", np.
 ↪mean(training_mask_green_channel_data))
print("Training Mask Pixel Mean Red Channel:", np.
 ↪mean(training_mask_red_channel_data))
print("Testing Image Pixel Mean Blue Channel:", np.
 ↪mean(testing_image_blue_channel_data))
print("Testing Image Pixel Mean Green Channel:", np.
 ↪mean(testing_image_green_channel_data))
print("Testing Image Pixel Mean Red Channel:", np.
 ↪mean(testing_image_red_channel_data))
print("Testing Mask Pixel Mean Blue Channel:", np.
 ↪mean(testing_mask_blue_channel_data))
print("Testing Mask Pixel Mean Green Channel:", np.
 ↪mean(testing_mask_green_channel_data))
print("Testing Mask Pixel Mean Red Channel:", np.
 ↪mean(testing_mask_red_channel_data))
print("Training Image Minimum Dimension:", np.min(training_image_dimensions))
print("Training Image Maximum Dimension:", np.max(training_image_dimensions))
print("Training Mask Minimum Dimension:", np.min(training_mask_dimensions))
print("Training Mask Maximum Dimension:", np.max(training_mask_dimensions))
print("Testing Image Minimum Dimension:", np.min(testing_image_dimensions))
print("Testing Image Maximum Dimension:", np.max(testing_image_dimensions))
print("Testing Mask Minimum Dimension:", np.min(testing_mask_dimensions))
print("Testing Mask Maximum Dimension:", np.max(testing_mask_dimensions))
###END YOUR CODE
```

```
/Users/auraaudio/Downloads/Assignment4/.venv/lib/python3.9/site-
packages/urllib3/__init__.py:34: NotOpenSSLWarning: urllib3 v2.0 only supports
OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'.
See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(

Training Image Sample Size: 80
Training Mask Sample Count: 80
```

```
Testing Image Sample Count: 20
Testing Mask Sample Count: 20
Training Image Pixel Mean Blue Channel: 41.91270318031311
Training Image Pixel Mean Green Channel: 69.92539114952088
Training Image Pixel Mean Red Channel: 128.49599194526672
Training Mask Pixel Mean Blue Channel: 22.343516063690185
Training Mask Pixel Mean Green Channel: 22.343516063690185
Training Mask Pixel Mean Red Channel: 22.343516063690185
Testing Image Pixel Mean Blue Channel: 40.85104465484619
Testing Image Pixel Mean Green Channel: 67.64550380706787
Testing Image Pixel Mean Red Channel: 132.04942092895507
Testing Mask Pixel Mean Blue Channel: 22.329174613952638
Testing Mask Pixel Mean Green Channel: 22.329174613952638
Testing Mask Pixel Mean Red Channel: 22.329174613952638
Training Image Minimum Dimension: 512
Training Image Maximum Dimension: 512
Training Mask Minimum Dimension: 512
Training Mask Maximum Dimension: 512
Testing Image Minimum Dimension: 512
Testing Image Maximum Dimension: 512
Testing Mask Minimum Dimension: 512
Testing Mask Maximum Dimension: 512
```



3 Question 2

a) Choice of Augmentations: Please specify the two augmentation techniques you selected for training data and explain why. Describe how these augmentations can benefit the training process.

Answer: I noticed most images tended to have the macula of the eye either on the left or right side. For this reason, for my second augmentation I chose `Rotation` to help create images with macula in various orientations. To further randomize the location of macula on either the left or right side of the image, I used `HorizontalFlip`.

b) Explain the purpose of normalizing pixel values in training and test data. Discuss why normalization is an important preprocessing step in deep learning.

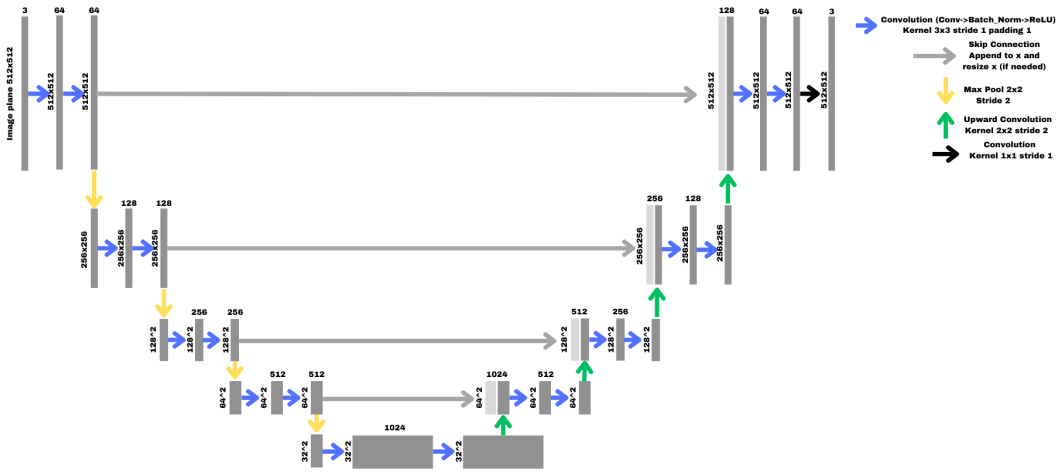
Answer: Normalizing the pixel values means reducing the overall brightness of each image to fall within a certain range. For a dataset and deep learning in general, normalization is an important preprocessing step to ensuring each sample of a dataset has an equal impact on what inferences can be made. For example, if one image had an extremely high brightness or rapid changes in brightness and color, this might influence the overall statistical characteristics of the entire dataset and thus the inferences a machine learning model might make.

c) Share your insights on why we did not add any augmentations (other than resizing and normalization) to the test data transformations. Consider the implications of augmenting test data and how it may differ from the training data scenario.

Answer: The reason test data typically does not have many augmentations is because while having more data during training would help the network improve accuracy, having augmented data during testing has little effect on training which has already occurred. Messing with the testing data may actually hinder evaluation of the network because it's best to test data as observed or captured in a real world scenario. This is what is done in scientific experiments as well. The test data is similar to the real data you would collect during the scientific experiment from participants. Messing with this data would be a form of post-hoc analysis and could seriously mess with the results you gather from said experiment.

4 Question 3

a) Create a visual block diagram representing the U-Net model implemented in the coding assignment. Ensure the diagram clearly illustrates how the feature map size changes throughout the model's architecture.



b) What is the primary role of skip connections within the U-Net architecture, and do they constitute a necessary component of the model's design?

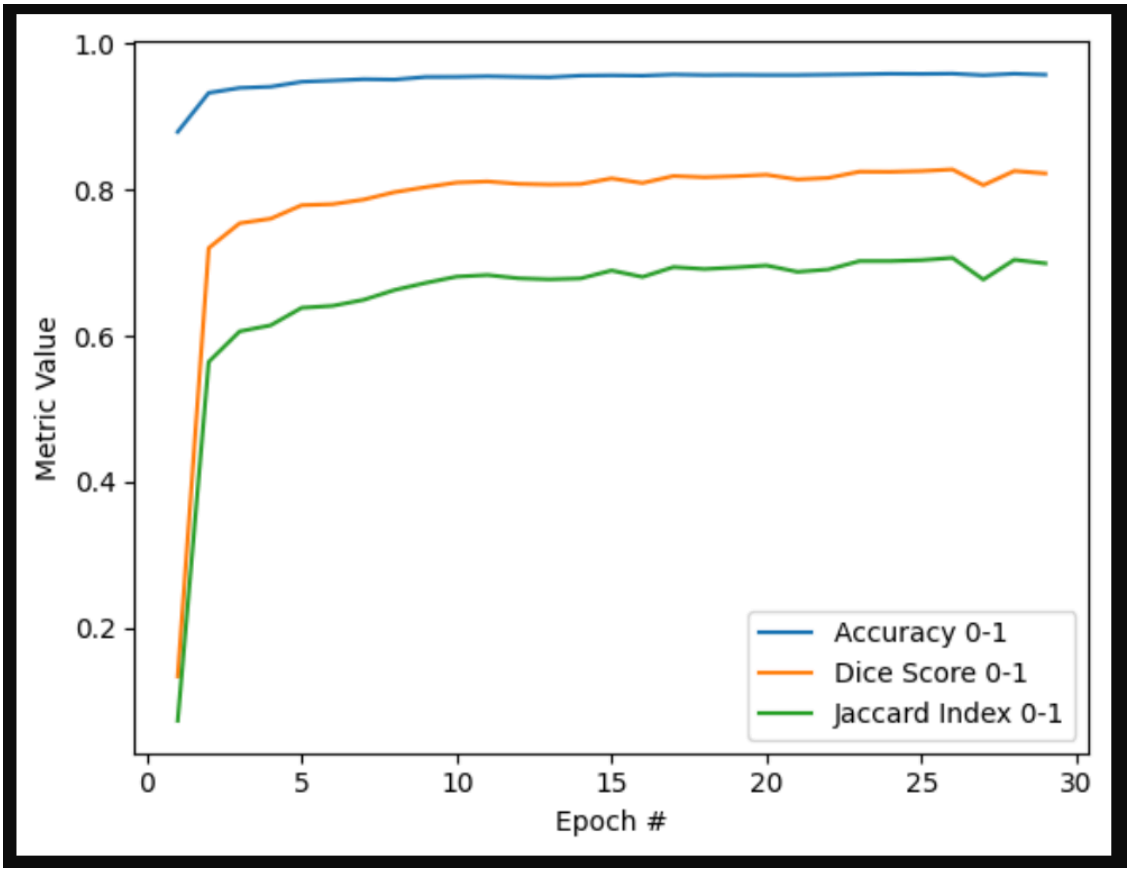
Answer: The primary role of the skip connections within the U-Net architecture is to reduce the "vanishing gradient problem." They do constitute a necessary component of the model's design because they were inspired from RESNET. Essentially, the skip connections in U-Net function as the residuals which remind the network of what it was learning in the first place after the features have been reduced.

c) U-Net is often used for medical image segmentation. What are the advantages of using this architecture in medical imaging tasks, and what considerations should be made when adapting it to such applications?

Answer: The advantage of using U-Net for medical image segmentation is for feature extraction. If one wants to identify a specific person's retina or a specific type of broken bone in an x-ray, then image segmentation can be useful because it helps extract the features we are interested in. The statistics and properties of the dataset should be considered when using it for any application to help improve the performance for a specific application.

5 Question 4

a) Plot the Graphs for Pixel Accuracy, Dice Score, and Jaccard Index accumulated during Training. Describe how these metrics change over epochs and what they mean in terms of the model output.



Answer: Over each epoch, the metrics improve and show the model output is getting closer to matching the mask.

b) Explaining the Differences Between These Metrics: Explain the distinct perspectives each metric offers in evaluating the segmentation model's performance.

Answer: The pixel accuracy shows how many pixels are correctly predicted in the image out of any pixel which is on or off. The dice score shows how similar the predicted pixels are to the mask (or how much they intersect) by comparing the intersection of the prediction and mask to the total number of pixels which are on. The Jaccard index computes the intersection over union or how many pixels intersect those which are on. The dice score is great for comparing how well the predictions match the edges of the mask and the jaccard index is great for predicting how much the predictions overlap with the mask.

c) Identifying the Least Reliable Measure and Why: Out of the three metrics, which one is the least reliable for assessing segmentation performance, and what are the specific reasons or limitations that make it less reliable?

Answer: The least reliable measurement for assesing segmentation performance would be pixel accuracy because it focuses on the image globally. Dice score and jaccard index focus on the areas where the segmentation is happening by incorporating the unions and intersections of the images and masks. Pixel accuracy is only a percentage of where each pixel should be vs. a ground truth pixel. For example, if a mask was guassian noise and some prediction somehow matched this guassian noise–although the pixel accuracy would be 100%, this guassian noise doesn't constitute

a shape or a segment. This is where a union or intersection of pixels come into play.

6 Question 5

a) Which loss function did you use to train your model? Why did you think this was the most apt in this case?

Answer: I used the BCEWithLogitsLoss() function because it includes Binary Cross Entropy loss along with the logits which are like sigmoids to map an input in between 0-1. This was the most apt in this case because the original U-Net paper uses a softmax loss combined with a cross entropy loss. In this case, since the class is binary we can get the same functionality out of the Binary Cross Entropy loss.

b) If instead of binary segmentation, the problem was multiclass segmentation, what changes would be required to be made in the code?

Answer: The changes which would be required for the code here would be for the convolution to output more channels instead of 1 and to use a generalized cross entropy loss or CrossEntropyLoss() which could provide functionality for more than binary classifcation.