Uncertainty Quantification for Satellite Image Segmentation

Mohamed Hasan

May 15, 2024

1 Introduction

This report details the development and evaluation of a custom Artificial Neural Network (ANN) designed for image segmentation tasks. I focus on a UNet model architecture, built from scratch using TensorFlow 2, to perform segmentation on satellite images. To enhance the robustness and reliability of the model, I implement Monte Carlo (MC) Dropout as a method for uncertainty quantification. This approach allows me to assess the confidence of the model's predictions, providing valuable insights into the reliability of segmentation results.

The experiments involve an evaluation of different activation functions, loss functions, and other hyperparameters, with a focus on their impact on uncertainty quantification. The satellite image dataset, comprising of a mix of real and synthetic images and corresponding masks, serves as a challenging real-world scenario due to factors like variable lighting conditions, diverse backgrounds, and the presence of objects (i.e. astroids).

2 Dataset

The dataset consists of satellite images and corresponding masks of size 1280×720, with each mask delineating the satellite from the background. The images are of varying resolutions and quality, reflecting the diversity of satellite imagery encountered in real-world scenarios. The dataset is divided into training and validation sets, with the training set containing 3117 images and the validation set comprising 600 images. The masks are categorized into fine and coarse masks, with 403 fine masks and 2114 coarse masks in the training set. The dataset presents a challenging segmentation task due to the presence of complex textures, varying lighting conditions, and the need to accurately delineate the satellite from the background.

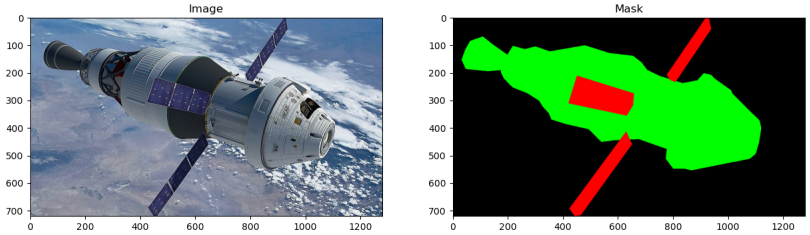

Figure 1: Sample image and mask from the dataset. Image size is 1280×720.

2.1 Data Preprocessing

The dataset undergoes preprocessing to ensure uniformity and compatibility with the model. The images are resized to a fixed resolution of 256×256 pixels and The masks are converted to binary format, with pixel values of 0 and 1 representing the background and satellite, respectively, to reduce computational complexity. The preprocessed dataset has the same split between training and validation sets as the original dataset.

The file `data_loader_unit.py` in the `utils` folder contains the function `load_and_process_files()` to load the dataset and preprocess the images and masks. After processing, the data will be saved as numpy arrays for easy access during training. The saved files are named as follows:

- **prepped_data/trainimages.npy**: Contains the preprocessed training images.
- **prepped_data/trainmasks.npy**: Contains the preprocessed training masks.
- **prepped_data/valimages.npy**: Contains the preprocessed validation images.
- **prepped_data/valmasks.npy**: Contains the preprocessed validation masks.

The function to check if the prepped files exist is `check_prepped_data()` in the `utils` folder. It is used in the scripts `train.py`, `predict.py`, and `main.py`. If the prepped files do not exist, the function is called to create them. Now, let's look at the processed dataset.

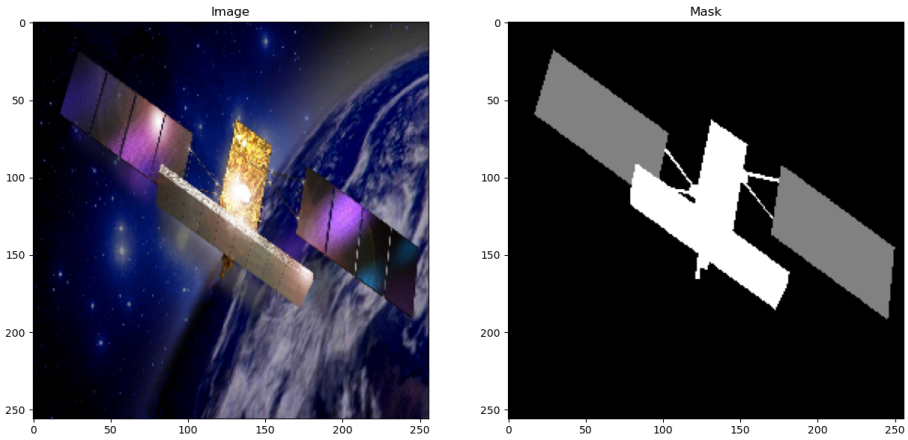

Figure 2: Sample processed image and mask from the dataset. Image size is 256×256 now and the mask is binary.

3 Model Architecture

The model architecture is based on the UNet architecture, a popular choice for image segmentation tasks due to its ability to capture both local and global features effectively. The model consists of an encoder and a decoder, with skip connections between corresponding layers to preserve spatial information. The encoder downsamples the input image to extract features, while the decoder upsamples the features to generate the final segmentation map. The model is implemented using TensorFlow 2 and Keras, further implementation details can be found in the model definition.

The model architecture is defined in the file `unet.py` in the `model` folder. The implementation is nearly identical to the standard UNet architecture. We add dropout layers after each convolutional layer in the encoder and decoder, so that we may experiment with Monte Carlo Dropout for uncertainty quantification.

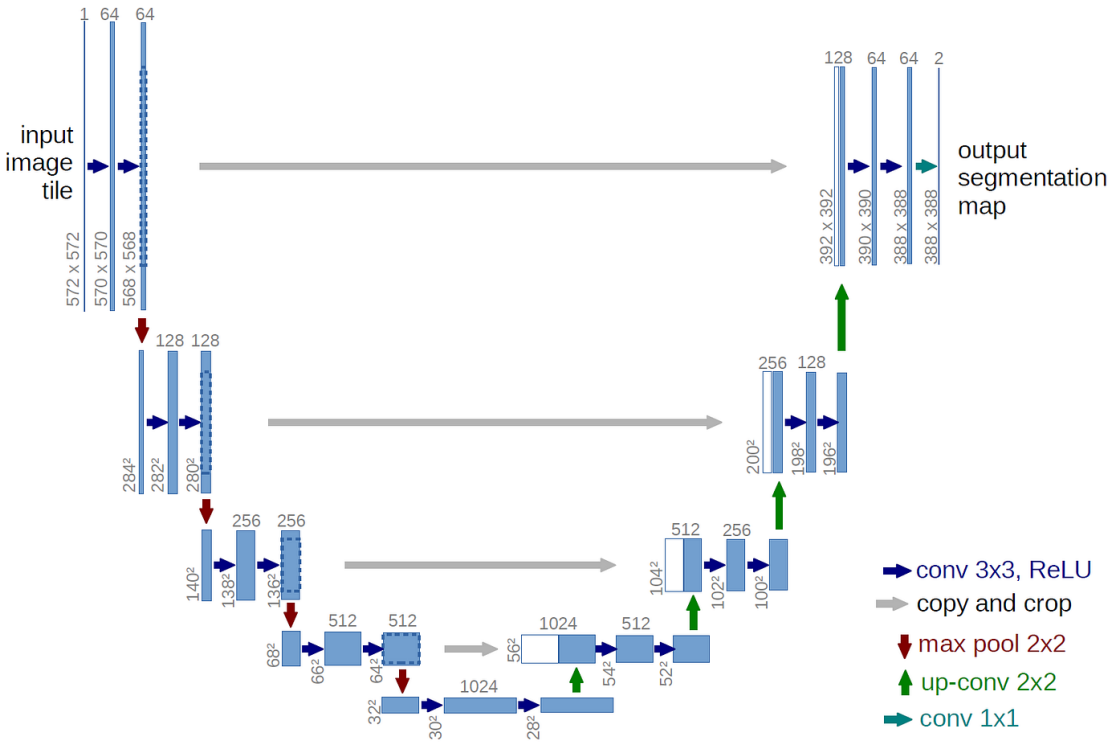

Figure 3: Original UNet model architecture. We use a similar architecture with dropout layers for uncertainty quantification.

3.1 Loss Function and Metrics

The loss function used for training the model is a combination of Binary Cross Entropy (BCE) and Dice Loss. The BCE loss is effective in handling class imbalance, while the Dice Loss directly optimizes for the Dice Coefficient, a common metric in image segmentation tasks. The Dice Coefficient is defined as

$$\text{Dice Coefficient} = \frac{2 \times \text{Intersection}}{\text{Union} + \text{Intersection}} = \frac{2 \sum_{i=1}^{N} y_i p_i}{\sum_{i=1}^{N} y_i + \sum_{i=1}^{N} p_i},$$

$$\text{Dice Loss} = 1 - \text{Dice Coefficient},$$

Where y_i is the ground truth label, p_i is the predicted probability, and N is the total number of pixels. The intersection is the number of overlapping pixels between the predicted and ground truth masks, and the union is the total number of pixels in both masks. The BCE loss is given by

$$\text{BCE} = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)],$$

A combined loss function is used to leverage the strengths of both BCE and Dice Loss. The model is trained using the Adam optimizer with a learning rate of 0.001. The metrics used for evaluation are the Dice Coefficient and Accuracy.

3.2 Training and Evaluation

The training configuration involved using a combination of Binary Cross Entropy (BCE) and Dice Loss as the loss function. The BCE loss, given by

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)],$$

where y_i is the ground truth label and p_i is the predicted probability, effectively handles the class imbalance inherent in segmentation tasks. The Dice Loss, formulated as

$$\text{Dice Loss} = 1 - \frac{2\sum_{i=1}^{N} y_i p_i}{\sum_{i=1}^{N} y_i + \sum_{i=1}^{N} p_i},$$

complements the BCE loss by directly optimizing for the Dice Coefficient, a common metric in image segmentation.

To evaluate the model's performance, I employed the Dice Coefficient and Accuracy as metrics. The dataset was divided into training and validation sets, with the training set comprising 3116 images (403 with fine masks and 2114 with coarse masks) and the validation set consisting of 600 images with fine masks.

The final model achieved a loss of 0.598, an accuracy of 0.868, and a Dice Coefficient of 0.646 on the validation set.

For uncertainty quantification, I implemented Monte Carlo (MC) Dropout. By incorporating dropout layers with a dropout rate of 0.35 during inference and performing multiple forward passes, I generated a distribution of predictions for each input image. This approach allowed me to compute the mean prediction and the standard deviation for each pixel, providing insights into the model's confidence. The mean prediction is given by

$$\mu(x) = \frac{1}{T}\sum_{t=1}^{T} \hat{y}_t(x),$$

where $\hat{y}_t(x)$ is the prediction at the t-th forward pass and T is the number of passes. The uncertainty is quantified as the standard deviation of these predictions,

$$\sigma(x) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\hat{y}_t(x) - \mu(x))^2}.$$

This method enabled me to identify areas with high uncertainty, particularly around object boundaries and regions with complex textures.

4 Uncertainty Quantification

4.1 Monte Carlo Dropout