Uncertainty Quantification for Satellite Image Segmentation

Mohamed Hasan

May 14, 2024

1 Introduction

This report discusses 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 experiment with the reliability of the model predictions, I implement Monte Carlo (MC) Dropout as a primary method for uncertainty quantification. Additionally, I explore and employ various other uncertainity quantification methods, including different activation functions, ensemble method, and an uncertainity-aware loss function, to further assess and quantify uncertainty. This approach allows me to evaluate the confidence of the model's predictions, providing valuable insights into the reliability of segmentation results.

2 Dataset

The dataset consists of 3117 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 2517 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 validation set contains all fine masks. 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.

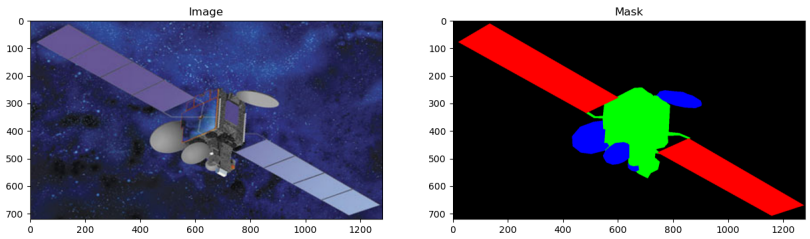

Figure 1: Sample image and mask from the dataset. Image size is 1280×720.

2.1 Data Preprocessing

The 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, since my computer has limited memory. 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.

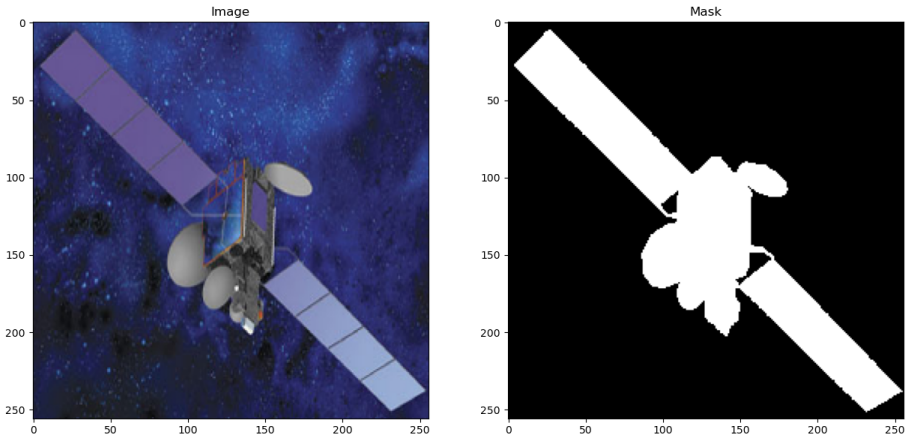

Figure 2: Sample processed image and mask from the dataset. Image size is 256×256 now and the mask is binary.

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. I add dropout layers after each convolutional layer in the encoder and decoder, so that I can experiment with Monte Carlo Dropout for uncertainty quantification.

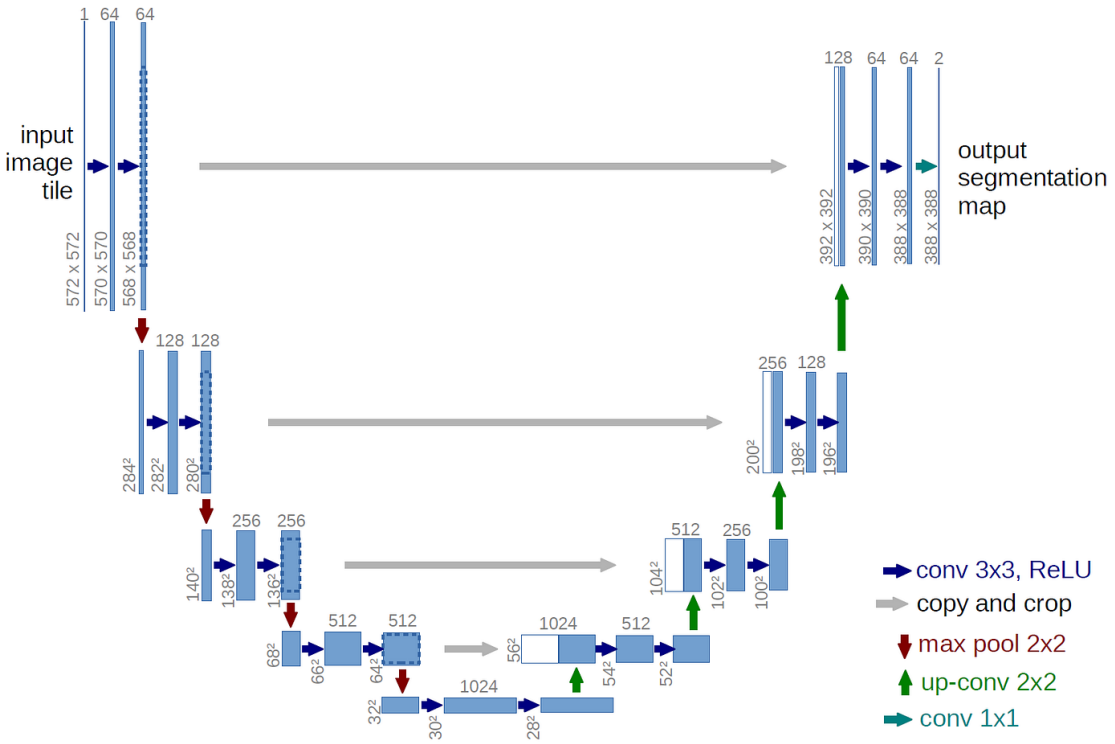

Figure 3: Original UNet model architecture. I use the same architecture, but with dropout layers for uncertainty quantification.

3.1 Loss Function, Metrics, and Hyperparameters

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.

$$\text{Combined Loss} = \text{BCE} + \text{Dice Loss}.$$

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.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

The hyperparameters and model configurations are defined in the file `config.py` in the `model` folder. The file contains the following hyperparameters:

- **BATCH_SIZE** = 1: The batch size used for training the model.
- **EPOCHS** = 5: The number of epochs for training.
- **DROPOUT_RATE** = 0.35: The dropout rate used in the model.

3.2 Evaluation

The final model achieved:

- Loss: 0.399
- Accuracy: 0.940
- Dice Coefficient: 0.803

The model shows good performance on the validation set, with high accuracy and Dice Coefficient, despite the challenging nature of the dataset and the low number of epochs. Let's visualize the model's predictions on a sample image from the validation set.

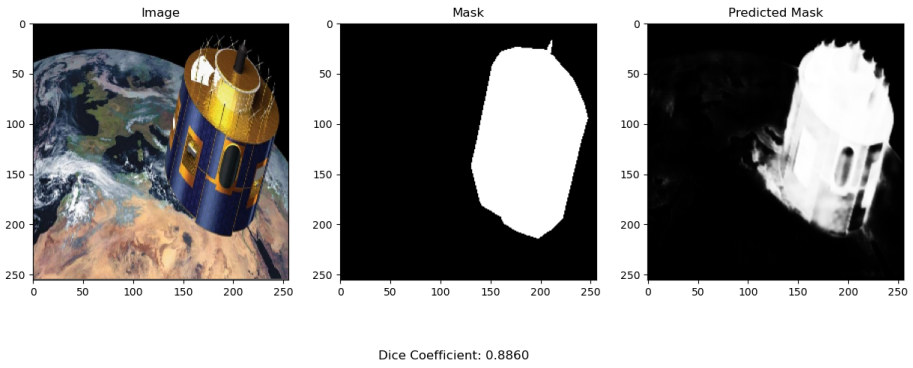

Figure 4: Sample prediction from the validation set, dice coefficient is for the mean prediction.

The model successfully segments the satellite from the background, capturing the main features of the satellite with a dice coefficient of 0.8860. Notice however that the model captures the details of the satellite which is not included in the masks. This is due to the model's ability to learn from the training data and generalize.

4 Uncertainty Quantification

4.1 Monte Carlo (MC) Dropout

For uncertainty quantification, I implemented the Monte Carlo (MC) Dropout technique. Dropout is a regularization method used during the training of neural networks to prevent overfitting. It works by randomly "dropping out" a subset of neurons during each forward pass, effectively creating an ensemble of different network architectures. Typically, dropout is disabled during inference to use the full capacity of the

trained model. However, in MC Dropout, dropout is kept active during inference to introduce stochasticity into the model's predictions.

By incorporating dropout layers with a dropout rate of 0.35 during inference and performing multiple forward passes (in this case, 10 passes), I generated a distribution of predictions for each input image. This approach allowed me to capture the variability in the model's predictions due to the stochastic dropout, which can be interpreted as a measure of uncertainty.

The mean prediction for each pixel is calculated as follows:

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

where $\hat{y}_t(x)$ is the prediction at the t-th forward pass and T is the number of passes. The mean prediction provides an aggregate view of the model's output across multiple 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 standard deviation reflects the model's confidence in its predictions. A higher standard deviation indicates greater uncertainty, whereas a lower standard deviation suggests higher confidence.

MC Dropout is particularly useful for identifying regions where the model is less certain about its predictions. In the context of image segmentation, uncertainty is often higher around the edges of objects and in regions with complex textures. This can be seen in Figure 5, where we can see the mean prediction and the standard deviation uncertainity generated by the MC Dropout technique for five of the validation images.

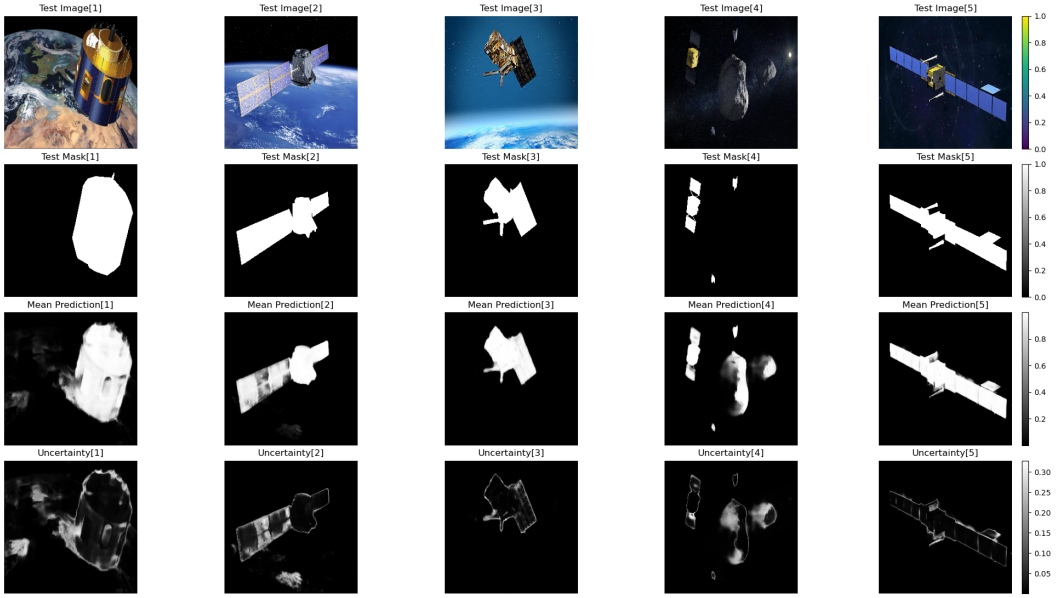

Figure 5: We can see the test image, its mask, the mean prediction, and the uncertainty.

- **test image[1]**: The satellite's body is well captured, but the model shows high uncertainty around the edges and finer details. We can also see that the uncertainity makes sense, espeically near the bottom lower edge of the satellite where there is a body of water. The water and the satellite are both blue in this case, so the model is less certain about the prediction in that region.

- **test image[2]**: The satellite and its solar panels are identified, but the prediction is somewhat spread, with significant uncertainty around the solar panels. This is most likely because the solar panels are blue and the background is the blue oceans which makes it harder for the model to distinguish between the two.
- **test image[3]**: The satellite is identified quite well, with uncertainity only around the edges.
- **test image[4]**: The three satellites are identifies, but the model also identifes two astroids as satellites. This is most likely due to the dataset not having many examples of astroids and the astroids having similar features (most likely textures) to the satellites.
- **test image[5]**: The satellite is identified quite well, with uncertainity only around the edges.

The results show that the mean predictions generally capture the satellite structures well. However, when it comes to the edges and the finer details, uncertainty is observed. Notably, significant uncertainty for similar colors or textures - for example, the overlap of the water and satellite in the first test image, or the solar panels against the blue ocean in the second one. And when the model mistakes asteroids for satellites in the fourth test image, the uncertainty increases even more. This shows where the model's weak spots are, most likely due to a lack of training examples. Overall, MC Dropout does a great job of pointing out the low confidence areas. This gives us a clear direction for improving the model's predictions - by focusing on these high-uncertainty areas.

Potential improvements could include increasing the number of training examples, augmenting the dataset with more diverse images, or fine-tuning the model architecture to better capture the distinguishing features of the satellite. However, since we are focusing on uncertainity quantification in this report, I would suggest combining uncertainity quantification methods. For example, I could use Bayesian Neural Networks to capture the model's uncertainty in the weights, and then use MC Dropout to capture the uncertainty in the predictions. This would give us a more comprehensive view of the model's uncertainty and help us make more informed decisions. I impelmented a Bayesian UNet model with Dropout, but could not train it due to computational limitations.

4.2 Different Activation Functions with MC Dropout

Let's now test a few activation functions to see how they effect uncertainity quantification. Each activation function has its unique characteristics and impacts the model's convergence and generalization in different ways. By evaluating multiple activation functions, I want to identify which ones achieve low uncertainty and high accuracy in the segmentation task.

Activation Function	Loss	Accuracy	Dice Coefficient
ReLU	0.399	0.94	0.803
ELU	0.499	0.914	0.773
Swish	0.43	0.934	0.762
GELU	0.436	0.932	0.775
Leaky ReLU	0.418	0.933	0.772

Table 1: Performance Metrics for Different Activation Functions

From the results in Table 1, we can see that the ReLU activation function performs the best in terms of accuracy and Dice coefficient. Luckily, I used ReLU as the activation function in the model for the MC Dropout uncertainity quantification, as it is the best performing activation function in this case.

Figure 6 shows that ReLU and Leaky ReLU provide the best performance for the predictions. Both functions result in mean predictions with high contrast and clear definition, while also exhibiting lower uncertainty in the prediction maps. This indicates more reliable and confident predictions. In contrast, ELU, Swish, and GELU show higher uncertainty and more blurring in the mean predictions, especially when considering various diverse backgrounds, showing they are less effective for this specific task.

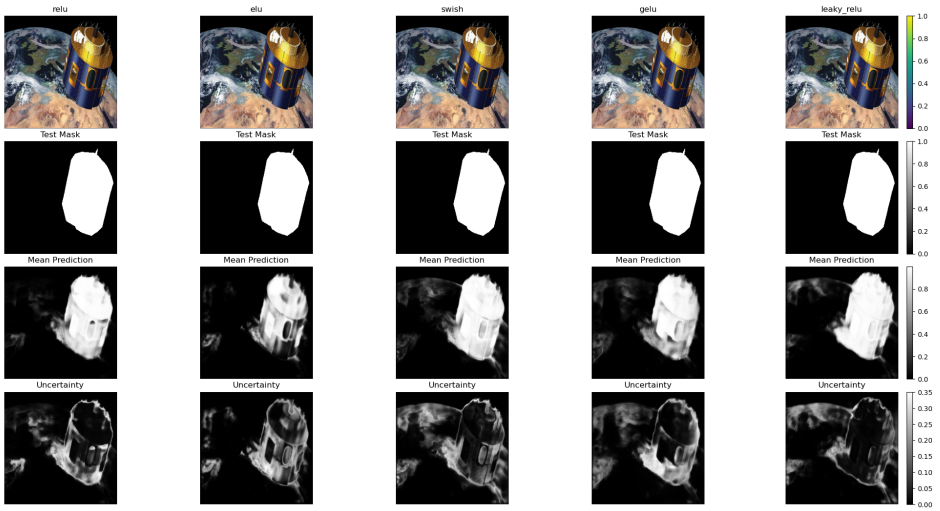

Figure 6: Sample predictions using different activation functions.

4.3 Ensemble of 5 Models with MC Dropout

To further improve the model's uncertainty quantification, I experimented with ensemble methods. In this approach, I train multiple models using the same architecture but with different initializations and stochastic variations during training. Each model in the ensemble is trained on the entire dataset, becayse of the limited amount of data available. By aggregating the predictions from these individual models, we can capture the variability in the model's sum predictions, providing a more comprehensive measure of uncertainty. The reason for using ensemble methods is that different models may still learn different aspects of the data due to their unique training paths, and their combined predictions can mitigate individual model biases and errors. I expect that this approach will improve the model's overall performance and provide more accurate and reliable uncertainty estimates, particularly in regions where individual models might be less confident.

Looking at figure 7, we can see that the combined predictions from the five models did not perform as well as anticipated. the dice coefficient is 0.8679, which is lower than the dice coefficient of the single model from figure 4, at 0.8860. This is most likely because all the models were trained on the same dataset without any augmentation, leading to similar learned patterns and limited diversity in their predictions. This suggests that for ensemble methods to be more effective, incorporating data augmentation or using different subsets of the data for training each model may be necessary to increase the diversity of the models and improve predictions.

4.4 Uncertainty-Aware Loss Function

For this experiment, I will incorporate uncertainty estimates directly into the loss function, allowing the model to pay more attention to regions where it is less confident. I expect this approach to be useful, especially in tasks like image segmentation, where accurate delineation of objects is crucial, and regions of high uncertainty often correspond to areas that are challenging for the model.

By penalizing these uncertain predictions more heavily during training, hopefully the model can learn more robust and reliable features. The uncertainty-aware loss is calculated by multiplying the base loss with a factor that incorporates the uncertainty, weighted by a hyperparameter λ.

The formula for the uncertainty-aware loss is given by:

$$\text{Uncertainty-Aware Loss} = \text{Mean}\left((\text{BCE} + \text{Dice Loss}) \times (1 + \lambda \times \text{Uncertainty})\right)$$

where BCE is the Binary Crossentropy loss and Uncertainty is the standard deviation of the predictions. λ is a hyperparameter that controls the influence of the uncertainty term on the overall loss.

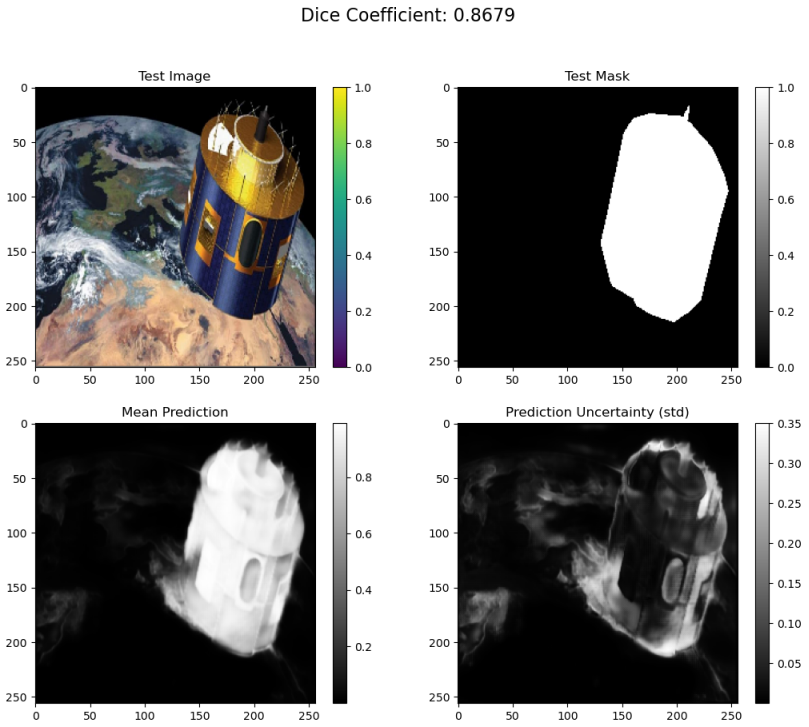

Figure 7: Sample predictions from the ensemble of 5 models, trained on the same dataset.

From figure 8, we can see that the addition of the uncertainty-aware loss function using differnt values of λ did not improve the model's performance. The dice coefficient of the best performing model ($\lambda = 0.1$) is 0.798, which is slightly lower than the dice coefficient of the base model at 0.803. Thus I can conclude that the uncertainty-aware loss function did not provide a significant improvement in this case. Perhaps further tuning of the hyperparameters or exploring different loss formulations could yield better results.

5 Conclusion

In conclusion, combining Monte Carlo (MC) Dropout with various experimental approaches for uncertainity quantification, provided significant insights into our model's uncertainty. Collectively, these methods allow us to measure and understand the confidence of our model's predictions effectively. By analyzing the variability and consistency across different models and configurations, we gain a deeper understanding of the model's strengths and areas for improvement. This approach to uncertainty quantification is crucial for enhancing the reliability of models that depend on accurate predictions.

It is a shame that the bayesian UNet model could not be trained due to my lack of memory. This model would have provided more comprehensive and detailed results for the model's uncertainty by capturing the uncertainty in the weights. This would have allowed us to make more informed decisions. It is very interesting and I will definitely explore this further in the future.

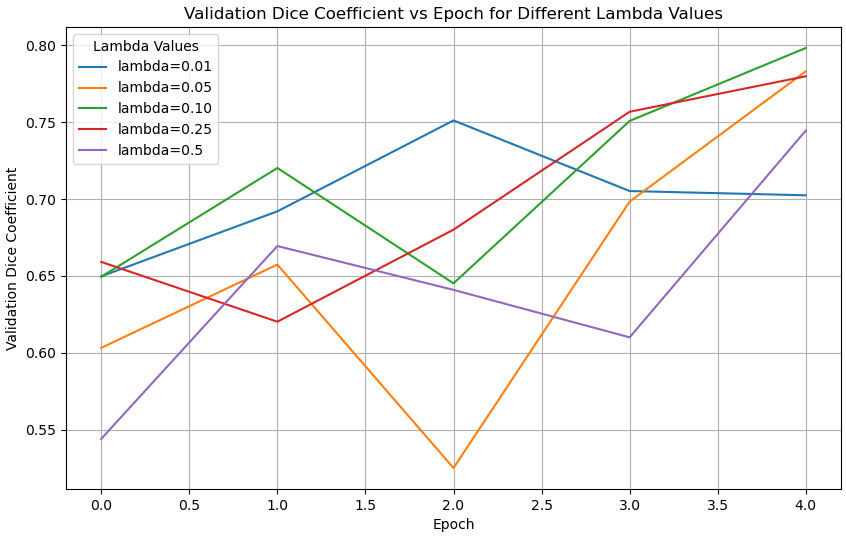

Figure 8: Validation dice coefficient for different values of the hyperparameter λ.