Water Body Segmentation from Satellite Images using U-Net

Deep Learning Project Work

Irene Burri
Student ID: 0001120380

May 2, 2025

1 Introduction

Accurate identification of water bodies like rivers, lakes, oceans and artificial basins from satellite images is essential for environmental monitoring, water resource management and disaster mitigation.

This project aims to implement a Deep Learning model based on the U-Net architecture for the automatic segmentation of water regions in satellite imagery. The goal is to enable fast, reliable and scalable identification of water regions within satellite images, offering a significant improvement over traditional methods in both efficiency and accuracy.

2 Dataset

The dataset used is a collection of water bodies images captured by the Sentinel-2 Satellite and it's available on Kaggle[1].

Each RGB image comes with a black and white mask where white represents water and black represents something else but water. The masks were generated by calculating the NWDI (Normalized Water Difference Index) which is frequently used to detect and measure vegetation in satellite images, but a greater threshold was used to detect water bodies.

Dataset structure:

- **Number of images and corresponding masks:** 2841
- **Format:** .jpg
- **Size:** variable (resized to 256×256 during preprocessing)

[1]`https://www.kaggle.com/datasets/franciscoescobar/satellite-images-of-water-bodies/data`

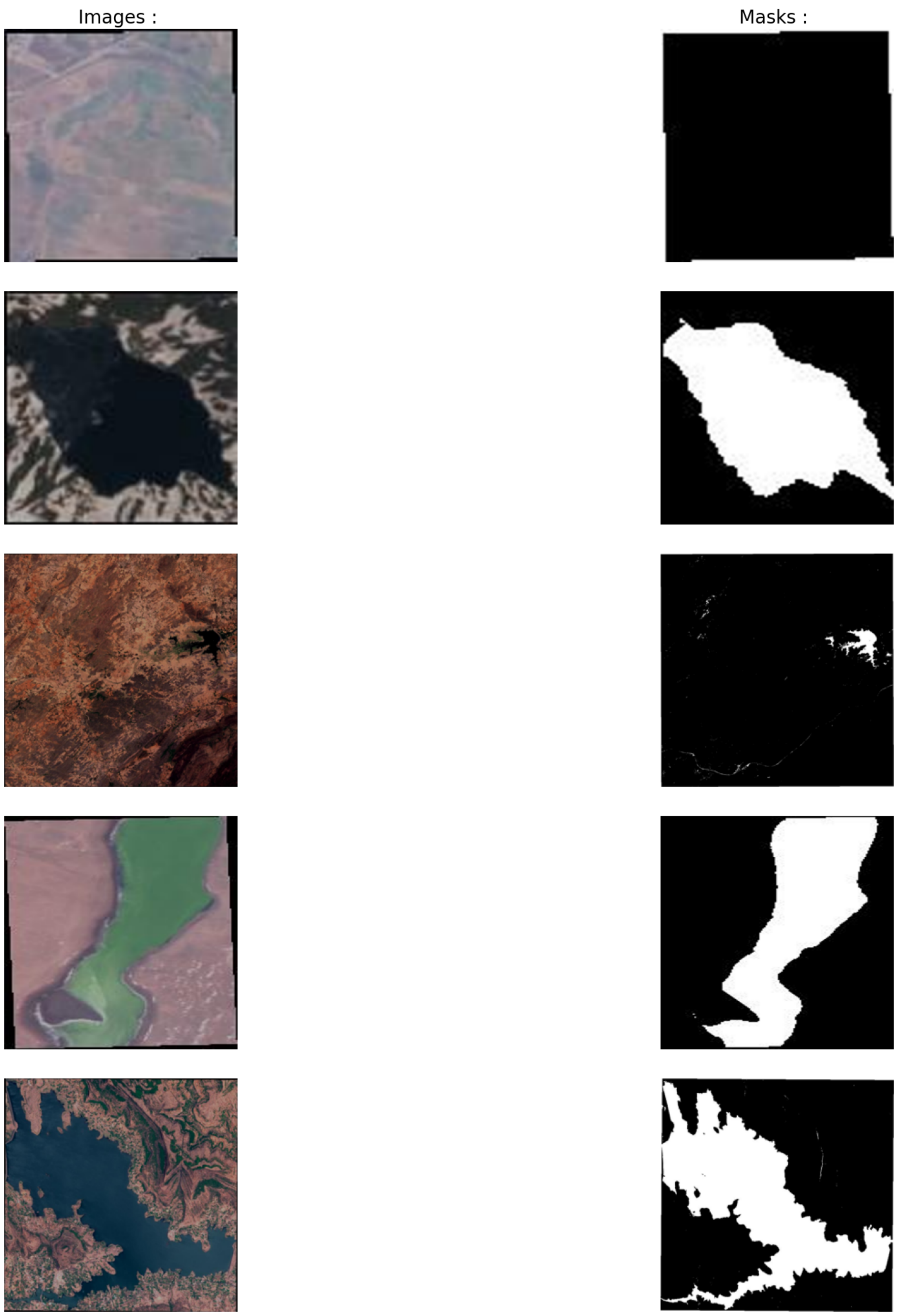


Figure 1: Example satellite images from the dataset showing the area of interest for water body segmentation.

3 Notebook Structure

3.1 Importing Essential Libraries

To develop and train the deep learning model for water body segmentation, several essential Python libraries were imported.

- TensorFlow: This is the core deep learning framework used to define, train, and evaluate the U-Net architecture. It provides high-level APIs for building neural networks.
- PIL: Used to load and process images. It plays a key role in handling image input and converting them into formats suitable for model training.
- NumPy
- Matplotlib
- Tqdm

3.2 Data Loading

To handle the large dataset size, it was uploaded as multiple ZIP files to a GitHub[2] repository and later extracted locally for use.

3.3 Data Exploration

The data has already been filtered and appears clean enough for further processing. Each file has a unique name, ensuring there are no duplicates.

Figure 2: Satellite image and its corresponding mask.

3.4 Data Preparation

The dataset was split into training and validation sets with an 80/20 ratio.

- X_train contains 2272 images.
- Y_train contains 2272 masks.
- X_val contains 569 images.
- Y_val contains 569 masks.

[2]`https://github.com/ireneburri/PW_DeepLearning.git`

3.5 Custom Data Generator for Image Segmentation

I define a custom data generator using `tf.keras.utils.Sequence` for efficiently loading and preprocessing satellite images and their corresponding masks in batches, since the dataset is big.

The `ImageSegmentationGenerator` inherits from the Keras Sequence class, ensuring that data loading is done in a multi-threaded and efficient way.

The generator is used during the training and validation phases to provide batches of data for model training.

Arguments:

- `image_filenames`: List of filenames for the input images.
- `mask_filenames`: List of filenames for the corresponding segmentation masks.
- `image_dir`: Directory path where the images are stored.
- `mask_dir`: Directory path where the masks are stored.
- `batch_size`: Number of samples per batch.
- `img_size`: Desired size for the images (height, width).
- `shuffle`: Whether to shuffle the data after each epoch.

The generator ensures that the images and masks are preprocessed correctly and batches are returned in a format suitable for training the model.
Applied preprocessing:

- Converting images to RGB and masks to greyscale
- Resizing
- Normalization of mask values to [0, 1]

3.6 U-Net Model Architecture

I decided to use a U-Net model since it's a type of convolutional neural network designed for semantic segmentation tasks, where the objective is to classify each pixel of an image.

The model consists of three main parts:

- Encoder: Extracts features from the input image using convolutional layers followed by max-pooling operations.
- Bottleneck: This part is the narrowest part of the U-Net, where the network learns the most abstract representations of the image.
- Decoder: Expands the feature maps back to the original image size using transposed convolutions, enabling pixel-wise segmentation.

ReLU activations are used in convolutional layers and batch normalization for regularization, which helps stabilize the training process. The final layer outputs a binary segmentation map using a sigmoid activation function.

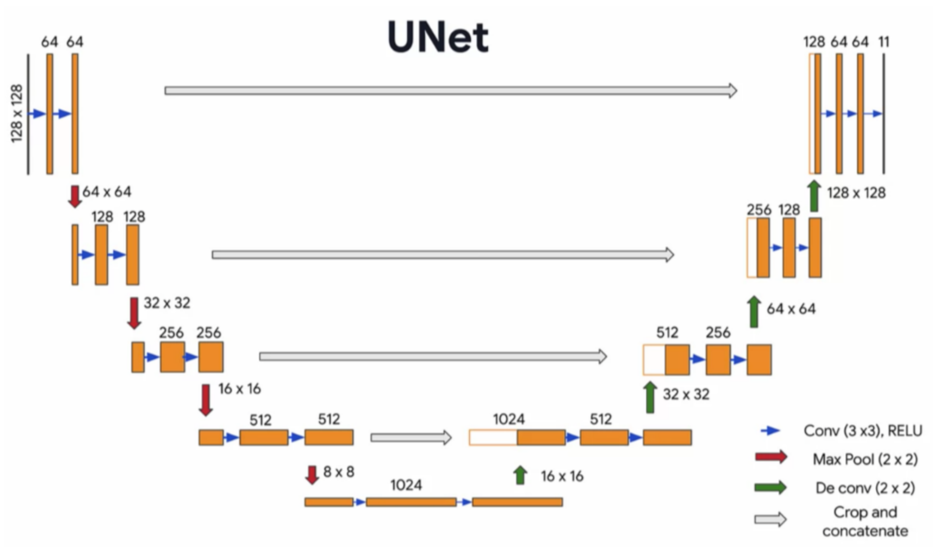


Figure 3: U-Net Architecture.

Model Summary and Size The resulting model is quite large, with more than 31 million parameters in total. This includes:

- Trainable parameters: 31,043,521
- Non-trainable parameters: 11,776

3.7 Compilation and Training

The U-Net model was compiled with the following:

- Optimizer: **Adam** with an initial learning rate $= 1 \cdot 10^{-4}$
- Loss function: **Binary Crossentropy**, suitable for water vs non-water segmentation.
- Metric: **Accuracy**

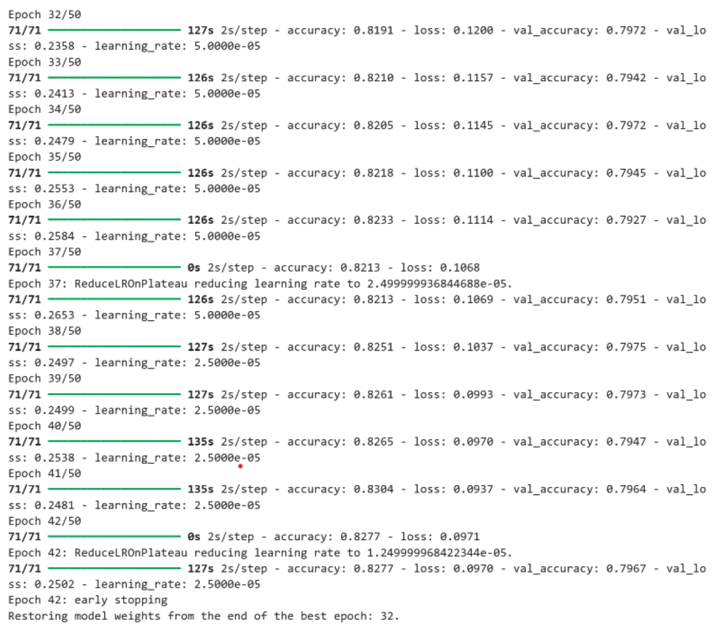
We also define two callbacks to prevent overfitting and improve model convergence:

- **ReduceLROnPlateau**: This reduces the learning rate by a factor of 0.5 if the validation loss plateaus for 5 consecutive epochs, with a minimum learning rate of $1 \cdot 10^{-6}$.
- **EarlyStopping**: Stops training if the validation loss does not improve for 10 epochs, restoring the model to the best weights seen.

3.8 Model Training

The `ImageSegmentationGenerator` explained in 3.5 is used to load batches of images and masks.

- Batch Size: 32 images per batch.
- Epochs: Train the model for up to 50 epochs with the defined callbacks for early stopping and learning rate reduction.



```
Epoch 32/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 127s 2s/step - accuracy: 0.8191 - loss: 0.1200 - val_accuracy: 0.7972 - val_lo
ss: 0.2358 - learning_rate: 5.0000e-05
Epoch 33/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 126s 2s/step - accuracy: 0.8210 - loss: 0.1157 - val_accuracy: 0.7942 - val_lo
ss: 0.2413 - learning_rate: 5.0000e-05
Epoch 34/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 126s 2s/step - accuracy: 0.8205 - loss: 0.1145 - val_accuracy: 0.7972 - val_lo
ss: 0.2479 - learning_rate: 5.0000e-05
Epoch 35/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 126s 2s/step - accuracy: 0.8218 - loss: 0.1100 - val_accuracy: 0.7945 - val_lo
ss: 0.2553 - learning_rate: 5.0000e-05
Epoch 36/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 126s 2s/step - accuracy: 0.8233 - loss: 0.1114 - val_accuracy: 0.7927 - val_lo
ss: 0.2584 - learning_rate: 5.0000e-05
Epoch 37/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8213 - loss: 0.1068
Epoch 37: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
71/71 ━━━━━━━━━━━━━━━━━━━━ 126s 2s/step - accuracy: 0.8213 - loss: 0.1069 - val_accuracy: 0.7951 - val_lo
ss: 0.2653 - learning_rate: 5.0000e-05
Epoch 38/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 127s 2s/step - accuracy: 0.8251 - loss: 0.1037 - val_accuracy: 0.7975 - val_lo
ss: 0.2497 - learning_rate: 2.5000e-05
Epoch 39/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 127s 2s/step - accuracy: 0.8261 - loss: 0.0993 - val_accuracy: 0.7973 - val_lo
ss: 0.2499 - learning_rate: 2.5000e-05
Epoch 40/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.8265 - loss: 0.0970 - val_accuracy: 0.7947 - val_lo
ss: 0.2538 - learning_rate: 2.5000e-05
Epoch 41/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.8304 - loss: 0.0937 - val_accuracy: 0.7964 - val_lo
ss: 0.2481 - learning_rate: 2.5000e-05
Epoch 42/50
71/71 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8277 - loss: 0.0971
Epoch 42: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.
71/71 ━━━━━━━━━━━━━━━━━━━━ 127s 2s/step - accuracy: 0.8277 - loss: 0.0970 - val_accuracy: 0.7967 - val_lo
ss: 0.2502 - learning_rate: 2.5000e-05
Epoch 42: early stopping
Restoring model weights from the end of the best epoch: 32.
```

Figure 4: Model Training from the best epoch, 32, to the early stopping.

3.9 Model Results Analysis

The model showed consistent improvement in accuracy and reduction in loss during training.

These results indicate that the model successfully learned to segment water regions, outperforming traditional methods in scalability and precision.

Model Performance and Early Stopping

- Best Epoch: **Epoch 32** achieved the best validation loss, and the model was restored to this checkpoint. Early stopping was correctly applied to avoid unnecessary training and potential degradation.
- Key Metrics:
  - Training Accuracy: 81.91%
  - Validation Accuracy: **79.72%**
  - Training Loss: 0.1200
  - Validation Loss: **0.2358**

- Learning Rate Scheduling: The use of ReduceLROnPlateau was appropriate: it kicked in at epoch 42, as shown in 4, reducing the learning rate to fine-tune model weights further.

Comparison plot of Training Accuracy vs. Validation Accuracy

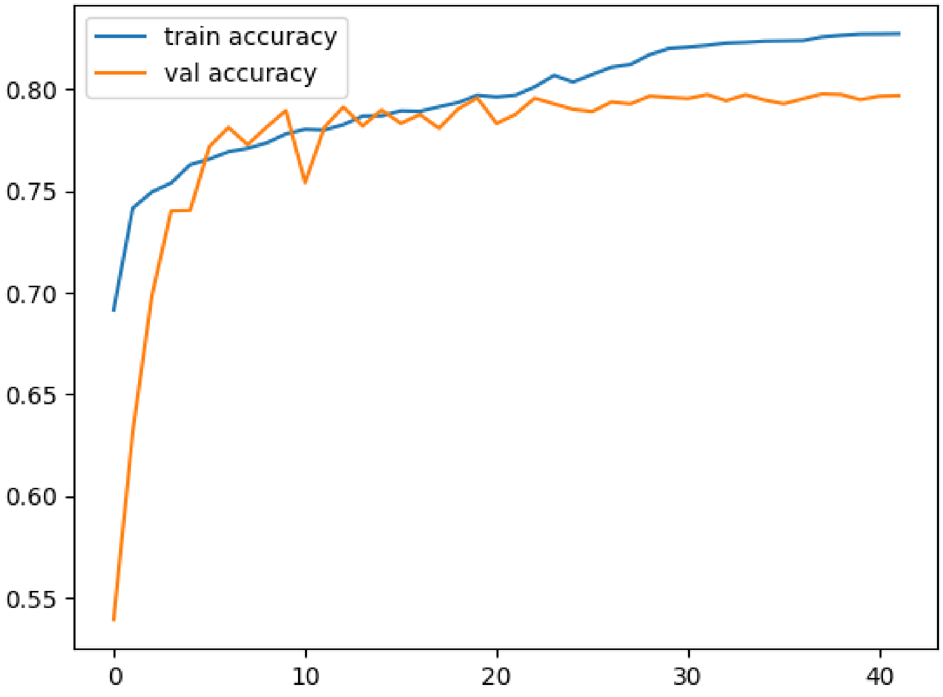


Figure 5: Training Accuracy vs. Validation Accuracy

- Initial Growth: The model shows a steep rise in both training and validation accuracy during the first 10–15 epochs. This is expected as the model quickly learns basic patterns and representations.
- Plateau & Gap: After around epoch 20, training accuracy continues to improve steadily, while validation accuracy begins to plateau, hovering just below 80%.
- Generalization: The small but consistent gap between training and validation accuracy indicates an overfitting. However, it's not drastic and suggests that the model is still generalizing reasonably well.

Comparison plot for Training Loss and Validation Loss

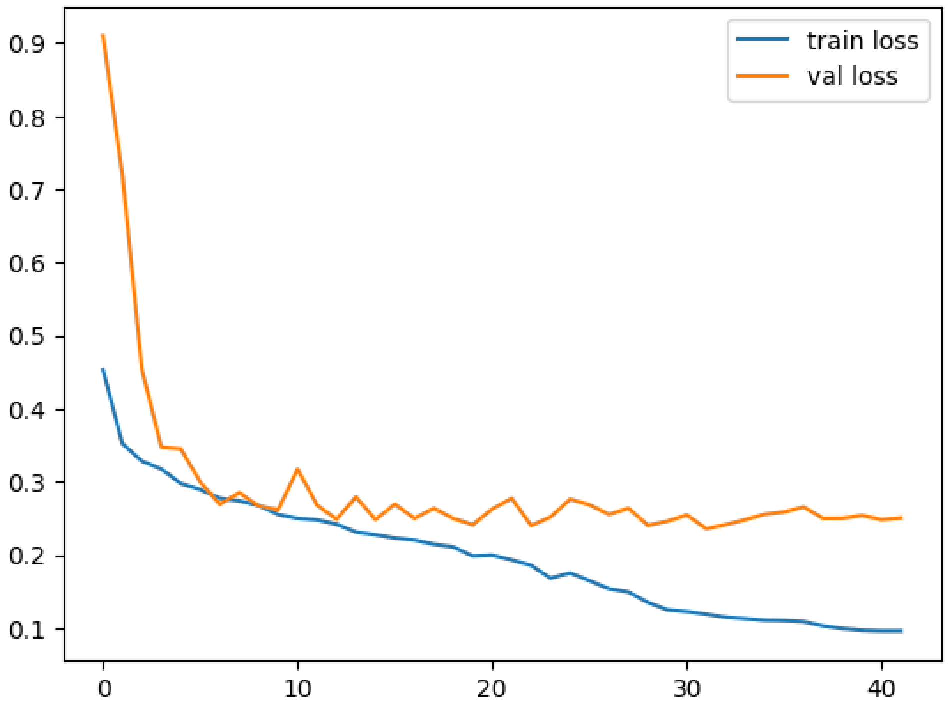


Figure 6: Training Loss vs. Validation Loss

- Loss Convergence: Training loss steadily decreases throughout, reaching about 0.097 by epoch 42, which is a strong indication of good convergence.
- Validation Loss Plateau: Validation loss drops significantly early on and then flattens out around 0.23-0.25, showing that the model is no longer significantly improving on unseen data.
- Overfitting Signs: Similar to the accuracy plot, the validation loss starts to diverge slightly from the training loss after epoch 25. This reinforces the indication of overfitting.

3.10 Prediction Results

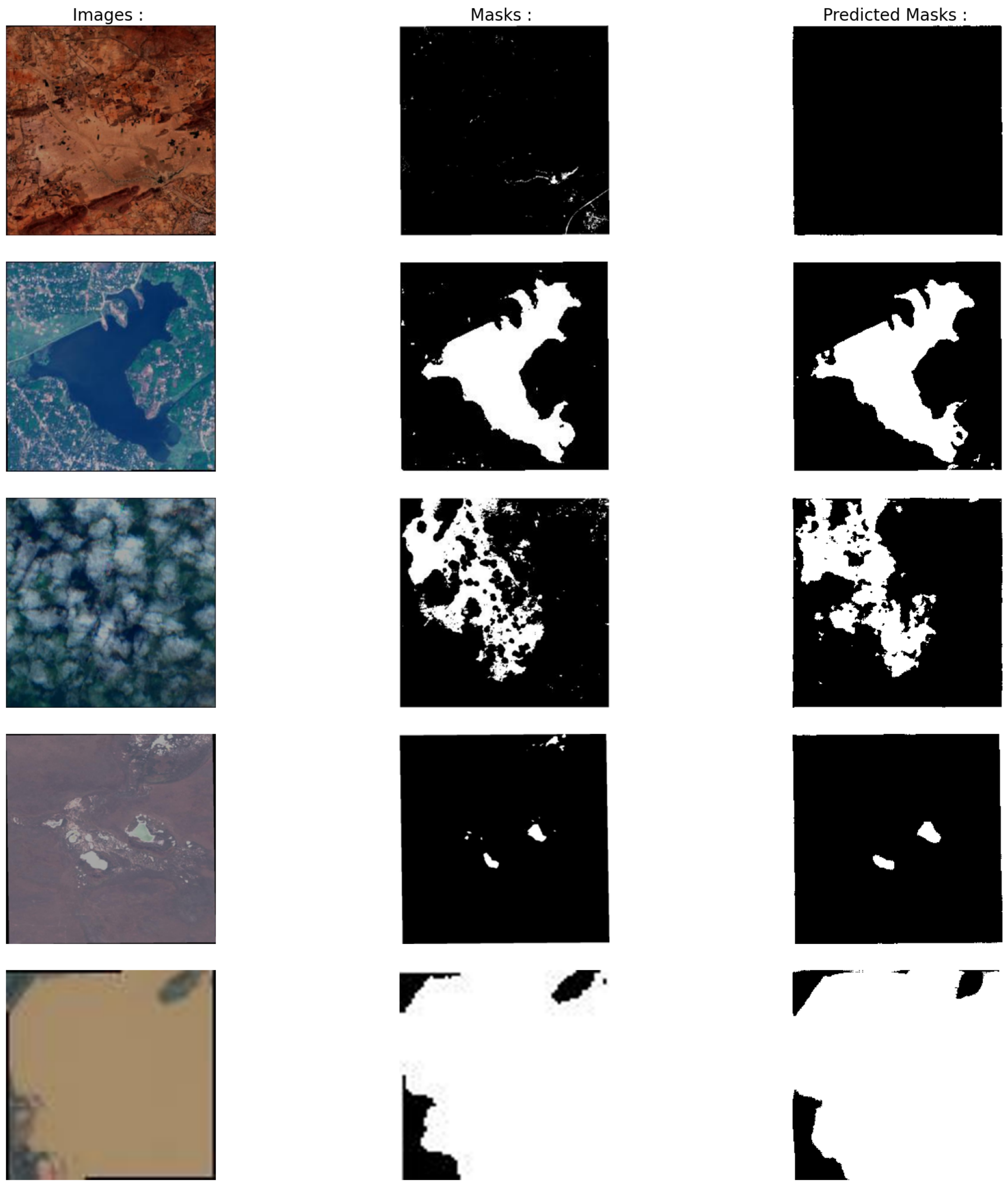


Figure 7: Example of random satellite images, their mask and the predicted water body segmentation.

4 Conclusions

The U-Net model demonstrates strong performance in segmenting water bodies from Sentinel-2 satellite imagery. The training process shows effective learning, with training accuracy reaching 81.91% and validation accuracy of 79.72% at its best epoch, with corresponding losses of 0.1200 and 0.2358 respectively. While there are signs of minor overfitting, expected given the large number of parameters (31M+), the gap between training and validation metrics remains small, indicating a well-generalized model.

Visual inspections 7 confirm the model's efficacy. Randomly selected test images reveal that the predicted masks closely align with the ground truth, accurately delineating water bodies. This pixel-level precision underscores the model's capability to handle complex satellite imagery.

To further improve performance and efficiency, one can introduce data augmentation, to introduce more variability and reduce overfitting.

Overall, this project demonstrates that deep learning models like U-Net are highly effective for the automatic segmentation of water bodies from satellite images and represents a solid foundation for further development in environmental monitoring systems.