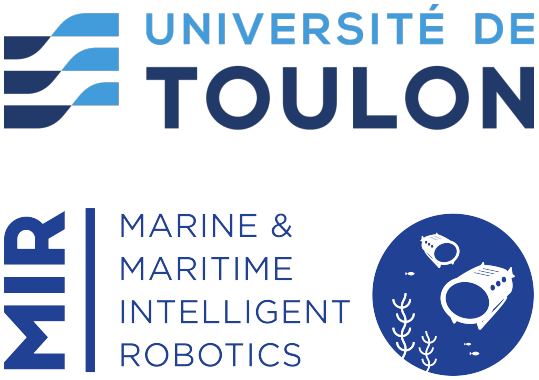


# Building a U-Net for Image Segmentation

**By**

**Hyejoo Kwon Helper : Mateus Pedrosa**

**Submitted to: Arnaud Pietrosemoli**

**Universite de Toulon**

January 31, 2025

# Contents

# Introduction

In this report, we delve into the implementation of a U-Net neural network for image segmentation using PyTorch. The dataset consists of infrared images and their corresponding segmentation masks, known as ground truth. The data was preprocessed to enhance the model's performance. We evaluated the model's effectiveness using metrics such as the Dice score, training loss, and validation loss. Additionally, we visualized the learning progress by saving and plotting the model's predictions at each epoch. The following sections of this report will detail the process of building the U-Net, including data preprocessing, model architecture, the training process, and evaluation metrics. Furthermore, we will discuss the results obtained and the challenges encountered during the implementation. Through this lab, we aim to distinguish between the sea and the sky in infrared images and gain a deeper understanding of the deep learning framework.

# 1 Key findings

In this section, I will summarize the key findings that I encountered during implementing U-net and explain how I addressed the major challenges I faced.

## 1.1 Training

During training, I encountered an error while accumulating the loss in `totalTrainLoss`. I didn't convert the tensor object to a numeric value, which caused problems during the computation process. To address this issue, I used `loss.item()` to convert the tensor to a numeric format. This allowed the loss value to accumulate properly. Now that I understand the role of the `item()` function, I will be able to implement it whenever I need to derive not only the loss value but also metrics like dice score and accuracy.

```
totalTrainLoss += loss.item()  % Convert tensor value to a number before
    accumulating
```

On top of that, I made a mistake while calculating the average training loss. After accumulating the loss for each batch, I forgot to divide by the number of batches. As a result, I was unable to calculate the correct average loss for the entire training dataset. I corrected the issue by dividing the total loss by the number of batches (len(loader)). This allowed me to calculate the correct average training loss for the entire dataset.

```
train_loss = totalTrainLoss / len(loader)
```

## 1.2 Model - Unet

At first, I didn't manage the input and output channel sizes correctly when using ConvTranspose2d. I didn't realize that when I upsample, the number of channels has to match

properly between the skip connection and the upsampled tensor x. I got an error because the channels didn't match when I tried to combine them. I need to double the input channels (since we combine the skip connection) and then reduce them back to the original number of channels after upsampling. By understanding this, I learned how important it is to match the channel sizes correctly before combining the tensors.

```
self.bottleneck = SimpleConv(in_channels, in_channels*2)
in_channels = in_channels*2
```

### 1.3 Check accuracy

I mistakenly thought the data loader provided **predicted masks** instead of **ground truth masks**, leading to misjudgement in my code. I realized that `val_masks` from the loader are **ground truth labels**, while `preds` must be generated by passing `val_images` through the model.

**Correct Code:**

```
val_images, val_masks = next(iter(loader))
ground truth masks (not predictions)
```

This mistake helped me understand that the loader only provides **input images (`val_images`) and ground truth masks (`val_masks`)**, while the predicted masks must be generated by the model.

## 2 Results

In this section, we will analyze the results of the U-Net model training for image segmentation. The model was trained for 10 epochs, and we will focus on the training loss, validation loss, and validation accuracy (Dice score) at epochs 1, 5, and 10. The goal is to understand how the model performed over time and whether it improved as expected.

### 2.1 Initial Results (Before Training - Epoch 0)

- **Dice Score**: 0.4872

Before training, the model had a Dice score of 0.4872, which is close to random guessing. This result is expected since the model had not learned anything yet.

## 2.2 Results at Epoch 1


Figure 1: Graph of evaluation loss, train loss and accuracy metric at epoch 1

- **Train Loss**: 0.0879
- **Validation Loss**: 0.0291
- **Dice Score**: 0.9789

At the first epoch, the model started with a relatively high training loss (0.0879) and a lower validation loss (0.0291). The Dice score improved significantly to 0.9789, which is a big jump from the initial score of 0.4872. This suggests that the model was able to learn some patterns early on, but there was still room for improvement.

## 2.3 Results at Epoch 5

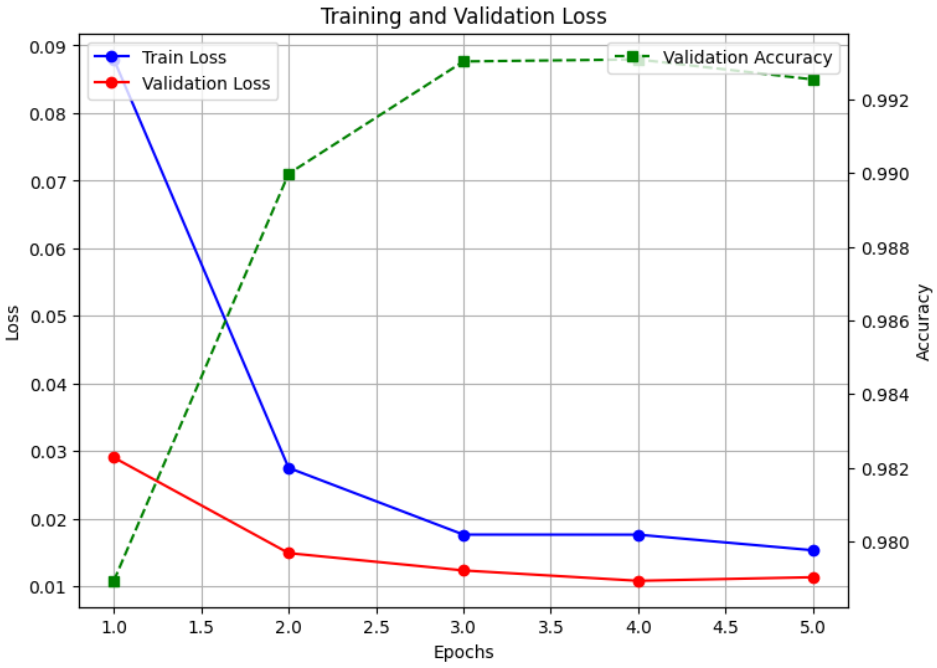


Figure 2: Graph of evaluation loss, train loss and accuracy metric at epoch 5

- **Train Loss**: 0.0153
- **Validation Loss**: 0.0114
- **Dice Score**: 0.9925

At the fifth epoch, the training loss dropped significantly to 0.0153, and the validation loss also decreased to 0.0114. The Dice score improved to 0.9925, indicating that the model was getting better at segmenting the images. The gap between the training and validation losses was small, which is a good sign that the model was not overfitting.

## 2.4 Results at Epoch 10

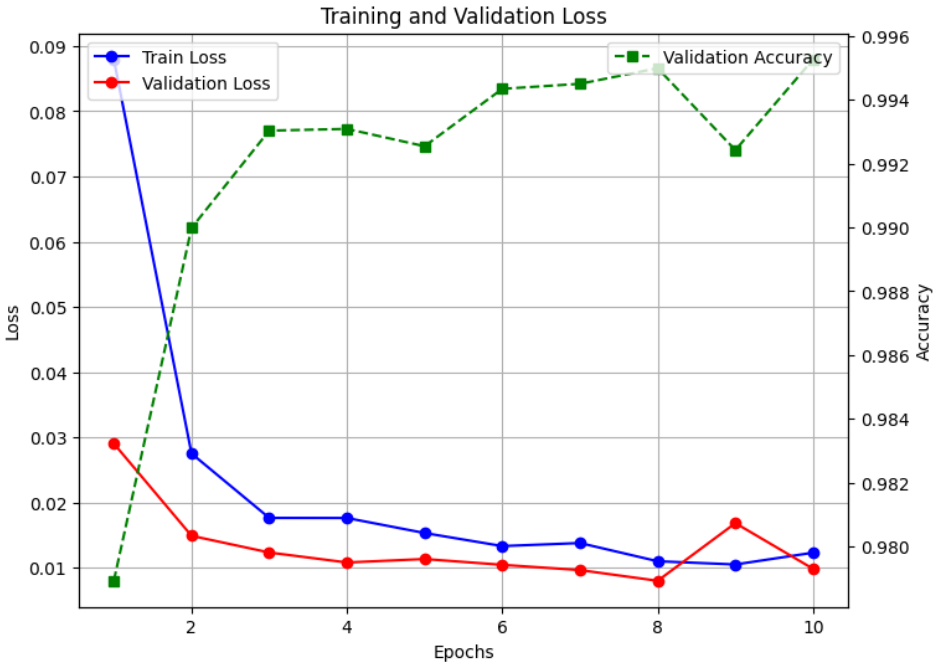


Figure 3: Graph of evaluation loss, train loss and accuracy metric at epoch 10

- **Train Loss**: 0.0123
- **Validation Loss**: 0.0099
- **Dice Score**: 0.9953

At the final epoch, the training loss further decreased to 0.0123, and the validation loss dropped to 0.0099. The Dice score reached 0.9953, which is very close to 1. This means the model was performing exceptionally well in segmenting the images. According to the final result at epoch 10, the train and validation loss started to converge from epoch 5. Additionally, the small difference between the training and validation losses suggests that the model generalized well to unseen data.

# 3 Results of image segmentation

In this report, we will analyze the 5 results of the U-Net model for image segmentation. The model was trained to distinguish between the sea and the sky in infrared images. we will evaluate the segmentation results by comparing the original images with the predicted masks.

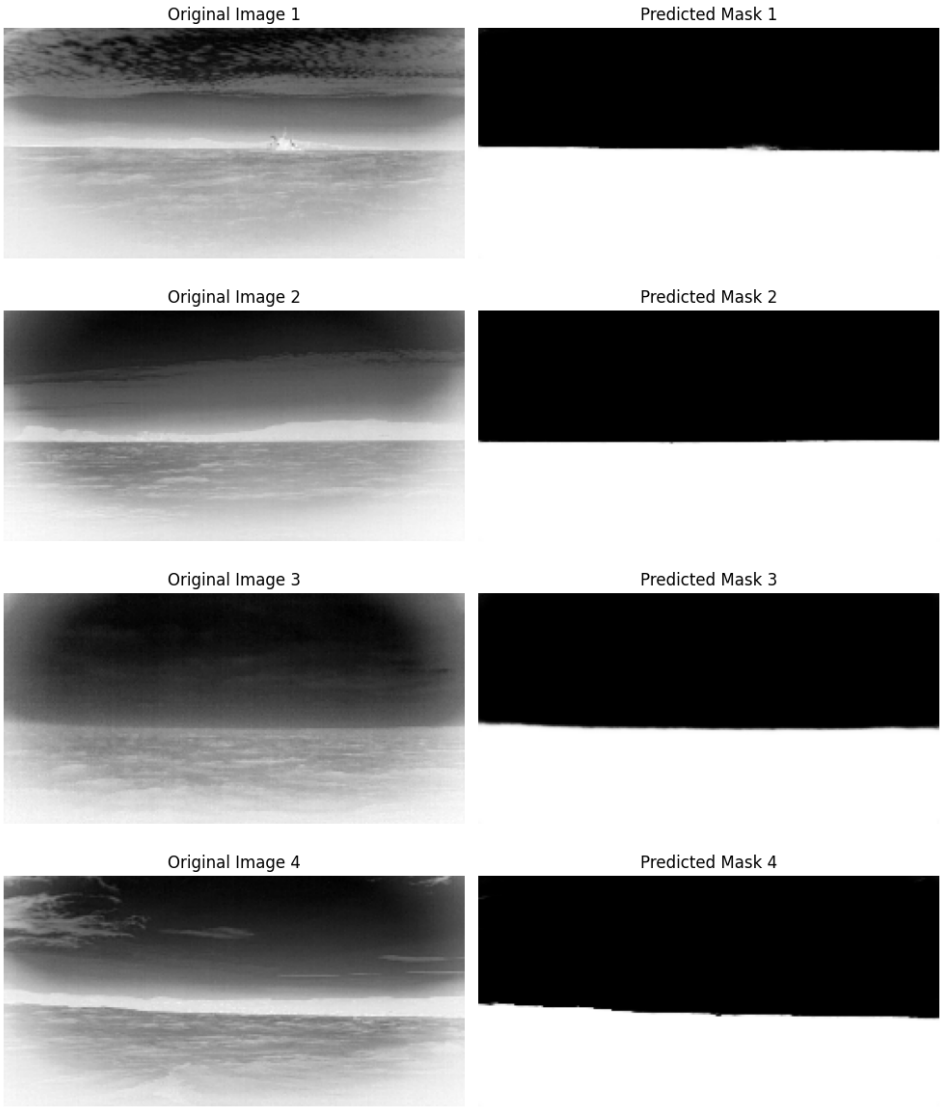


Figure 4: 5 results of image segmentation

The U-Net model demonstrated strong performance in segmenting the sea and the sky in the provided images. The predicted masks are mostly accurate, with only minor errors near the horizon and edges.

# 4 Conclusion

Overall, the U-Net model perform well in segmenting the sea and the sky. The Dice score was high, which means the model was able to separate the two areas well. However, we noticed that the accuracy was always 0.00, which doesn't make sense. we assume that there might be a mistake in how the accuracy is calculated. Even with this issue, the model improved significantly after 5 epochs.