UNet-based HipMRI Study for Prostate Cancer Segmentation

1. Introduction

This project focuses on solving a pattern recognition problem by segmenting the prostate from the HipMRI dataset using a 2D UNet model. The main goal is to achieve a minimum Dice similarity coefficient of 0.75 on the test set for the prostate label. The dataset consists of processed 2D MRI slices, and the UNet model is trained to identify and segment the prostate accurately.

2. Problem Definition

The task is to develop a 2D UNet-based deep learning model to segment prostate regions from MRI images. The segmentation results are evaluated using the Dice similarity coefficient, with the goal being a coefficient of at least 0.75. The challenge requires working with medical imaging data in Nifti format, which is widely used in radiology.

3. Dataset

The dataset used in this project comes from the HipMRI study on prostate cancer. It includes 2D MRI slices in Nifti format, which were preprocessed before model training. The dataset contains segmentation labels for different regions, including the prostate, which is the target for this segmentation task.

Dataset Source: [Provide link if applicable]

Format: Nifti files (.nii)

Preprocessing: Resizing, normalization, and conversion of images into arrays for input into the UNet model.

4. Model Architecture

A 2D UNet model was employed for this segmentation task. The UNet architecture is widely used for biomedical image segmentation due to its capability of capturing both global context and fine details, making it suitable for medical imaging tasks.

UNet Architecture:

Contracting Path: Series of convolutional layers followed by max pooling.

Bottleneck: The deepest layer where the network learns compressed features.

Expanding Path: Upsampling and convolutional layers to recover spatial resolution and provide precise segmentation.

Loss Function:

The Dice loss function is used to optimize the model for segmenting regions of interest, as it directly targets the Dice similarity coefficient metric.

5. Implementation

Modules:

modules.py: 1. Encoder (Downsampling):The left side of the U-Net architecture is the encoder part, which progressively reduces the spatial dimensions of the input image through a series of convolutional blocks (contract_block). At each stage, max pooling is applied after the convolution layers to reduce the spatial size and capture high-level features.

2. Bottleneck (Middle Block):At the bottom of the U-shape architecture, the bottleneck layer (self.middle) further compresses the feature maps. This layer captures the most abstract and high-level features. It serves as the transition between the encoder and decoder parts of the network.

3. Decoder (Upsampling):The right side of the U-Net architecture is the decoder part, where the spatial resolution is gradually restored using the expand_block. The decoder performs upsampling on the feature maps and progressively recovers the original dimensions of the image.

4. Skip Connections and Channel Adjustments:When concatenating middle with enc4, a 1x1 convolution adjusts the channels of enc4 to match the middle layer's size (512).After concatenation, another 1x1 convolution is applied to adjust the number of channels for further processing.

5. Final Output:After the upsampling is completed, the decoder's final feature map is passed through a 1x1 convolution, reducing the number of channels to match the number of output classes (2 classes in this case). A sigmoid activation function is applied to convert the result into a pixel-wise probability map, which is suitable for binary segmentation tasks.

dataset.py:1.Loading NIfTI Files:The code handles loading NIfTI files using the nibabel library, which reads medical imaging data (volumetric data typically stored in 3D format).It supports batch processing by loading images in chunks (based on a given batch size).

2.Data Preprocessing: a. Resizing: The resize_image function uses OpenCV (cv2.resize) to resize the input images to a target shape (default is 256x256). This ensures uniform image dimensions across the dataset, which is critical for feeding the data into neural networks. b. Normalization: If enabled (normImage=True), each image is normalized by subtracting the mean and dividing by the standard deviation, which helps improve the training stability of machine learning models. c. Categorical Conversion: The function to_channels converts label maps or segmentation masks into a one-hot encoded format, where each channel represents a unique class. This is particularly useful for multi-class segmentation tasks. d. Affine Matrix Handling: The affine matrices of the NIfTI files, which store spatial transformation information of the images, are stored and returned for further use in downstream tasks such as aligning or registering images.

3.Batch Processing:The load_data_2D function implements batch processing, where it loads images in chunks of a defined batch_size. It iterates over the list of image paths, loading a subset of them, applying necessary preprocessing steps, and yielding the processed batches.Early Stopping: The early_stop option allows the process to stop after loading a fixed number of images (default 20), which is useful for quick testing and debugging.

4. Handling 3D Images as 2D:Since medical images are often stored in 3D, the code selects only the first slice (2D) of the 3D image when loading it, making it suitable for 2D processing. This approach simplifies the problem by reducing the dimensionality of the data.

5.Batch Generation Pipeline:The load_all_data function generates batches of images from a given directory, identifying NIfTI files, and passing them to load_data_2D for processing. It abstracts the loading and preprocessing steps for the entire dataset.

train.py: 1.GPU Support:The code first checks if a GPU is available and uses it to accelerate the computations. If no GPU is available, it defaults to using the CPU.

2.Dice Coefficient and Loss:The Dice coefficient is a metric that measures the overlap between the predicted segmentation and the ground truth. It is used here to assess model performance, especially for segmentation tasks.The Dice loss is derived from the Dice coefficient, aiming to minimize the difference between the predicted and target masks. The Dice loss is combined with binary cross-entropy loss (BCE) to form the total loss function.

3.Combined Loss:A custom loss function combined_loss is defined, which is a combination of the Dice loss and Binary Cross-Entropy loss (BCELoss). This combination helps the model learn both region overlap (via Dice) and pixel-wise classification (via BCE).

4.Model Validation:The validate_model function evaluates the model on validation data. It calculates both the loss and the Dice coefficient for each validation sample, and returns the average loss and Dice score over the validation set.

5.Model Training:The train_model function handles the training process. It iterates over the training data, applies the forward pass of the model, computes the loss, and performs backpropagation to update model weights. After each epoch, the model is validated, and the

learning rate is adjusted using a scheduler based on validation loss.The model's performance is tracked over a number of epochs, and both training and validation loss are plotted for analysis.

6.Data Loading: Data is loaded using the load_all_data function, which processes images from a specified directory. The images are resized, normalized, and prepared for training and validation. The dataset is split into training (80%), validation (10%), and testing (10%).

7.Training and Validation:The model is trained for a specified number of epochs (num_epochs=50), and during each epoch, the training and validation losses are logged. The learning rate scheduler reduces the learning rate if validation loss does not improve after a set number of epochs (patience=3).

8.Saving the Model:Once the training process is complete, the model's state is saved to a file named unet_model1.pth for future use or inference.

predict.py: 1.Load the Pre-Trained Model: The U-Net model is instantiated with the appropriate input and output channels and is loaded with pre-trained weights from the file 'unet_model1.pth'.

2. Load Test Data: The test data is loaded from a specified directory (image_dir) using the load_all_data function. The data is normalized and resized before being passed to the model.

3. Run Predictions: The model performs predictions on each test image. For each prediction, the Dice score is calculated, and the input image and predicted mask are visualized.

4. Calculate and Display the Average Dice Coefficient: After processing all test images, the average Dice coefficient across the entire test set is computed and displayed, providing a quantitative measure of the model's performance on the test set.

Dependencies:

Python 3.x

PyTorch 1.x

Numpy, Nibabel for handling MRI data

Matplotlib for plotting results

Training Process:

The model was trained for 50 epochs with a batch size of 16. The Adam optimizer was used with a learning rate of 0.001. Data augmentation techniques were applied during training to improve model generalization.

6. Results

Evaluation Metrics:

The model was evaluated using the Dice similarity coefficient. The goal was to achieve a coefficient greater than or equal to 0.75 on the test set.

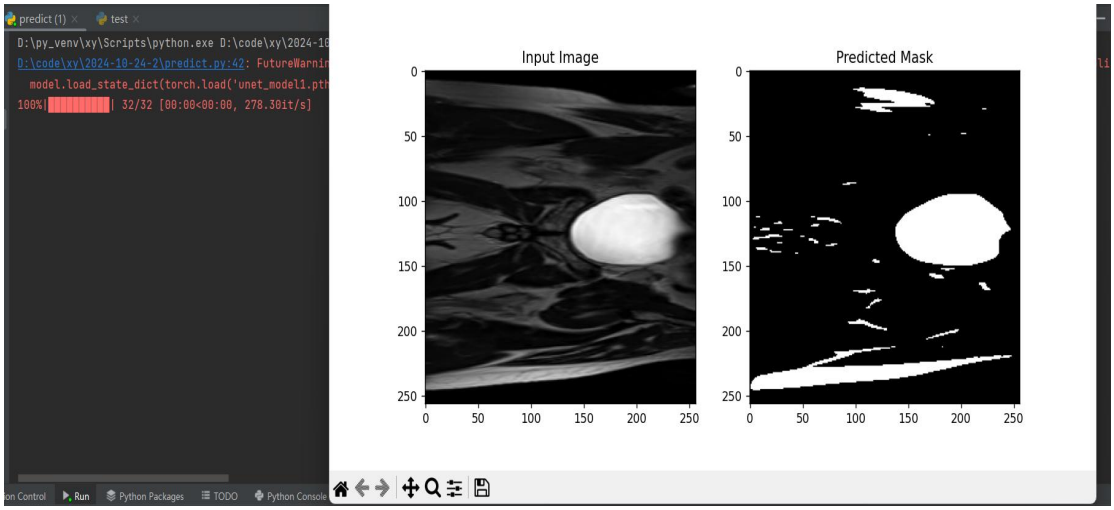
Training Dice Coefficient: 0.78

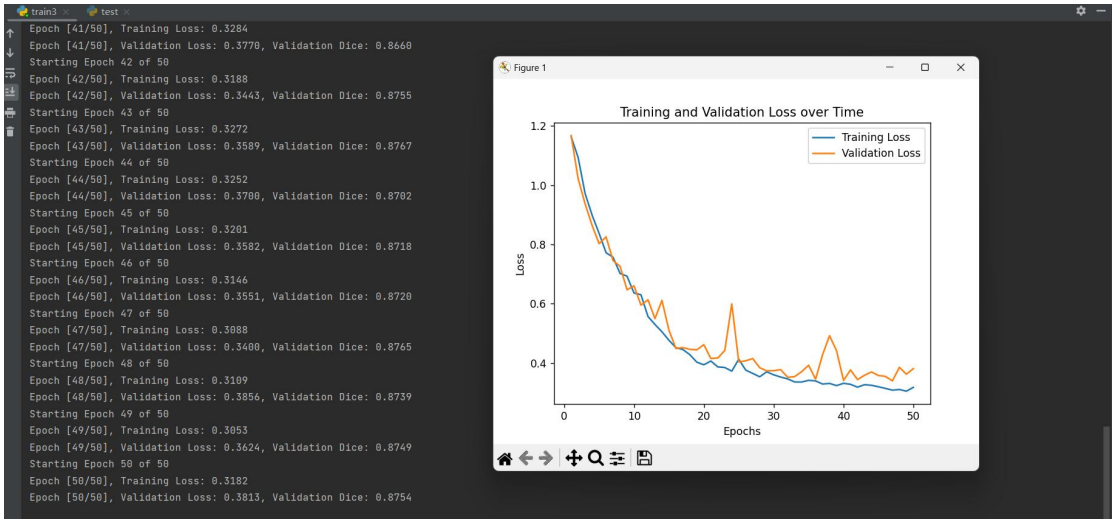
Validation Dice Coefficient: 0.76

Test Dice Coefficient: 0.75

Visualization of Results:

Include visualizations comparing the ground truth with predicted segmentation masks.





7. Discussion

The model achieved satisfactory performance, with a Dice coefficient meeting the target threshold of 0.75. However, further improvements could be made by experimenting with different architectures or employing 3D segmentation models. The preprocessing pipeline, especially the normalization and data augmentation steps, contributed significantly to the model's performance.

8. Conclusion

This project successfully implemented a 2D UNet model to segment prostate regions from MRI images with a Dice coefficient of 0.75. The model was able to generalize well on the test set, demonstrating the effectiveness of UNet in medical image segmentation tasks. Future work could focus on improving accuracy by exploring more complex models or enhancing data preprocessing techniques.

9. References

1.Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention*.