AI Lab Project: Image Segmentation UNet model

Konstantinos Kafteranis (2162765)
Jon Dorronsoro (2168329)

Date: June 3, 2024

Abstract

This project leverages PyTorch for image segmentation of satellite imagery, utilizing a deep learning model to delineate and classify various land cover types.

1 Introduction

Satellite imagery provides a wealth of information essential for monitoring and managing our environment. However, effectively interpreting this data requires sophisticated techniques to extract meaningful insights from the vast and complex images. Image segmentation, a process of partitioning images into distinct regions, plays a crucial role in this context. This project uses PyTorch, a leading deep learning framework, to experiment with image segmentation models tailored for satellite images.

2 Methodology

This project is inspired by a video workshop on a similar implementation utilizing TensorFlow and is focused on developing a modular codebase for image segmentation of satellite images using PyTorch. Our approach is structured to ensure clarity and reusability, dividing the core components into datasets, models, training, and utilities.

The utilities module contains functions for general operations and calculations, such as directory creation, which are not directly related to the model but are frequently used throughout the project. The dataset module features a custom data loader designed to efficiently manage our specific satellite imagery dataset.

Although largely a reconstruction of the original implementation, our approach is object-oriented, enhancing maintainability and extensibility. A significant improvement is the accurate association of colors to classes. We achieved this by creating a dictionary to map each class to its corresponding RGB values and implementing a function to assign these RGB values to the correct labels.

The training module is a critical component of our image segmentation project, responsible for orchestrating the training process of the UNet model using the satellite image dataset. This module integrates various elements including data loading, model training, evaluation, and logging of metrics.

2.1 Training Module

1. **Dataset Preparation**: The dataset is prepared using a custom data loader from the SatelliteImageSegmentation class, which processes the satellite imagery and corresponding masks. The dataset is then split into training and validation sets.

2. **DataLoader Initialization**: PyTorch's DataLoader is used to create iterators for the training and validation datasets, allowing efficient batch processing and shuffling of data during training.

3. **Model, Loss Function, and Optimizer**: The UNet model is initialized along with the cross-entropy loss function and the Adam optimizer. The model is then moved to the appropriate device (CPU or GPU) for training.

4. **Training Loop**: The training process runs for a specified number of epochs. In each epoch, the model is set to training mode, and a forward pass is performed on each batch of images. The loss is computed and backpropagation is used to update the model parameters. Metrics such as Jaccard index, Dice coefficient, pixel accuracy, precision, and recall are calculated for each batch to monitor performance.

5. **Validation Loop**: After each training epoch, the model is evaluated on the validation set. The average validation loss is computed to assess the model's performance on unseen data.

6. **Metrics Logging**: Training and validation metrics are logged to a CSV file after each epoch. This includes the average loss, Jaccard index, Dice coefficient, pixel accuracy, precision, and recall, providing a comprehensive overview of the model's performance over time.

7. **Model Checkpointing**: The model's state is saved at the end of each epoch, allowing for checkpointing and potential recovery if needed. The final trained model is also saved at the end of the training process.

At the heart of our model module is a simple yet effective UNet implementation. This architecture is well-suited for image segmentation tasks, enabling precise delineation and classification of various land cover types in satellite images. Our enhancements and structured approach aim to improve the functionality and accuracy of satellite image segmentation, making it a valuable tool for environmental monitoring, urban planning, and disaster management.

3 Data

The dataset used for training the U-Net model is the "Semantic Segmentation of Aerial Imagery" dataset from Kaggle. This dataset consists of high-resolution aerial images captured by drones. Each image is paired with a corresponding mask that annotates different objects in the scene, such as buildings, roads, vegetation, water bodies, and other classes. The annotations are pixel-level, enabling precise segmentation tasks. The dataset is designed to support various applications in urban planning, environmental monitoring, and disaster response by providing detailed and accurate segmentations of urban and rural areas.

Link to dataset: `https://www.kaggle.com/datasets/humansintheloop/semantic-segmentation-of-aerial-imagery`

0.45

Figure 1: RGB Image

0.45

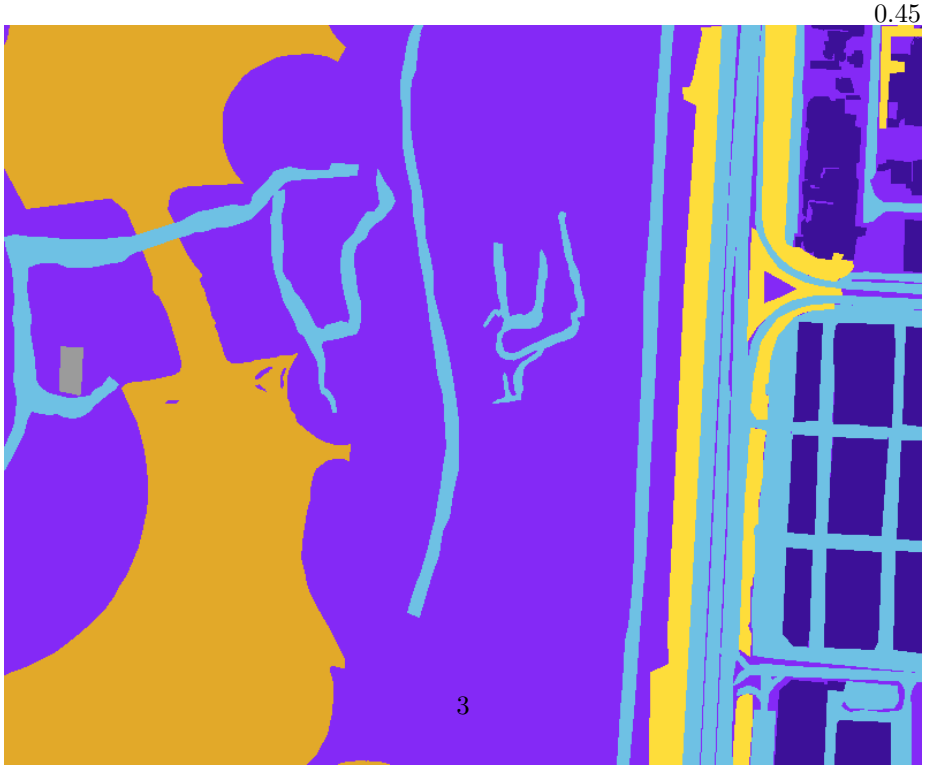

Figure 2: Mask Image

4 Model

The U-Net model implemented in this project is designed for semantic segmentation tasks using the PyTorch library. Here's a detailed description of the model components:

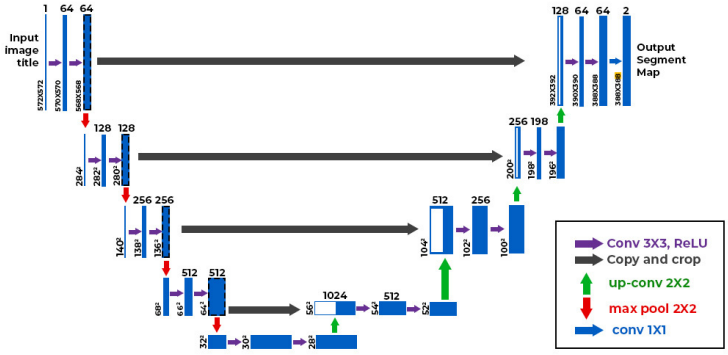

Figure 3: UNet Architecture

- **Conv_Block**: A custom convolutional block consisting of two convolution layers, each followed by a ReLU activation function and dropout for regularization. This block is used throughout the encoder and decoder to extract and refine features.
- **Encoder**: The encoder comprises four convolutional blocks, each followed by a max-pooling layer to progressively reduce the spatial dimensions while increasing the depth of the feature maps. The final output of the encoder is passed through a bottleneck layer, which further compresses the feature maps.
- **Decoder**: The decoder uses transposed convolutional layers to upsample the feature maps and restore the original spatial dimensions. After each upsampling, the corresponding feature maps from the encoder are concatenated to retain spatial information. Convolutional blocks refine these concatenated features.
- **Final Layer**: The final layer of the U-Net is a 1x1 convolution that maps the refined feature maps to the desired number of output classes, producing a multi-class segmentation map.

The model architecture can be summarized as follows:

- **Input**: RGB image of size 256x256.
- **Encoder**:

  - Conv_Block (3, 16) → MaxPool
  - Conv_Block (16, 32) → MaxPool
  - Conv_Block (32, 64) → MaxPool
  - Conv_Block (64, 128) → MaxPool
  - Bottleneck: Conv_Block (128, 256)

- **Decoder**:
  - UpConv (256, 128) → Conv_Block (256, 128)
  - UpConv (128, 64) → Conv_Block (128, 64)
  - UpConv (64, 32) → Conv_Block (64, 32)
  - UpConv (32, 16) → Conv_Block (32, 16)

- **Output**: 1x1 Conv (16, num_classes)

4.1 Example Usage

```
# Test
if __name__ == "__main__":
    num_classes = 6
    model = UNet(num_classes)
    x = torch.randn(1, 3, 256, 256)  # batch size 1, 3 color channels, 256x256 image
    output = model(x)
    print(output.shape)  # result [1, 6, 256, 256]
```

This U-Net model is effective for semantic segmentation tasks, particularly for high-resolution aerial imagery, by leveraging its encoder-decoder architecture to capture and reconstruct spatial features accurately.

5 Results

The model shows promising progress across epochs, indicating its capability to accurately segment images. Initially, the model starts with modest segmentation accuracy, as seen from the average loss of 1.67 in the first epoch. However, as training continues, there's a noticeable improvement. By the 25th epoch, the average loss decreases to 0.72, showing that the model is getting better at capturing detailed image features. Metrics like Jaccard, Dice, Pixel Accuracy, Precision, and Recall also improve consistently, suggesting that the model is learning to create more precise segmentation masks over time. This indicates that the UNet model has the potential to be a reliable tool for various image segmentation tasks.

An intriguing part of the results to observe is the relationship between the average training loss and the average validation loss across epochs. Initially, during the early epochs, the average training loss tends to be higher than the

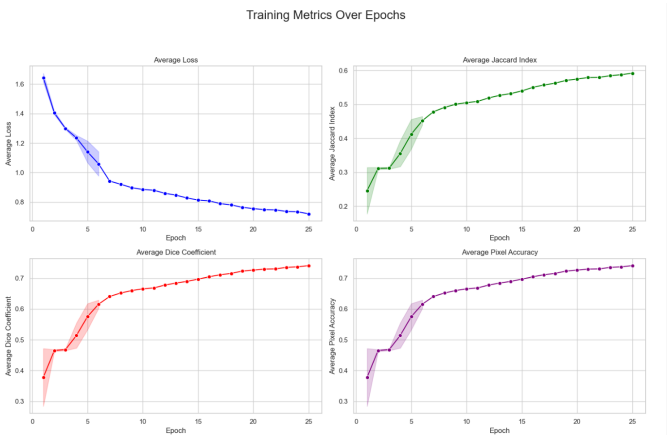

Figure 4: Metrics Over Epochs

average validation loss. This discrepancy suggests that the model is effectively learning from the training data but may struggle slightly when generalizing to unseen validation data. However, as training progresses, the average training loss steadily decreases, converging towards the average validation loss. This convergence indicates that the model is becoming more adept at generalizing its learned features to unseen data, resulting in a comparable performance between the training and validation sets. Ultimately, this alignment between the training and validation losses signifies a well-trained model that is capable of effectively capturing the underlying patterns in the data while maintaining its ability to generalize to new, unseen instances.

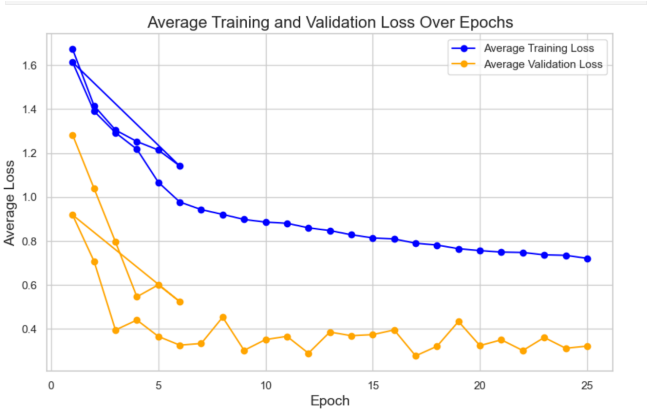

Figure 5: Training vs Validation Loss

6 Resources

- Video Workshop Tutorial
- PyTorch Documentation
- GeeksforGeeks PyTorch Page
- OpenCV Documentation
- LearnPytorch
- U-Net: A PyTorch Implementation in 60 lines of Code
- Understanding Semantic Segmentation with UNET
- Practical Deep Learning
- UNet Video Tutorial
- U-Net: Convolutional Networks for Biomedical Image Segmentation