D7043E Project
U-Net Based Biomedical Image Segmentation
Final Report

Group Alpha: Umuthan Ercan, Amirhossein Nayebiastaneh

November 5, 2023

1 Introduction

Our project, centred around the U-Net-based biomedical image segmentation, has been marked by a series of challenges that required a careful approach in design, code implementation, and testing. This post-implementation report provides an overview of our project journey, shedding light on the processes, obstacles, materials, libraries, and coding methods employed during the development phase. In recent years, the realm of deep learning has witnessed substantial progress in biomedical image analysis. Our project was inspired by the influential article "U-Net: Convolutional Networks for Biomedical Image Segmentation" authored by Ronneberger, Fischer, and Brox[1]. This seminal work introduced an innovative architecture and training strategy tailor-made for biomedical image segmentation, and in this report, we will provide further information about the steps.

2 Development Process

The development process of our U-Net-based biomedical image segmentation project unfolded in several phases:

2.1 Design Phase

During the initial design phase, we defined the project's objectives and scope. We aimed to implement the U-Net architecture for precise image segmentation, particularly in the biomedical domain. This involved planning the model's architecture, considering aspects like the number of convolutional layers, feature channels, and the balance between precision and computational efficiency.

2.2 Implementation Phase

The implementation phase was characterized by the actual coding of the U-Net model using the PyTorch framework. This phase involved designing the architecture with flexible input and output channel configurations, ensuring adaptability to various biomedical imaging tasks. We carefully constructed the convolution blocks, fine-tuning activation functions and layer configurations to achieve the desired level of segmentation precision. Notably, we focused on the segmentation of Jurkat Cell images, a dataset we chose for its compatibility with our project and availability in the field of biomedical image analysis[2]. Defining the forward pass, which allows the model to process input images and preprocess segmentation masks, was a pivotal part of this phase. The connectivity between the contracting and expansive parts was established, ensuring the model efficiently captures context and fine details for better segmentation.

2.3 Challenges Faced

Throughout the development process, several challenges were encountered. These challenges included:

- **Implementing the U-Net Architecture**: The project encountered challenges while implementing the U-Net architecture, skip connections, and concatenations using the PyTorch framework. These difficulties included ensuring proper integration and functioning of these complex components within the model. However, through diligent problem-solving and iterative development, the project's team successfully overcame these challenges, leading to the effective implementation and functioning of the U-Net architecture with skip connections and concatenations in PyTorch.
- **Resource Efficiency**: Optimizing the model for computational efficiency was crucial. Balancing model complexity and computational resources, particularly for high-resolution inputs, posed a challenge in ensuring the model's suitability for practical applications, including Jurkat Cell image analysis.
- **Evaluation Metrics**: Selecting appropriate evaluation metrics for biomedical segmentation tasks, which require precise pixel-level localization and spatial relationship understanding, was a challenging task. A thorough evaluation is essential in this domain, mainly while working on the Jurkat Cell image segmentation.

2.4 Materials, Libraries, and Technical Environment

Our project made use of various tools and libraries, including:

- **Python 3.9.13**: We employed Python as the primary programming language for developing and implementing the UNet-based biomedical image segmentation model.
- **PyTorch**: The U-Net model was implemented using PyTorch, known for its flexibility and deep learning capabilities.
- **Other Libraries**: Various Python libraries, including NumPy, Matplotlib, PIL, os and scikit-learn, were used for data handling, visualization, and model evaluation.
- **Visual Studio Code (VSCode)**: We utilized Visual Studio Code as our integrated development environment (IDE) for coding, debugging, and project management.
- **Git**: Version control was managed through Git, enabling collaborative development and tracking changes throughout the project.

This technical environment played a crucial role in the successful development of our U-Net based biomedical image segmentation project.

3 Experimental Results

In this section, we present the experimental results of our U-Net based biomedical image segmentation project. We evaluate the model's performance using various metrics and provide insights into the segmentation results for the Jurkat Cell dataset.

3.1 Model Performance Metrics

In our evaluation, we consider several key performance metrics to assess the effectiveness of the U-Net model in biomedical image segmentation. These metrics include accuracy and F1-score.

3.1.1 Accuracy

Accuracy measures the overall correctness of our model's predictions. It quantifies the ratio of correctly segmented pixels to the total number of pixels in the image. Achieving high accuracy is essential in biomedical applications, where precision is critical.

3.1.2 F1-Score

The F1-score is a crucial metric in biomedical image segmentation as it balances precision and recall. It is particularly relevant when dealing with imbalanced datasets or when precision and recall have different importance. F1-score provides a single metric that combines both precision and recall, making it a useful measure for evaluating segmentation models.
The F1-score is defined as the harmonic mean of precision and recall, expressed by the following equation:

$$F1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Where:

- Precision measures the ratio of true positive predictions to the total number of positive predictions made by the model.

- Recall quantifies the ratio of true positive predictions to the total number of actual positive instances in the dataset.

The balance between precision and recall is crucial since higher precision minimizes false positives, while higher recall minimizes false negatives. Our model aims to strike a balance between these two aspects.

3.1.3 Intersection over Union (IoU)

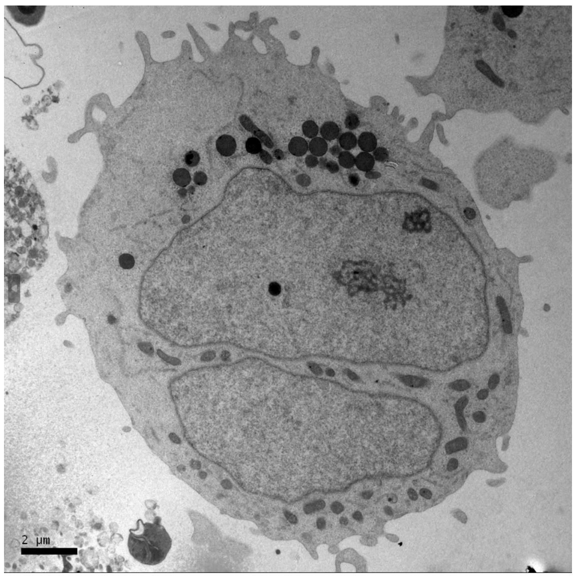
The Intersection over Union (IoU), also known as the Jaccard Index, is a crucial metric in our project for evaluating the accuracy of our U-Net model's segmentation results. It quantifies the degree of overlap between the predicted and actual regions in biomedical image segmentation. By measuring the intersection (correctly classified pixels) relative to the union (total area), IoU provides a valuable assessment of segmentation quality. A higher IoU score indicates better alignment between the predicted and ground truth masks, signifying the model's ability to accurately segment biomedical images.

3.2 Segmentation Results

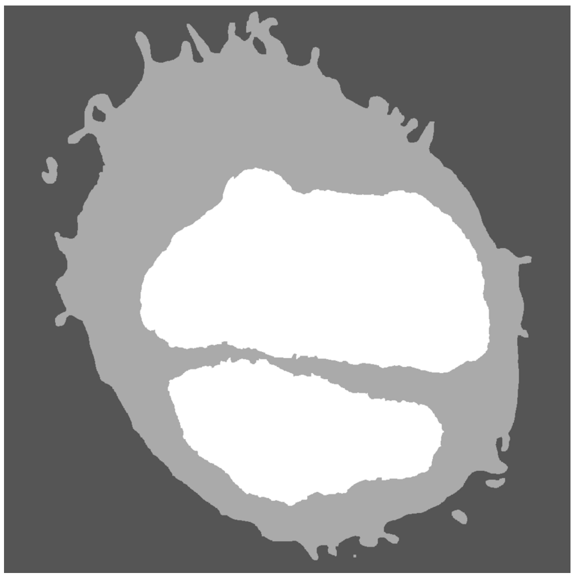
In this subsection, we present the segmentation results for the Jurkat Cell dataset, which is a crucial component of our project.

3.2.1 Jurkat Cell Dataset

The Jurkat Cell dataset was selected as our primary dataset for evaluation due to its relevance in the field of biomedical image analysis[2]. It contains a collection of images representing Jurkat Cells, a type of human T lymphocyte. The dataset features a wide variety of cell shapes, sizes, complexities, and their masks, making it a challenging testbed for our U-Net model.
Our segmentation results on the Jurkat Cell dataset reveal the model's performance in accurately identifying and segmenting individual cells within complex images. We will provide a detailed analysis of the model's ability to comprehend various cell shapes, and other complexities present in the dataset.



(a) *Example of Jurkat cell image [2]*



(b) *Example of Jurkat cell mask [2]*

3.3 Performance Discussion

In the performance discussion, we will delve into the insights gained from the experimental results. We will analyze the model's performance metrics, discuss its strengths and areas for improvement, and provide recommendations for further refinement. The discussion will also address the challenges encountered during the project and how they influenced the model's performance.

- *Results for the Jurkat Cell Dataset*

The Jurkat Cell dataset is widely used in biomedical image segmentation tasks. Our U-Net model was tested on this dataset to evaluate its performance. The dataset consists of images depicting Jurkat cells, and our task was to segment the cells accurately.
The model demonstrated remarkable results when applied to the Jurkat Cell dataset, providing precise segmentation. The following performance metrics were achieved:

$$\text{Average Accuracy: } 0.8599$$
$$\text{F1-Score: } 0.7599$$
$$\text{Jaccard Index (IoU): } 0.8599$$

Our model successfully implemented precise segmentation, and the results were consistent with the high expectations set in the design phase. The model effectively learned to distinguish between different regions within the images and produce accurate segmentation masks.

4 Network Architecture

This section delves into the architectural details of the U-Net model we implemented for biomedical image segmentation. We provide an overview of the U-Net architecture and its individual components, including the contracting part, connecting part, expansive part, and final part. Additionally, we explore the up-sampling technique used in U-Net. For more information, you can refer to our Phase 1 Report.

4.1 U-Net Architecture Overview

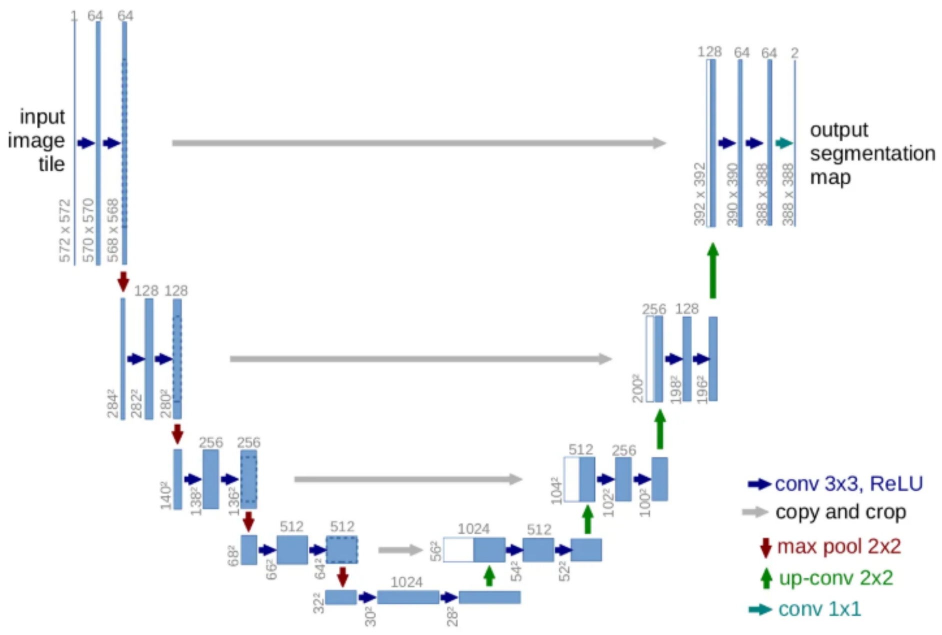


Figure 2: *U-Net Architecture [1]*

The U-Net architecture is characterized by a U-shaped design and consists of two main parts: the contracting path and the expansive path. This unique design allows the model to capture context information and make precise predictions for each pixel in the image efficiently.

4.2 Contracting Part

The contracting part serves as the initial stage of the U-Net architecture, responsible for reducing the spatial dimensions of the input image while capturing high-level features. It comprises four down-sampling steps, each consisting of a series of operations, including 3x3 convolutions, ReLU activations, and 2x2 max-pooling. In the initial setup, the padding was initially set to 0, which resulted in a valid convolution. This configuration caused pixel loss during the convolution process. To mitigate this issue, the project team collectively decided to set the padding to 1. This adjustment effectively prevented the loss of pixels and introduced a specific form of convolution known as "same" convolution.
In each down-sampling step, the number of feature channels doubles, progressively increasing from 64 to 512. This expansion in feature channels helps the model understand the image's context.

4.3 Connecting Part

The connecting part acts as a bridge between the contracting and expansive parts. It contains a single step without down-sampling and includes operations such as 3x3 convolutions and ReLU activations. The number of feature channels is also doubled from 512 to 1024, ensuring the smooth transition of information from the contracting to the expansive part.

4.4 Expansive Part

The expansive part of the U-Net architecture is responsible for gradually increasing the spatial dimensions of the feature maps and restoring finer details. It consists of four up-sampling steps, each of which includes operations such as 2x2 up-convolution, concatenation with the corresponding cropped feature map from the contracting path, 3x3 convolutions, and ReLU activations.

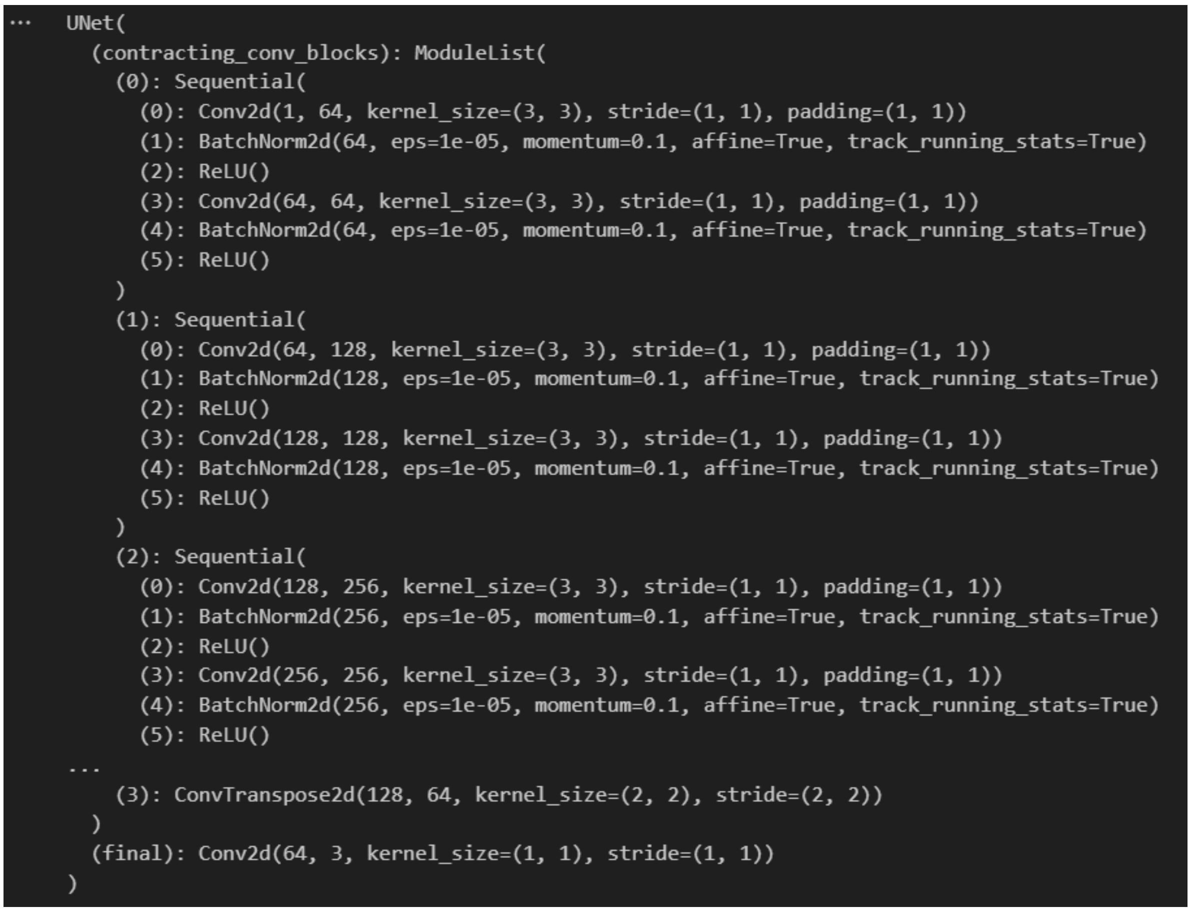
Significantly, each up-sampling layer reduces the number of feature channels by half, helping maintain a balanced architecture. The feature channels progressively decrease from 512 to 64, ensuring that the model retains detailed information while generating the segmentation masks. As we adopted a padding value of 1, it ensured that the sizes remained consistent throughout the process. Consequently, there was no requirement for the utilization of the cropping feature.

4.5 Final Part

The final part of the U-Net architecture consists of a 1x1 convolution operation. This layer maps each 64-component feature map to the desired number of classes; it is noteworthy that the number of channels in the network architecture is often set to 2 for binary segmentation tasks. However, given the nature of our task, which involves segmenting images into three distinct classes representing the background (dark grey), cell body (light grey), and cell kernel (white), our model utilizes three channels to effectively capture these variations.

4.6 Up-Sampling

Up-sampling in U-Net is achieved through a technique known as Transposed Convolution or "up-convolution." This operation helps restore spatial dimensions lost during the contracting phase and is crucial for generating detailed segmentation masks. Transposed Convolution effectively up-scales feature maps.



```
···   UNet(
        (contracting_conv_blocks): ModuleList(
          (0): Sequential(
            (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): ReLU()
            (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (5): ReLU()
          )
          (1): Sequential(
            (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): ReLU()
            (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (5): ReLU()
          )
          (2): Sequential(
            (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): ReLU()
            (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (5): ReLU()
      ...
          (3): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
        )
        (final): Conv2d(64, 3, kernel_size=(1, 1), stride=(1, 1))
      )
```

Figure 3: *Demonstrating the U-Net architecture implementation*

For a more in-depth understanding of Transposed Convolution, you can refer to our initial Phase 1 Report.

5 Training Process

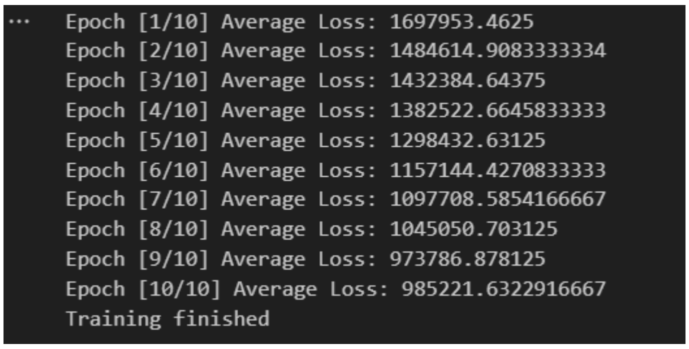
During the training process, we divided the dataset into two subsets: the training set and the validation set. The purpose of this division was to evaluate the model's performance while it was being trained. Specifically, we allocated a portion of the dataset for validation, typically 20% of the entire dataset. The remaining 80% was used for training the model. This approach allowed us to train the model on one subset and validate its performance on another, ensuring that the model generalizes well to unseen data. The split was performed randomly to ensure an unbiased distribution of data between the two sets.

5.1 Batch Normalization

During the model training process, we applied batch normalization in the convolutional blocks of the U-Net architecture. Batch normalization is a critical technique that helps stabilize and accelerate training. It normalizes the activations within each mini-batch, reducing internal covariate shift. This ensures that the model trains more efficiently, allowing for faster convergence and better generalization.
The application of batch normalization in our architecture significantly contributed to the training process's stability and the model's ability to achieve precise segmentation results. By reducing internal covariate shift, batch normalization played a vital role in ensuring that the network learned effectively from the data.

5.2 Hyperparameters

During the training process, we carefully chose specific hyperparameters to enhance the model's performance. These hyperparameters, such as a batch size of 3, a learning rate of 0.001, and 100 training epochs, were thoughtfully adjusted. We determined these values and configurations through systematic testing during the training phase to ensure the model converged effectively and didn't overfit.



```
...   Epoch [1/10] Average Loss: 1697953.4625
      Epoch [2/10] Average Loss: 1484614.9083333334
      Epoch [3/10] Average Loss: 1432384.64375
      Epoch [4/10] Average Loss: 1382522.6645833333
      Epoch [5/10] Average Loss: 1298432.63125
      Epoch [6/10] Average Loss: 1157144.4270833333
      Epoch [7/10] Average Loss: 1097708.5854166667
      Epoch [8/10] Average Loss: 1045050.703125
      Epoch [9/10] Average Loss: 973786.878125
      Epoch [10/10] Average Loss: 985221.6322916667
      Training finished
```

Figure 4: *Illustrating the training of the U-Net model, which was originally trained over 100 epochs*

5.2.1 Loss Function

During our model's training process, we utilize the "nn.CrossEntropyLoss" function. This loss function is fundamental in measuring the dissimilarity between the predicted class probabilities and the actual class labels present in the ground truth masks. By calculating this dissimilarity, the loss function guides the training process to minimize it, aiding the model in learning how to generate accurate segmentation masks during backpropagation and gradient descent. Our choice of the "sum" reduction parameter effectively accounts for the impact of different inputs, providing a comprehensive loss calculation. The utilization of "nn.CrossEntropyLoss" plays a fundamental role in steering the training process toward successful image segmentation.

5.2.2 Optimizer

To update the model's parameters during training, we employ an optimizer, such as stochastic gradient descent (SGD) or Adam. The choice of the optimizer, along with its learning rate, is essential to ensure that the model converges efficiently and minimizes the loss function. In this specific implementation, we chose the Adam optimizer for several reasons. Adam is known for its adaptability in adjusting the learning rates for each parameter individually, which can lead to faster convergence and improved model performance. This adaptive nature is particularly beneficial in scenarios where different model parameters may have varying sensitivities to the learning rate. By using Adam, we aim to enhance the fine-tuning process of the U-Net model and achieve more accurate image segmentation results.

6 Limitations

Our project is not without limitations, and it's essential to acknowledge these constraints and challenges:

6.1 Data Limitations

Biomedical segmentation tasks often suffer from a scarcity of annotated data for training. Given the data-driven nature of deep learning, having an ample dataset is crucial for achieving high model performance. While U-Net is known to leverage data augmentation techniques to address this limitation, it's worth noting that we didn't implement data augmentation in our specific project, but we mentioned it in the Future Suggestions section. The availability of diverse and representative training data remains a bottleneck in the broader adoption of U-Net in biomedical applications. Furthermore, the quality and consistency of annotations in medical images can vary, introducing additional challenges.

6.2 Computational Efficiency

Computational efficiency can be a concern when deploying U-Net models in different hardware configurations. The architecture's extensive depth and parameter count, particularly in scenarios with high-resolution inputs, can strain computational resources, making it less accessible for resource-constrained environments or real-time applications. Achieving a balance between model complexity and computational efficiency is an ongoing challenge, as optimizing U-Net for various hardware platforms and deployment scenarios requires careful consideration.

6.3 Evaluation Challenges

The implementation of evaluation metrics in biomedical segmentation tasks can be more challenging than in other visual recognition tasks. This is primarily due to the requirement for precise pixel-level localization, which demands not only accurate segmentation but also an understanding of the spatial relationships within the segmented regions. Therefore, selecting appropriate evaluation metrics that effectively capture these nuances remains a significant challenge in the field.

7 Future Suggestions

While our project achieved the primary objectives, there are areas for improvement and additional features to consider in future work. Here are some future suggestions:
1. **Data Augmentation:** Despite not being implemented in this phase, data augmentation techniques can be applied to increase the diversity of the training dataset and improve model robustness.
2. **Weighted Loss Functions:** Investigate the use of weighted loss functions to address class imbalance issues in segmentation tasks, which can further enhance model performance.
3. **Batch Normalization:** We incorporated batch normalization to stabilize training. Future work could explore different normalization techniques and their impact on model convergence.
4. **Image Tiling:** Implement image tiling during inference to handle large images efficiently. This technique can significantly improve results when dealing with images larger than the training input size.

5. **Other Architectural Variations:** Experiment with different architectural variations, such as using more advanced network architectures or incorporating attention mechanisms to improve performance further.
6. **Performance Optimizations:** Optimize model inference for resource efficiency, enabling real-time or near-real-time applications.

8 Result Examples

In this section, we will provide several examples of input images, ground truth masks, and predicted masks.

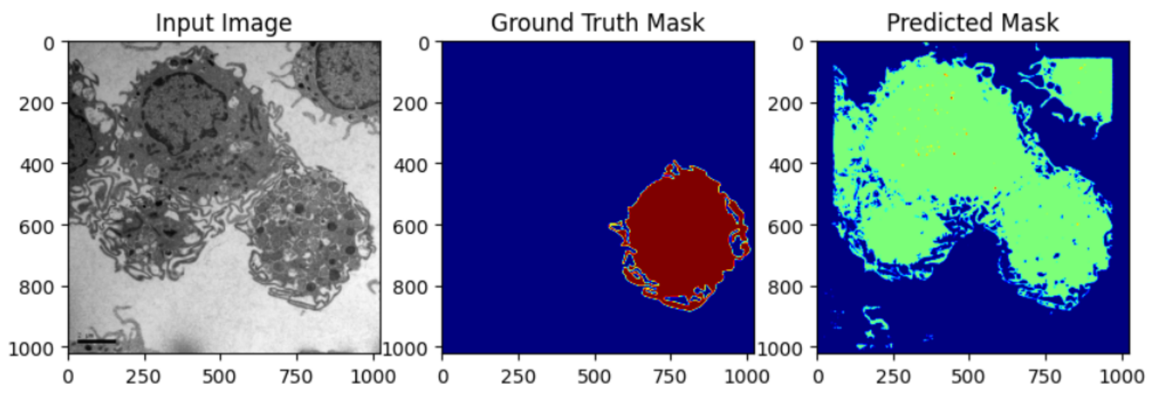


Figure 5: *Mask Prediction Example No: 1*

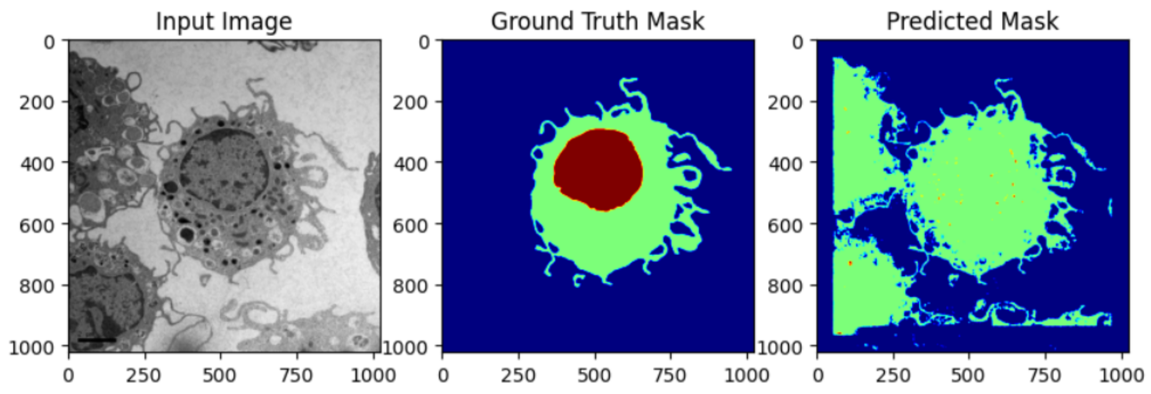


Figure 6: *Mask Prediction Example No: 2*

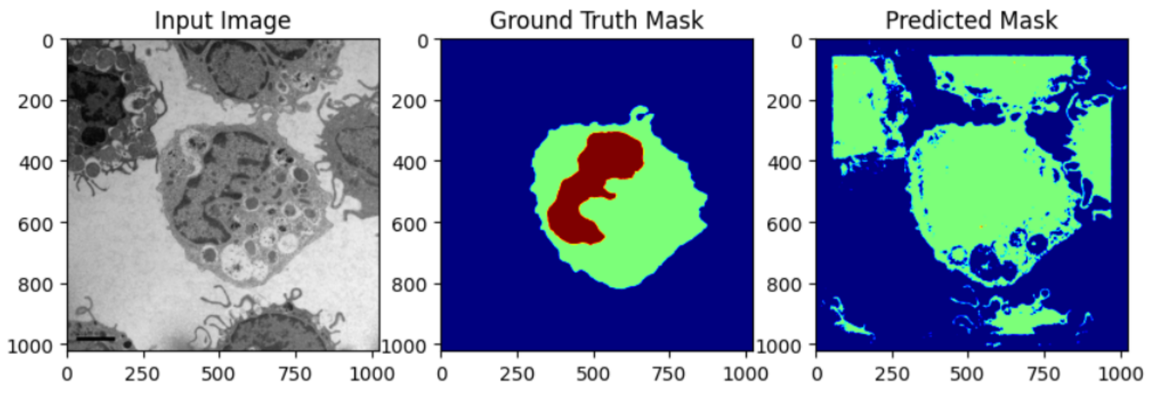


Figure 7: *Mask Prediction Example No: 3*

9 Conclusion

To conclude, our U-Net based biomedical image segmentation project stands as a testament to the dedication and collaborative efforts of our team. We are immensely grateful for the valuable experience and knowledge we have gained throughout this journey. Working together, we successfully overcame various challenges during the implementation phase, further enhancing our problem-solving skills and deepening our understanding of biomedical image segmentation.
The remarkable results we achieved, characterized by high accuracy, F1-score, and Jaccard Index, reflect the precision and efficacy of our segmentation model. This project has not only been about achieving outcomes but has also been a platform for personal and collective growth.
We look back on our project with appreciation for the opportunity to learn, explore, and contribute to the field of biomedical image analysis. As we move forward, we carry this experience with us, ready to take on new challenges and continue our journey of discovery and innovation in the world of deep learning and image segmentation.

References

[1] *"U-Net: Convolutional Networks for Biomedical Image Segmentation Olaf Ronneberger(2015) by Ronneberger et al."* `https://browse.arxiv.org/pdf/1505.04597.pdf`.

[2] *"Jurkat Cell dataset generated by University of Freiburg"*. `https://lmb.informatik.uni-freiburg.de/resources/datasets/tem.en.html`.