Nuclei Segmentation in Microscope Cell Images Using U-Net

Gheorghe Andrei-Bogdan

June 18, 2025

1 Introduction

In this project, I implement the U-Net neural network architecture within an application designed to detect and segment cell nuclei in microscopic images. The U-Net model is well-suited for image segmentation tasks due to its encoder-decoder structure, allowing precise localization of nuclei boundaries in complex cellular images.

This project uses datasets from the Kaggle platform for nuclei segmentation in microscopic cell images. The main data sources used are:

https://www.kaggle.com/datasets/sinjoysaha/nucleiimagesmasksfromdsb2018

https://www.kaggle.com/datasets/gangadhar/nuclei-segmentation-in-microscope-cell-images

2 Challenges and Milestones

- One of the first issues encountered was related to the mask dimensions in the dataset, which had an extra dimension: `[4, 1, 1, 256, 256]` instead of `[4, 1, 256, 256]`. In the `data_loader.py` file, when applying the `ToTensor()` transform on masks, the result had an additional dimension because the mask was already a 2D NumPy array. To fix this, I added a `.squeeze(0)` operation when assigning the mask.
- In the dataset, masks for each nucleus were separate, so I modified the code to merge all individual masks into a single one, which I then used for training and calculating prediction accuracy.
- The initial training was very slow and the loss was high (up to approximately 9.95). The predicted images at this stage were completely black, which I later discovered it was due to a problem with the masks. They were being interpreted as having values in the `[0,1]` range instead of `[0,255]`, which disrupted the learning process and caused the loss to plateau around 9 to 10. Also, to speed up the training, I used Google Colab with a T4 GPU.
- Although I saved the model to the `unet.pth` file, I did not properly load it before training, which caused the learning to almost restart from scratch every time.
- After correcting all those problems and properly loading the trained model, I achieved a significant decrease in loss on a subset of 50 images, with the following values: 556.2468, 0.6246, 0.2318, 0.1984, 0.1871, 0.1650, 0.1553, 0.1460, 0.1294, 0.1189.
- I continued training on the entire dataset using Colab, obtaining the following loss values: 0.9943, 0.7835, 0.7587, 0.7490, 0.7393, 0.7369, 0.7310, 0.7193, 0.7095, 0.7028.
- After 30 epochs, the loss decreased to 0.6837. Initially, the output threshold was set to a very small value (`3e-08`), successfully obtaining decent but not perfect mask predictions. Later, it was modified to one tenth of its maximum value, which yielded the best mask results.
- Finally, after reaching a loss of only 0.6775, I implemented a function that compares the predicted mask to the ground truth mask, achieving an accuracy of approximately `91.5%`.

3 Model and Implementation Details

The core of this project relies on the U-Net architecture, a convolutional neural network originally designed for biomedical image segmentation. Its encoder-decoder structure allows it accurately segment cell nuclei.

3.1 U-Net Architecture

The U-Net implemented in this project follows this structure:

- The encoder consists of four levels, each containing two convolutional layers followed by a ReLU activation and a max pooling operation. This part progressively reduces the spatial resolution while increasing the number of feature channels.
- The bottleneck is a convolutional block with 1024 feature channels, which captures the deepest representation of the input.
- The decoder upsamples the feature maps using transpose convolutions and concatenates them with corresponding feature maps from the encoder.
- The final layer applies a 1x1 convolution to reduce the number of output channels to one, producing the segmentation mask as a single-channel image.

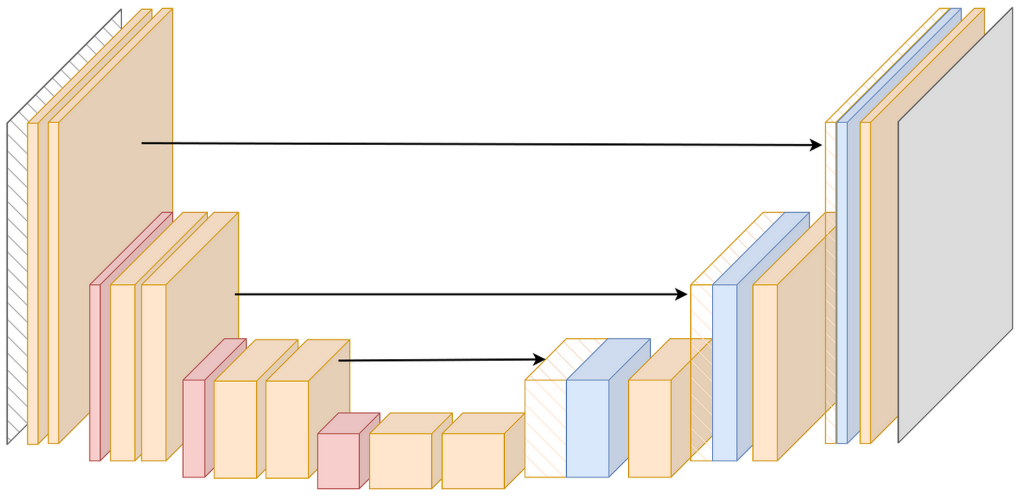


Figure 1: U-Net Architecture.

3.2 Data Loading and Preprocessing

Data loading is defined in the `data_loader.py` script. Each training sample consists of:

- An RGB image resized to `256x256`.
- A binary mask, constructed by combining all individual nucleus masks into one.

The images and masks are converted to PyTorch tensors. Basic preprocessing is performed using `ToTensor()` from `Torchvision`.

3.3 Training

The training loop is defined in the `train.py` script. The main steps are:

- A U-Net model is instantiated and moved to the available device (GPU or CPU).
- If a saved model file (`unet.pth`) exists, it is loaded to continue training from the last state.
- The model is trained for ten (or more) epochs using a batch size of 4 and the Adam optimizer.
- The loss function used is `BCEWithLogitsLoss`, suitable for binary classification tasks such as segmentation.
- After each epoch, the average training loss is shown, and at the end, the model is saved.

The output of the model is a tensor of shape `[B, 1, 256, 256]` with raw scores (logits). During inference, these scores are passed through a sigmoid function and dynamically thresholded to obtain binary masks.

3.4 Evaluation

To evaluate the model, a custom function compares predicted masks to the ground truth using pixel-wise accuracy. For user interaction, a graphical interface was developed using `Streamlit`, allowing users to upload an image and immediately see the predicted mask displayed on screen.

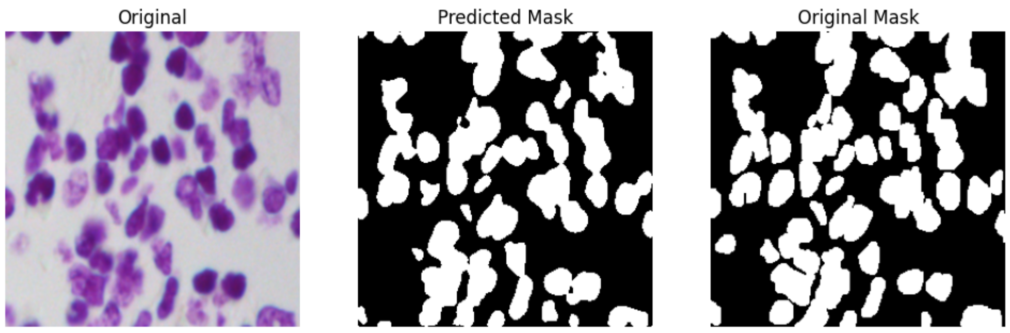
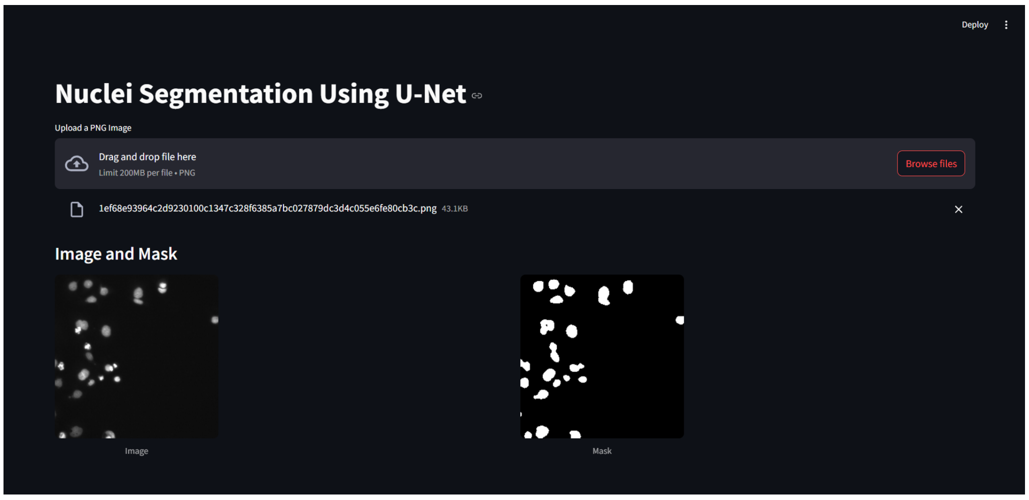


Figure 2: Mask comparison.



Nuclei Segmentation Using U-Net

Upload a PNG Image

Drag and drop file here

Limit 200MB per file • PNG

1ef68e93964c2d9230100c1347c328f6385a7bc027879dc3d4c055e6fe80cb3c.png 43.1KB

Image and Mask

Figure 3: User Interface.

References and Useful Links

- https://github.com/kamalkraj/DATA-SCIENCE-BOWL-2018
- https://www.kaggle.com/code/dimakyn/pytorch-unet
- https://www.youtube.com/watch?v=cAkMcPfY_Ns
- https://www.youtube.com/watch?v=aircAruvnKk