Applying_SMP_Task3

October 4, 2023

H I

This script is provided for **task 3**.

This script is privately provided by Mohamad NAJAFI for the DiaDeep PhD applicant task. The task consists of three parts, and all the necessary scripts have been provided to achieve the requested results. The description of task is provided as Segmentation: WSI and U-Net Pipeline Manipulation.

It should be mentioned that the main chalenges and important points are as follows for me:

- The 'annotation.csv' file cannot be read as a dataframe because all the data is stored in the first column of the CSV file. Additionally, there is an issue with reading the LINESTRING format of the geometry data. This requires parsing the data to obtain WKT or OPENCV geometry data, which needs to be addressed.
- I have expriences in tensorflow and leras for DML and DL application, however, I tried to not use tf libraries and just work with **torch** in this project.

I am available and open to optimizing this script for various WSI image segmentation purposes. Having gained a better understanding of its functionality, I believe it would be applicable to focus on a specific class of annotation terms or a selected tissue for further refinement. Additionally, to achieve superior results, I've prepared a pilot format for fine-tuning SMP models using predefined weights. This can be applied on an augmented dataset generated by me from tiles of the original image, especially when ample image data is available and can be processed on a robust server.

I am available to offer explanations or make modifications to this script based on your specific inquiry.

Mohamad Najafi

[1]:
```
# # These libraries sould be install specifically for this task purpose
# !pip install segmentation-models-pytorch
# !pip install pytorch-lightning==1.5.4
# !pip install torchtext==0.6.0
```

[2]:
```
from google.colab import drive
from zipfile import ZipFile
from glob import glob
import os
```

```
import torch
import matplotlib.pyplot as plt
# import pytorch_lightning as pl
import segmentation_models_pytorch as smp
import numpy as np
from pprint import pprint
from torch.utils.data import DataLoader
from PIL import Image
from torch.utils.data import Dataset
from torchvision import transforms
from torch.utils.data import DataLoader
import torch.optim as optim
import torch.nn as nn
```

[3]:
```
drive.mount('/content/drive')
```

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

[4]:
```
%cd /content/drive/MyDrive/TestDiaDeep
```

```
/content/drive/MyDrive/TestDiaDeep
```

[5]:
```
class CustomDataset(Dataset):
    def __init__(self, image_folder, mask_folder, transform=None):
        self.image_folder = image_folder
        self.mask_folder = mask_folder
        self.transform = transform

        self.images = sorted(os.listdir(image_folder))
        self.masks = sorted(os.listdir(mask_folder))

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

    def __getitem__(self, idx):
        img_path = os.path.join(self.image_folder, self.images[idx])
        mask_path = os.path.join(self.mask_folder, self.masks[idx])

        image = Image.open(img_path)
        mask = Image.open(mask_path)

        if self.transform:
            image = self.transform(image)
            mask = self.transform(mask)

        return {'image': image, 'mask': mask}
```

[6]:
```python
data_transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
])
```

Loading data to be prepared for segmentation algorithm using pytorch framework

[7]:
```python
dataset_path = "/content/drive/MyDrive/TestDiaDeep/new_data_Morph_256_1"
train_path = os.path.join(dataset_path, "train")
valid_path = os.path.join(dataset_path, "test")


from torch.utils.data import DataLoader

train_dataset = CustomDataset(train_path+'/image', train_path+'/mask',␣
 ↪transform=data_transform)
val_dataset = CustomDataset(valid_path+'/image', valid_path+'/mask',␣
 ↪transform=data_transform)

train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=8, shuffle=False)
```

Train a model without pretrained weights of imagenet

[8]:
```python
model_res = smp.Unet('resnet34', in_channels=3, classes=1)
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model_res.parameters(), lr=0.001)
```

[12]:
```python
num_epochs = 10
AV_val_losses_model = []
```

[61]:
```python
# for epoch in range(num_epochs):
#     model_res.train()
#     for i, batch in enumerate(train_loader):
#         images, masks = batch['image'], batch['mask']
#         outputs = model_res(images)
#         loss = criterion(outputs, masks)
#         optimizer.zero_grad()
#         loss.backward()
#         optimizer.step()

#         # #Print progress
#         # print(f'Training Batch [{i+1}/{len(train_loader)}], Loss: {loss.
 ↪item():.4f}')

#     model_res.eval()
#     val_losses = []
```

```
#     with torch.no_grad():
#         for i, batch in enumerate(val_loader):
#             images, masks = batch['image'], batch['mask']
#             outputs = model_res(images)
#             val_loss = criterion(outputs, masks)
#             val_losses.append(val_loss.item())

#             # # Print progress
#             # print(f'Validation Batch [{i+1}/{len(val_loader)}], Val Loss:␣
 ↪{val_loss.item():.4f}')

#     avg_val_loss = sum(val_losses) / len(val_losses)
#     AV_val_losses_model.append(avg_val_loss)
#     print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}, Val Loss:
 ↪ {avg_val_loss:.4f}')
```

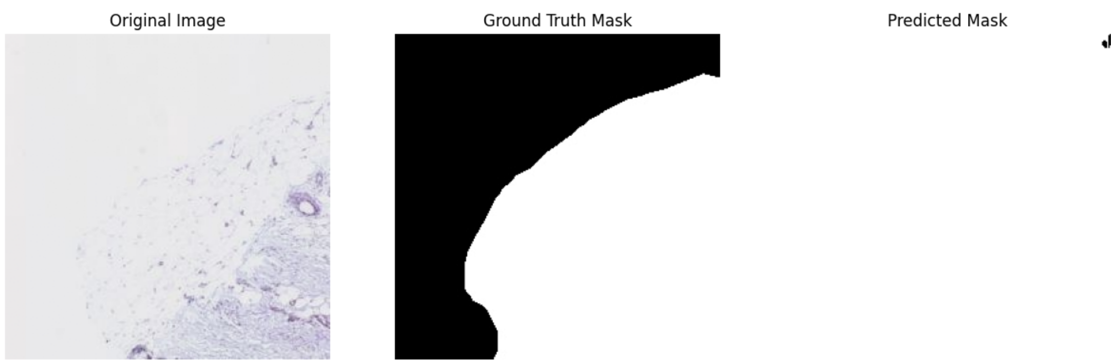
[14]:
```
torch.save(model_res.state_dict(), 'segmentation_model_res.pth')
```

[62]:
```
# # Plot validation loss
# plt.figure(figsize=(10, 5))
# plt.plot(range(1, num_epochs+1), AV_val_losses_model, label='Validation Loss')
# plt.xlabel('Epochs')
# plt.ylabel('Loss')
# plt.title('Validation Loss')
# plt.legend()
# plt.show()
```

In the folowing the result of model_res without pretrained weight is shown and as validation loss investigation it cannot be converged well since there is no sufficient data applied for training to reach the suited weights.

[22]:
```
import random
# Set model to evaluation mode
model_res.eval()

# Get a random sample from the validation set
sample = val_dataset[random.randint(0, len(val_dataset))]

# Extract image and mask
image = sample['image'].unsqueeze(0)  # Add batch dimension
mask = sample['mask'].unsqueeze(0)    # Add batch dimension

# Make prediction
with torch.no_grad():
    prediction = model_res(image)

# Convert prediction to binary mask
```

```python
predicted_mask = torch.sigmoid(prediction) > 0.5

# Plot the images
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(image.squeeze(0).permute(1, 2, 0))
axes[0].set_title('Original Image')

axes[1].imshow(mask.squeeze(0).squeeze(0), cmap='gray')
axes[1].set_title('Ground Truth Mask')

axes[2].imshow(predicted_mask.squeeze(0).squeeze(0), cmap='gray')
axes[2].set_title('Predicted Mask')

for ax in axes:
    ax.axis('off')

plt.show()
```



[52]:
```python
# # Load a UNet with pre-trained ImageNet weights
model_pre = smp.Unet('resnet34', in_channels=3, classes=1,␣
↪encoder_weights='imagenet')
```

[55]:
```python
num_epochs_pre = 15
```

[56]:
```python
# Defining parameters and optimization specification
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model_pre.parameters(), lr=0.001)
val_losses = []
AV_val_losses = []
```

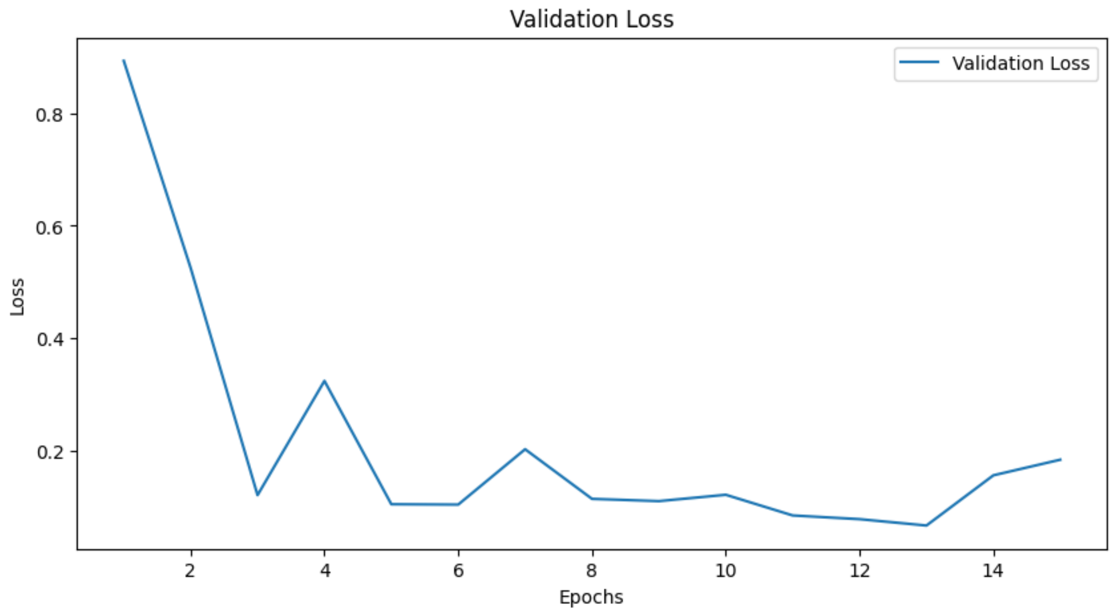
[63]:
```python
# # Train pretrained model with finetunning for our specefic image data
# for epoch in range(num_epochs_pre):
#     model_pre.train()
#     for i, batch in enumerate(train_loader):
```

```
#         images, masks = batch['image'], batch['mask']
#         outputs = model_pre(images)
#         loss = criterion(outputs, masks)
#         optimizer.zero_grad()
#         loss.backward()
#         optimizer.step()

#         # Print progress
#         # print(f'Training Batch [{i+1}/{len(train_loader)}], Loss: {loss.
↪item():.4f}')

#     model_pre.eval()
#     val_losses = []
#     with torch.no_grad():
#         for i, batch in enumerate(val_loader):
#             images, masks = batch['image'], batch['mask']
#             outputs = model_pre(images)
#             val_loss = criterion(outputs, masks)
#             val_losses.append(val_loss.item())

#             # Print progress
#             # print(f'Validation Batch [{i+1}/{len(val_loader)}], Val Loss:␣
↪{val_loss.item():.4f}')

#     avg_val_loss = sum(val_losses) / len(val_losses)
#     AV_val_losses.append(avg_val_loss)
#     print(f'Epoch [{epoch+1}/{num_epochs_pre}], Loss: {loss.item():.4f}, Val␣
↪Loss: {avg_val_loss:.4f}')
```

[59]:
```
torch.save(model_pre.state_dict(), 'segmentation_Model_pre.pth')
```

Validation plot for 15 epochs. It should be mentioned that if more images and computational sources are available the result would be better than this as the model is in the convergence path.

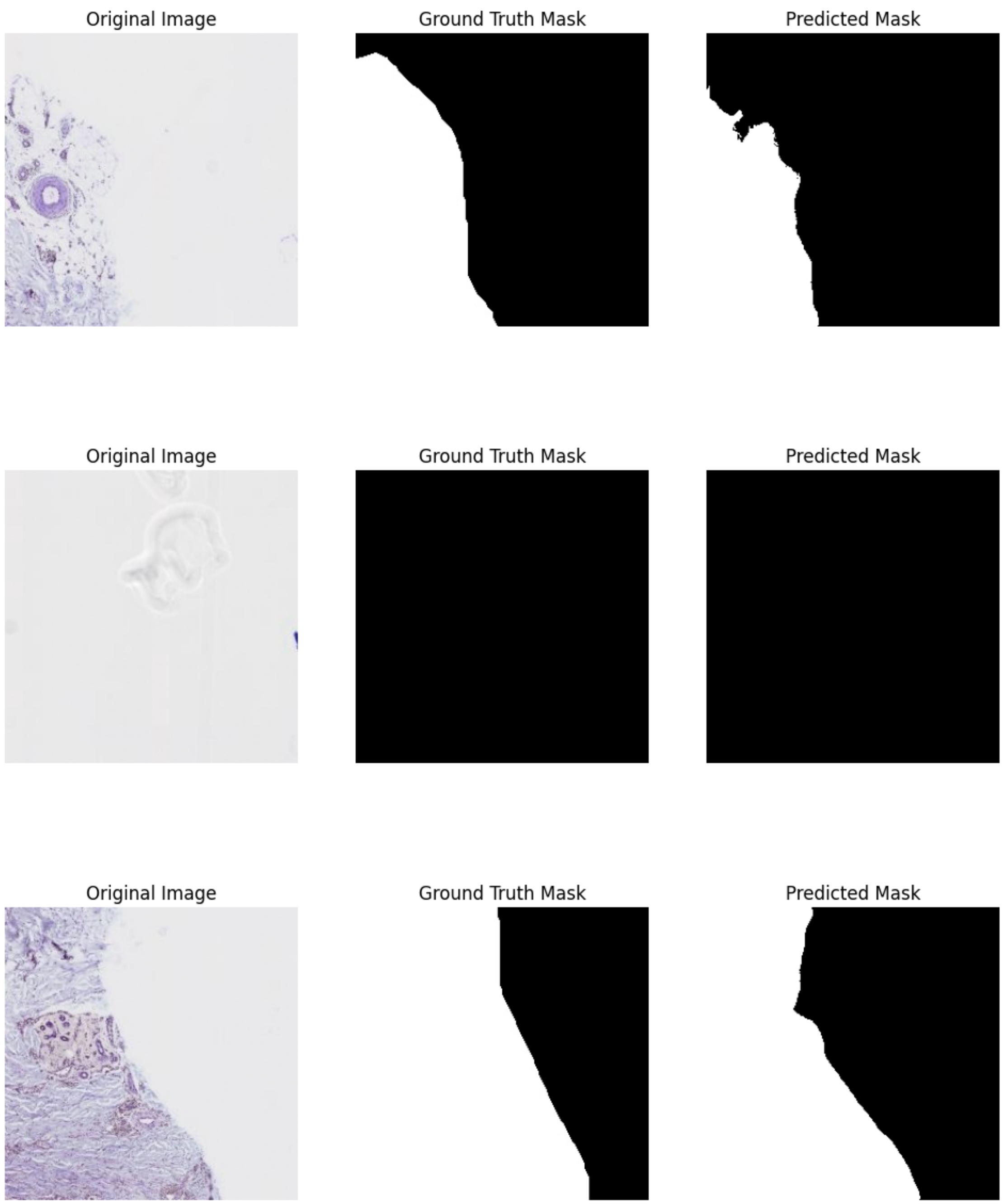
[70]:
```
# Plot validation loss
plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs+1), AV_val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Validation Loss')
plt.legend()
plt.show()
```



Plotting 3 random sample with their masks and predicted result

[66]:
```
# Set model to evaluation mode
model_pre.eval()

# Plot three random samples
for _ in range(3):
    # Get a random sample from the validation set
    sample = val_dataset[random.randint(0, len(val_dataset))]

    # Extract image and mask
    image = sample['image'].unsqueeze(0)
    mask = sample['mask'].unsqueeze(0)

    # Make prediction
    with torch.no_grad():
        prediction = model_pre(image)

    # Convert prediction to binary mask
    predicted_mask = torch.sigmoid(prediction) > 0.5

    # Plot the images
    fig, axes = plt.subplots(1, 3, figsize=(12, 4))
    axes[0].imshow(image.squeeze(0).permute(1, 2, 0))
    axes[0].set_title('Original Image')

    axes[1].imshow(mask.squeeze(0).squeeze(0), cmap='gray')
```

```
axes[1].set_title('Ground Truth Mask')

axes[2].imshow(predicted_mask.squeeze(0).squeeze(0), cmap='gray')
axes[2].set_title('Predicted Mask')

for ax in axes:
    ax.axis('off')

plt.show()
```



[71]:
```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[72]:
```
!jupyter nbconvert --to pdf /content/drive/MyDrive/TestDiaDeep/
↪Applying_SMP_Task3.ipynb
```

```
[NbConvertApp] Converting notebook
/content/drive/MyDrive/TestDiaDeep/Applying_SMP_Task3.ipynb to pdf
[NbConvertApp] Support files will be in Applying_SMP_Task3_files/
[NbConvertApp] Making directory ./Applying_SMP_Task3_files
[NbConvertApp] Making directory ./Applying_SMP_Task3_files
[NbConvertApp] Making directory ./Applying_SMP_Task3_files
[NbConvertApp] Making directory ./Applying_SMP_Task3_files
[NbConvertApp] Making directory ./Applying_SMP_Task3_files
[NbConvertApp] Writing 76724 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 393937 bytes to
/content/drive/MyDrive/TestDiaDeep/Applying_SMP_Task3.pdf
```