ENHANCED CELL SEGMENTATION THROUGH INTEGRATED SPATIAL TRANSFORMER NETWORK IN UNET

A Main Project submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
In
ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Submitted by

1. **Sasi Sandeep Singh Bhemal - 20PA1A5408**
2. **Durga Pavan Pendyala - 20PA1A5442**
3. **Manikanta Yadala - 20PA1A5455**
4. **Likith Vijay Sai Lakkoju - 20PA1A5427**
5. **Rahul Kuruganti - 20PA1A5426**

Under the esteemed guidance of

Dr. P Sita Ram Murty

Associate Professor

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

VISHNU INSTITUTE OF TECHNOLOGY
(Autonomous)
(Approved by AICTE, Accredited by NBA & NAAC and permanently affiliated to JNTU Kakinada)
BHIMAVARAM – 534 202

2023 - 2024

VISHNU INSTITUTE OF TECHNOLOGY

(Autonomous)

(Approved by AICTE, accredited by NBA & NAAC, and permanently affiliated to JNTU Kakinada)

BHIMAVARAM-534202

2023-2024

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

VISHNU
UNIVERSAL LEARNING

CERTIFICATE

This is to certify that the project entitled "ENHANCHED CELL SEGMENTATION THROUGH INTEGRATED SPATIAL TRANSFORMER NETWORKS IN U-NET", is being submitted by **B. Sasi Sandeep Singh, P. Durga Pavan, Y. Manikanta, L. Likith Vijay Sai AND K. Rahul** bearing the **REGD.NOS : 20PA1A5408, 20PA1A5442, 20PA1A5455, 20PA1A5427, 20PA1A5426** submitted in fulfilment for the award of the degree of **"BACHELOR OF TECHNOLOGY"** in **"ARTIFICIAL INTELLIGENCE AND DATA SCIENCE"** is a record of work carried out by them under my guidance and supervision during the academic year 2023-2024 and it has beenfound worthy of acceptance according to the requirements of university.

Internal Guide

Dr. P Sita Ram Murty

Head of the Department

Dr. P Sita Ram Murty

External Examiner

ACKNOWLEDGEMENT

It is natural and inevitable that the thoughts and ideas of other people tend to drift into the subconscious due to various human parameters, where one feels the need to acknowledge the help and guidance derived from others. We express our gratitude to those who have contributed to the fulfillment of this project.

We take the opportunity to extend our sincere thanks to **Dr. D. Suryanaryana**, the director of VIT, Bhimavaram, whose guidance from time to time helped us complete this project successfully.

We also express our sincere gratitude to **Dr. M. Venu**, the principal of VIT, Bhimavaram, for his continuous support in helping us complete the project on time.

We are thankful to **Dr. P. Sita Rama Murty**, Head of the Department of Artificial Intelligence and Data Science, for his continuous and unwavering support and guidance. We acknowledge our gratitude for his valuable guidance and support extended to us from the conception of the idea to the completion of this project.

We are very thankful to **Dr. P Sita Ram Murty**, Associate Professor, our internal guide, whose guidance from time to time helped us complete this project successfully.

Project Associates

B. Sasi Sandeep Singh (20PA1A5408)
P. Durga Pavan (20PA1A5442)
Y. Manikanta (20PA1A5455)
L. Likith Vijay Sai (20PA1A5427)
K. Rahul (20PA1A5426)

ABSTRACT

Cell segmentation is a crucial task in biomedical image analysis, facilitating the study of cellular morphology, function, and pathology. While previous segmentation models based on Unet architecture have shown promise, they often encounter challenges such as inaccuracies in boundary delineation and robustness to image variations. In this study, we propose a novel approach to cell segmentation by integrating a Spatial Transformer Network (STN) into the Unet framework. The STN module enables adaptive spatial transformation of input images, allowing the network to focus on relevant regions and improve segmentation accuracy. Leveraging the strengths of both Unet and STN, our model addresses limitations of existing segmentation methods and achieves superior performance in segmenting complex cell structures, handling variations in cell morphology, and robustness to noise and artifacts. Experimental results on benchmark datasets demonstrate the effectiveness of the proposed approach, outperforming state-of-the-art segmentation models. Our integrated framework offers a versatile and efficient solution for cell segmentation in biomedical imaging, with potential applications in cell biology, pathology, and drug discovery.

Keywords: Cell segmentation, Deep learning, Spatial Transformer Networks (STNs), U-Net architecture, Biomedical image analysis, Serial-section Transmission Electron Microscopy (ssTEM), Neural structures, Automated segmentation

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Cell segmentation, the process of identifying and delineating individual cells within microscopic images, is a fundamental task in biomedical image analysis with numerous applications ranging from basic research to clinical diagnostics. Accurate segmentation of cells is crucial for understanding cellular morphology, spatial organization, and function, thereby facilitating advancements in areas such as cancer detection, drug discovery, and tissue engineering. However, the complex and heterogeneous nature of biological tissues presents significant challenges for automated cell segmentation algorithms. Variations in cell morphology, staining artifacts, imaging noise, and overlapping structures can hinder the performance of traditional segmentation methods. To address these challenges, recent research has increasingly turned to deep learning techniques, leveraging the power of convolutional neural networks (CNNs) to achieve more robust and accurate segmentation results.

This project focuses on the development and evaluation of an innovative approach for enhanced cell segmentation by Integrating Spatial Transformer Networks (STNs) within the U-Net architecture. The U-Net model, originally proposed by Ronneberger et al. (2015), has gained widespread popularity in biomedical image segmentation tasks due to its unique architecture, which combines contracting and expansive pathways to capture both local and global context information. By integrating STNs into the U-Net framework, we aim to enhance the model's ability to adapt to variations in cell morphology and spatial arrangements, ultimately improving segmentation accuracy and robustness.

Spatial Transformer Networks (STNs) were introduced by Jaderberg et al. (2015) as a mechanism for end-to-end learning of spatial transformations within convolutional neural networks. STNs enable neural networks to perform spatial transformations, such as translation, rotation, scaling, and non-rigid deformation, directly on the input data. By incorporating STNs into the U-Net architecture, we can empower the model to dynamically adjust its receptive field and spatial resolution, effectively localizing and segmenting individual cells while mitigating the effects of variations in cell size, shape, and orientation.

The proposed approach offers several advantages over traditional cell segmentation methods. Firstly, by learning spatial transformations from the input data, the model can adapt to diverse cell morphologies and spatial arrangements, making it more robust to variations in image quality and staining protocols. Secondly, the end-to-end training of the integrated U-Net with STNs allows for the simultaneous optimization of both the segmentation network and the spatial transformation parameters, leading to improved overall performance. Additionally, the use of deep learning techniques enables the model to capture complex image features and contextual information, further enhancing segmentation accuracy.

In this project, we present a comprehensive evaluation of the Enhanced Cell Segmentation through Integrated Spatial Transformer Networks in U-Net approach. We describe the methodology for model development, training, and evaluation, including details of the network architecture, data preprocessing techniques, and experimental setup. We employ publicly available benchmark datasets containing a diverse range of microscopic images to assess the performance of the proposed method. Performance metrics such as pixel-wise accuracy, intersection over union (IoU), and Dice coefficient are utilized to quantitatively evaluate segmentation accuracy and compare against baseline methods. Overall, this project contributes to the advancement of cell segmentation techniques by leveraging the capabilities of deep learning and spatial transformation networks, offering a promising solution for accurate and efficient analysis of biomedical images in research and clinical settings.

1.1 PROBLEM STATEMENT

.

While U-Net and Spatial Transformer Networks (STNs) have individually demonstrated effectiveness in cell segmentation, their standalone applications exhibit certain limitations. U-Net struggles with generalization across diverse cell types, shapes, and sizes, facing challenges in scenarios with overlapping structures and limited annotated data. On the other hand, STNs, while offering valuable spatial transformation capabilities, may lack robustness to irregular cell morphologies, demonstrate sensitivity to hyperparameters, and pose computational challenge.

2 SYSTEM ANALYSIS

2.1.1 Hardware Requirements

Processor - quad-core or higher (I3/Intel Processor or above)

RAM - Minimum 16GB

GPU - Mid-Range to High-End (>= 4GB VRAM)

HDD/SSD - Minimum of 256 GB

Key Board - Standard Windows Keyboard

Mouse - Two or Three Button Mouse

Monitor - SVGA

2.1.2 Software Requirements

Operating System - Windows 10/11

Programming Language - Python

Essential Libraries - TensorFlow, Keras OpenCV, Numpy, Matplotlib and Seaborn, Pandas, Gradio, Streamlit.

Tools - Git, Github

IDE/Workbench - Visual Studio Code, PyCharm, JupyterLab, Kaggle, Colab.

Version - Python 3.6+

2.2 EXISTING SYSTEM & DISADVANTAGES

Existing systems for cell semantic segmentation encompass a variety of approaches, including traditional image processing methods and deep learning-based models. Traditional methods often involve thresholding, edge detection, and morphological operations, but struggle with complex cell structures and variations. Deep learning approaches, such as U-Net, Mask R-CNN, and FCN, have gained prominence for their ability to learn hierarchical features and capture intricate cell boundaries. These models require annotated data for training and leverage architectures tailored for semantic segmentation tasks.

- **Reliance on Handcrafted Features**: Traditional methods heavily depend on manually designed features, which may not capture all the nuances in cell images, leading to less accurate segmentation.

- **Manual Parameter Tuning**: These methods often require manual adjustment of parameters, which can be time-consuming and subjective, potentially leading to suboptimal results.

- **Limited Robustness**: Traditional approaches may struggle to handle variations in cell appearance, staining, and imaging conditions, resulting in segmentation errors when faced with diverse datasets.

- **Difficulty with Complex Cell Structures**:Traditional methods may have difficulty accurately segmenting cells with irregular shapes, overlapping boundaries, or complex structures, limiting their applicability in challenging scenarios.

- **Scalability Issues**:Scaling traditional methods to handle large datasets or high-resolution images can be computationally expensive and impractical due to limitations in computational resources and efficiency.

2.3 PROPOSED SYSTEM & ADVANTAGES

In response to the limitations of existing cell segmentation methods, our proposed system introduces an innovative approach that integrates Spatial Transformer Networks (STNs) within the U-Net architecture to enhance segmentation accuracy and robustness. The proposed system leverages the flexibility and adaptability of STNs to dynamically adjust the spatial transformations applied to input images, thereby addressing challenges related to variations in cell morphology, staining artifacts, and imaging conditions.

- **Adaptability to Variations in Cell Morphology**: Leveraging STNs, the system adapts to diverse cell morphologies, enhancing robustness to image quality, staining protocols, and biological samples. Learning spatial transformations from data enables dynamic adjustment to various imaging conditions without manual parameter tuning.

- **Improved Segmentation Accuracy**: Integrating STNs within U-Net enhances the model's ability to capture spatial relationships and contextual information, improving segmentation accuracy.

- **Reduced Dependency on Annotated Data**: The system leverages self-supervised learning to learn spatial transformations from unlabeled data, reducing reliance on annotated datasets and enhancing generalization to unseen samples and conditions.

- **Efficient Use of Computational Resources**: Despite enhanced capabilities, the system maintains computational efficiency through lightweight architectures, making it suitable for real-time applications and resource-constrained environments.

- **End-to-End Learning**: Facilitating end-to-end learning of both segmentation and spatial transformation parameters ensures joint optimization, leading to improved overall performance.

2.4 FEASIBILITY STUDY

Technical Feasibility:

Assessing the capability of the proposed system's development using available technology and resources, the integration of Spatial Transformer Networks (STNs) within the U-Net architecture appears technically feasible. Established components in deep learning, including STNs and U-Net, are supported by numerous open-source libraries like TensorFlow and PyTorch, streamlining development. High-performance computing resources, such as GPUs and cloud platforms, facilitate efficient model training and evaluation. Challenges may arise in optimizing model architecture, hyperparameters, and resource management.

Economic Feasibility:

Evaluating cost-effectiveness, while deep learning frameworks and computing resources are widely accessible, specialized hardware like GPUs may incur costs. Personnel with expertise in deep learning, computer vision, and biomedical image analysis may also be required, adding expenses. Long-term benefits include improved accuracy, reduced labor costs, and enhanced productivity in research and clinical settings, potentially outweighing initial investments.

Operational Feasibility:

Examining seamless integration into existing workflows, operational feasibility depends on user interface design, deployment ease, and compatibility with current image analysis pipelines. A user-friendly interface for training, evaluation, and inference is essential, ensuring efficient interaction for researchers and clinicians.

Conclusion:

The feasibility study indicates the proposed system's viability for enhanced cell segmentation with Integrated Spatial Transformer Networks in U-Net. Though challenges like model optimization and resource management may arise initially, potential benefits in accuracy, efficiency, and scalability justify further development.

3 SYSTEM DESIGN

3.1 BACKGROUND

Historically, conventional image analysis methods and shallow learning algorithms were employed for segmenting EM images. These include statistical analysis of pixel neighborhoods (Kylberg et al., 2012), eigenvector analysis (Frangakis and Hegerl, 2002), watershed and hierarchical region merging (Liu et al., 2012, Liu et al., 2014), superpixel analysis and shape modeling (Karabağ et al., 2019), and random forest (Cao et al., 2019). However, in recent years, deep learning (DL) has emerged as the dominant approach in this field, mirroring the trends observed in segmentation techniques for light microscopy and other medical imaging modalities (Liu et al., 2021, Litjens et al., 2017).

Compared to traditional image analysis and machine learning methods that rely on handcrafted features, DL-based segmentation eliminates or significantly reduces the need for domain-specific knowledge to extract relevant features from the imaged sample (Liu et al., 2021).. DL-based segmentation has gained popularity, leading to the development of plug-ins for commonly used biomedical image analysis tools like CellProfiler (Carpenter et al., 2006), ImageJ (Schindelin et al., 2012), Weka (Arganda-Carreras et al., 2017), and Ilastik (Berg et al., 2019), which were previously limited to traditional image processing or shallow learning.

We review the recent progress of automatic image segmentation in EM, with a focus on the last six years that marked significant progress in both DL-based semantic and instance segmentation, while also giving an overview of the main DL architectures that enabled this progress.

Despite the advancements in deep learning-based approaches, challenges remain in areas such as generalization to unseen data, interpretability of model decisions, and scalability to large datasets. Additionally, the integration of domain knowledge, validation of results, and usability in real-world biomedical applications are ongoing areas of research and development in the field of cell segmentation.

3.1.1 RELATED WORK

In the last five years, deep learning has gained much attention, largely because it has surpassed the human level in solving many complex problems. It is comprised of many perceptron layers that form a deep neural network. In visual recognition tasks, this type of architecture can learn to recognize patterns such as handwritten digits and other features of interests in images hierarchically. However, the main drawback of using deep neural networks is that it requires a huge amount of data for training the network. In order to overcome this issue, researchers have started to collect a large database which contains millions of images from hundreds of categories.

Since then, many advanced architectures have been introduced including VGG , Googlenet . Computers are now able to mimic artistic painting to produce new pictures by transferring the style from one image to another . In addition, researchers are also actively working on extending deep learning methods for medical image data beyond the scope of natural images . These approaches impose vast changes in automatic classification and segmentation on other image modalities, such as CT and MRI . These studies have opened a revolutionary era in which software can self-program to achieve or even outperform human capabilities in image processing and computer vision areas. Deep learning has been quickly adopted by connectomics research for automatic EM image segmentation.

One of the earliest applications to EM segmentation was made by Ciresan et al.. This method involves the straightforward application of a CNN for pixel-wise membrane probability estimation and it won the ISBI 2012 challenge. As new deep learning methods are introduced, automatic EM segmentation techniques evolves, as well. One notable recent advancement in the machine learning domain is the introduction of a fully convolutional neural network (FCN) for the end-to-end semantic segmentation problem. Inspired by this work, many successive variants of FCN have been proposed for EM image segmentation.

Chen et al. proposed multi-level upscaling layers and their combination for final segmentation. A new-post processing step, namely lifted multi-cut, was also introduced

to refine the segmentation. Ronneberger et al. presented skip connections for concatenating feature maps in their U-net architecture. Although U-net and its variants can learn multi-contextual information from the input data, they are limited in the depth of the network they can construct because of the vanishing gradient problem. Recently, the 3D extension of U-net was proposed for confocal microscopy segmentation .

U-Net demonstrated exceptional performance in the cell tracking and segmentation challenge at the IEEE International Symposium on Biomedical Imaging. Designed for semantic segmentation, U-Net effectively captures contextual and local information within images, making it well-suited for biomedical cell segmentation tasks.

Additionally, we explored the integration of Spatial Transformer Networks (STNs) with U-Net. STNs enhance adaptability to variations in cell morphology, improving robustness to diverse image conditions. By dynamically adjusting to different imaging scenarios, the combined U-Net and STN framework achieves more precise localization and segmentation of individual cells, essential for comprehensive cell analysis.

The incorporation of STNs alongside U-Net signifies a promising direction in advancing semantic segmentation techniques for biomedical applications, offering enhanced adaptability and accuracy in cell segmentation tasks.

In recent years, the field of biomedical imaging has witnessed remarkable advancements driven by deep learning techniques like U-Net and innovative integration strategies such as Spatial Transformer Networks (STNs). Building upon the success of U-Net in cell tracking and segmentation challenges, researchers have been exploring ways to further enhance its capabilities for intricate biomedical tasks. One avenue of exploration involves refining the architecture of U-Net to better accommodate the complexities inherent in biomedical images. This includes incorporating attention mechanisms or multi-scale features to improve the model's ability to capture subtle variations in cell morphology and spatial relationships. By enhancing the network's understanding of context and local features, these adaptations can significantly boost its performance in challenging biomedical segmentation tasks.

PARAMETERS	SIMPLE CLASSICAL METHODS (EX. HISTOGRAM OR THRESHOLDING)	BETTER CLASSICAL METHODS (EX. WATERSHED)	Al MODELS (EX. MASK R-CNN)
Time	Fast if only low accuracy needed	Much slower than simple classical methods	Takes a while to train and implement, fast once model is trained
Software Performance	Not very accurate	More accurate	Most accurate by far with a good model
Cost	Not very accurate	Many free implementations available	Design model and train yourself or may be very expensive
Ease of Use	Usually out-of-the-box, largest community with experience	Simple, can find implementations online - may require coding experience	May be more difficult for scientists, even in a commercial product
Analysis Speed	Usually out-of-the-box, largest community with experience	Slow, may run into major scaling issues	Requires specialized GPU infrastructure to scale
Scalability	Usually out-of-the-box, largest community with experience	Many free programs available	Code and train yourself or can be very expensive
Parameter Tuning	Requires few parameters	Requires more parameters	Can usually generalize parameters - much less tuning

Table 3.1 (Comparison Table for Cell Segmentation Methods)

3.2 DATASET DESCRIPTION

The Dataset contains 110 sections from a serial section Transmission Electron Microscopy (ssTEM) data set of the Drosophila first instar larva ventral nerve cord (VNC). The microcube measures 2 x 2 x 1.5 microns approx., with a resolution of 4x4x50 nm/pixel. This data set to train machine learning software for the purpose of automatic segmentation of neural structures in ssTEM.

The images are representative of actual images in the real-world: there is a bit of noise; there are image registration errors; there is even a small stitching error in one section. None of these led to any difficulties in the manual labeling of each element in the image stack by an expert human neuroanatomist. A software application that aims at removing or reducing human operation must be able to cope with all these issues.

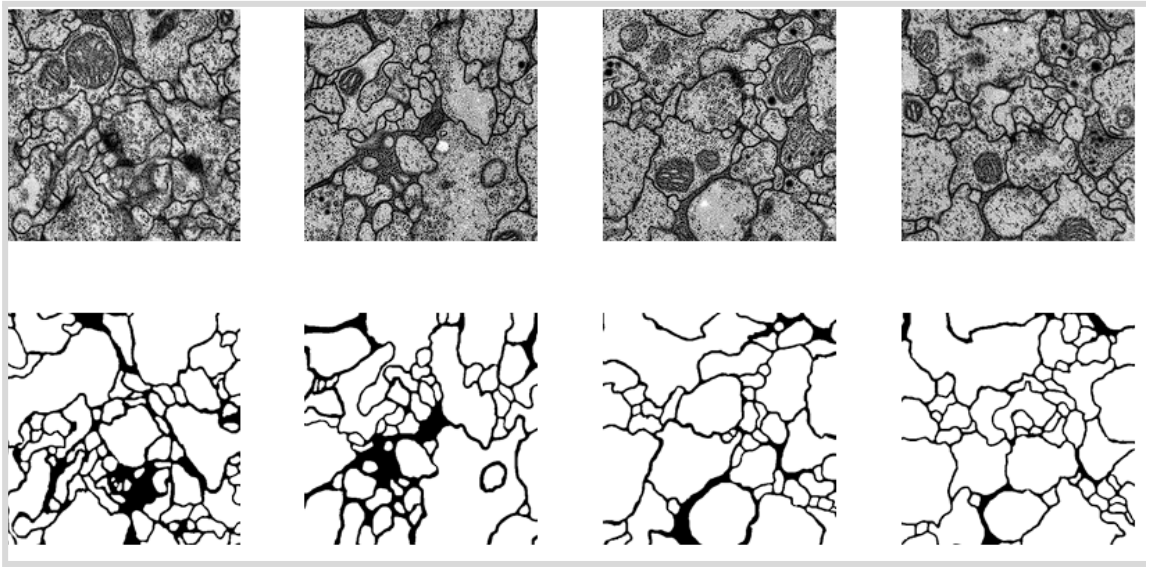


Fig 3.1 (Sample Dataset Images and Corresponding Masks)

Serial-section transmission or scanning EM (ssTEM or ssSEM) is used for studying synaptic junctions and highly-resolved membranes in neural tissues. Advances in microscopy techniques in serial section EM have enabled the study of neurons with increased connectivity in complex mammalian tissues (such as mice and humans) and even whole brain tissues of smaller animal models, like the fruit fly and zebrafish. This

imaging approach visualizes the generated volumes in a highly anisotropic manner, i.e. the x and y directions have a high resolution, however, the z-direction has a lower resolution, as it is reliant on serial cutting precision.

The Drosophila larvae dataset of the ISBI 2012 challenge was the first notable EM dataset for automatic neuronal segmentation, featuring two volumes with 30 sections each. The main challenge of that dataset is to develop algorithms that can accurately segment the neural structures present in the EM images. The success of deep neural networks as pixel classifiers in the ISBI 2012 challenge (Ciresan et al., 2012) paved the way for deep learning in serial section EM segmentation.

In conclusion, the ssTEM dataset of the Drosophila larva ventral nerve cord provides a comprehensive resource for training machine learning algorithms to automatically segment neural structures. With its realistic representation of real-world challenges such as noise and registration errors, this dataset enables the development of robust segmentation models capable of coping with diverse imaging conditions. The success of deep learning approaches in previous challenges, like the ISBI 2012 challenge, highlights the potential of these techniques to revolutionize serial-section EM segmentation and pave the way for further advancements in neuroscience.

3.3 DATA PRE-PROCESSING STEPS

- **Data Splitting**:

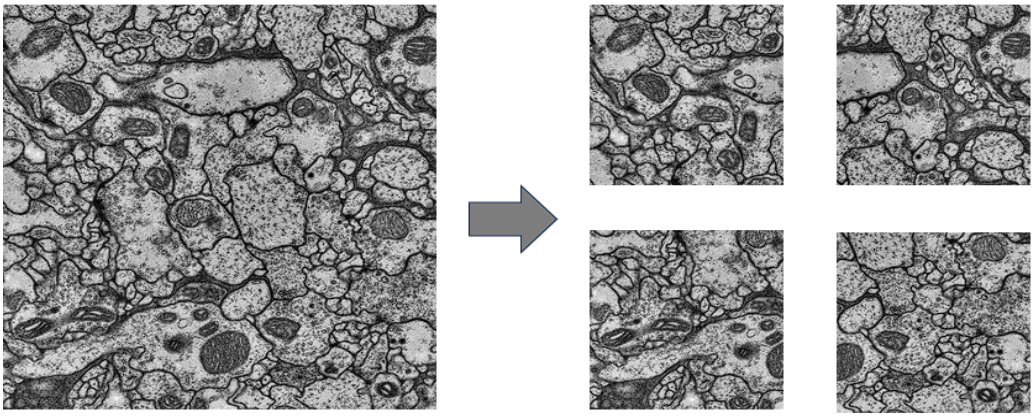


Fig 3.2 (Data Splitting)

We have split our dataset of images, each with a size of 1024x1024 pixels, into smaller images through a process called data splitting or image tiling. Each original image, which may have been larger in size, has been divided into smaller, non-overlapping tiles or patches.

- **Color Inversion**:

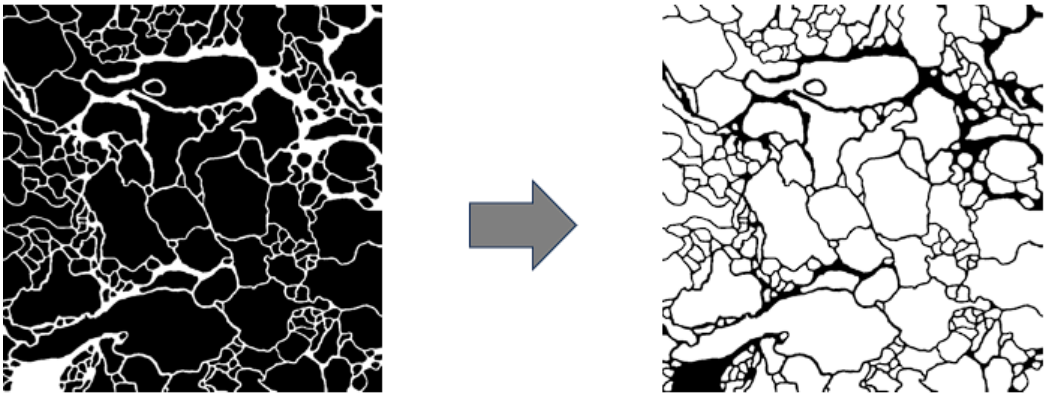


Fig 3.3 (Color Inversion)

We have applied color inversion to our images, swapping black and white pixel values. This technique transforms white regions into black and vice versa in our images. By inverting the colors, we aim to enhance the contrast and visibility of objects within the images, which can aid in segmentation tasks and improve the interpretability of our data.

- **Rotation**:

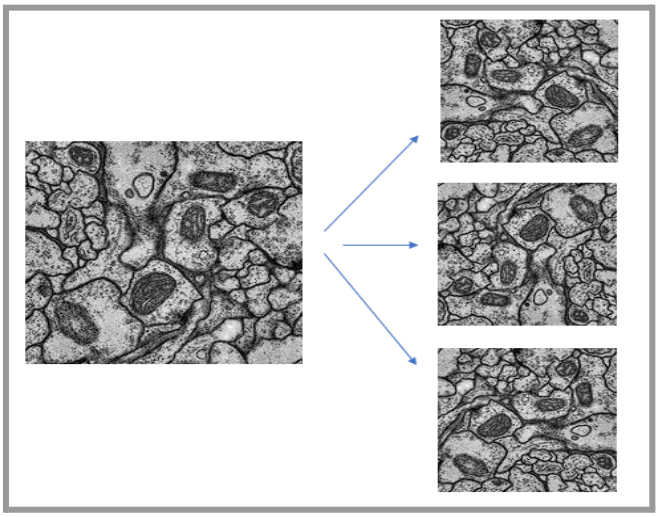


Fig 3.4 (Rotation)

We have applied rotation to our images, rotating them by 90, 180, and 270 degrees. This helps introduce variability in the orientation of objects in the images, making our dataset more robust and diverse. Rotation alters image orientation around its center, useful for addressing orientation bias in datasets and improving the model's ability to recognize objects regardless of their orientation in images.

- **Flipping**:

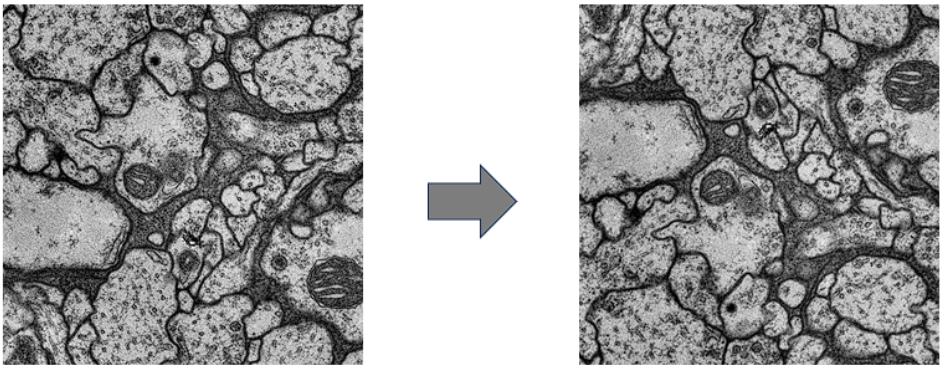


Fig 3.5 (Flipping)

We have flipped our images horizontally and vertically. Horizontal flipping mirrors the images along the vertical axis, while vertical flipping mirrors them along the horizontal axis. Flipping helps ensure that our model learns invariant features from different orientations of objects.

- **Scaling**:

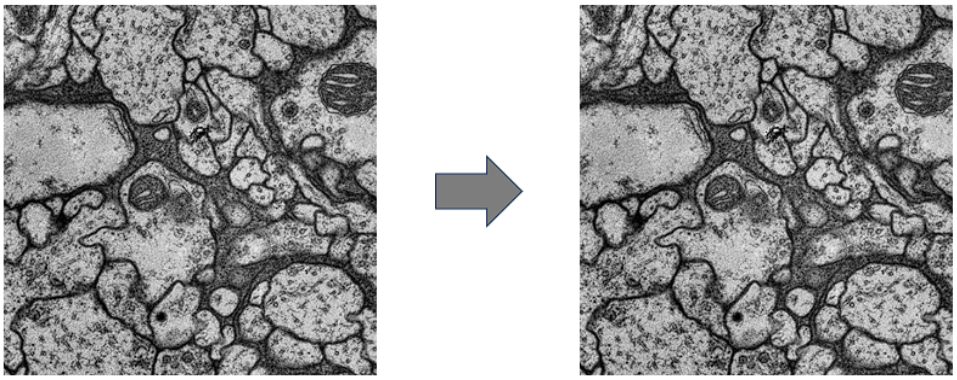


Fig 3.6 (Scaling)

Scaling resizes images while preserving aspect ratios, facilitating data augmentation and standardizing object sizes for model training. It ensures consistency in feature representation across different scales, enhancing model generalization and performance on images with varying resolutions and object sizes.

- **Translation**:

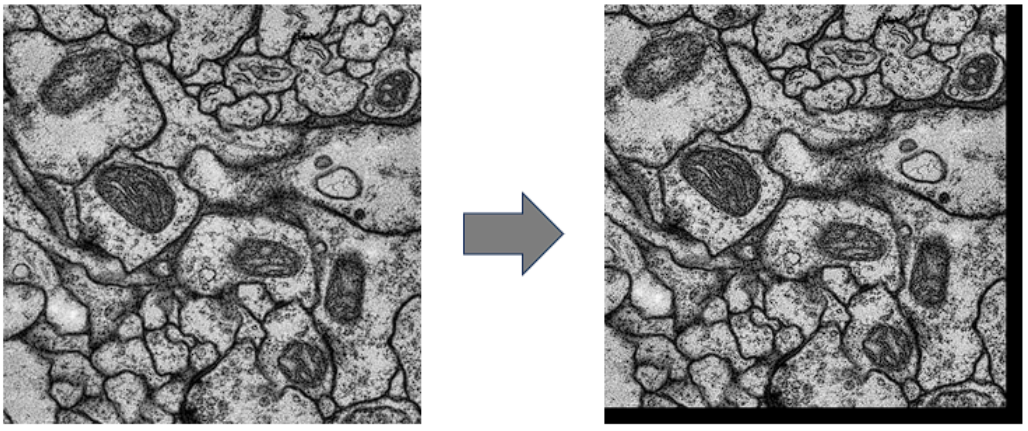


Fig 3.7 (Translation)

We have translated our images by shifting them 20 pixels to the left and up, and 20 pixels to the right and down. Translation helps simulate changes in the position of objects within the images, enhancing the robustness of our model to object location variations. Translation shifts images along x and y axes, simulating viewpoint changes and spatial variations.

- **Shearing**:

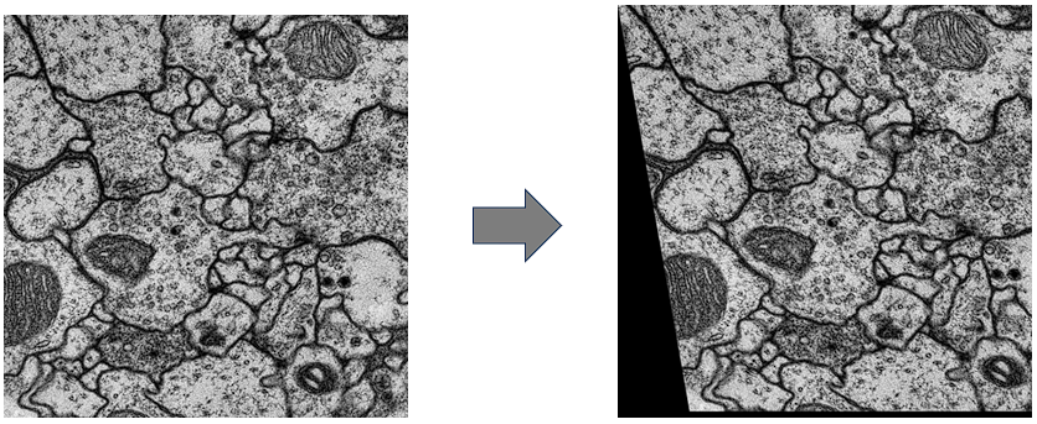


Fig 3.8 (Shearing)

Shearing distorts images by skewing along an axis, compensating for perspective distortions caused by viewing angles. It aids in training models robust to perspective variations, improving performance in scenarios where objects are viewed from different angles or perspectives.

- **Brightness and Contrast Adjustment**:

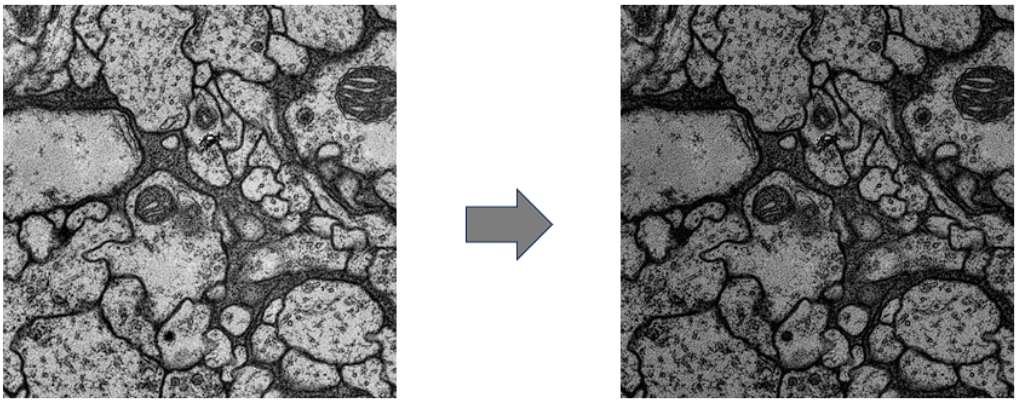


Fig 3.9 (Brightness and Contrast Adjustment)

Brightness adjustment modifies overall brightness levels, improving image visibility and ensuring consistent illumination conditions across datasets. Contrast adjustment alters the difference in intensity between the brightest and darkest parts of an image, enhancing image clarity and feature visibility. We have adjusted the brightness and contrast of our images by varying alpha (contrast) and beta (brightness) values.

3.4 ARCHITECTURE

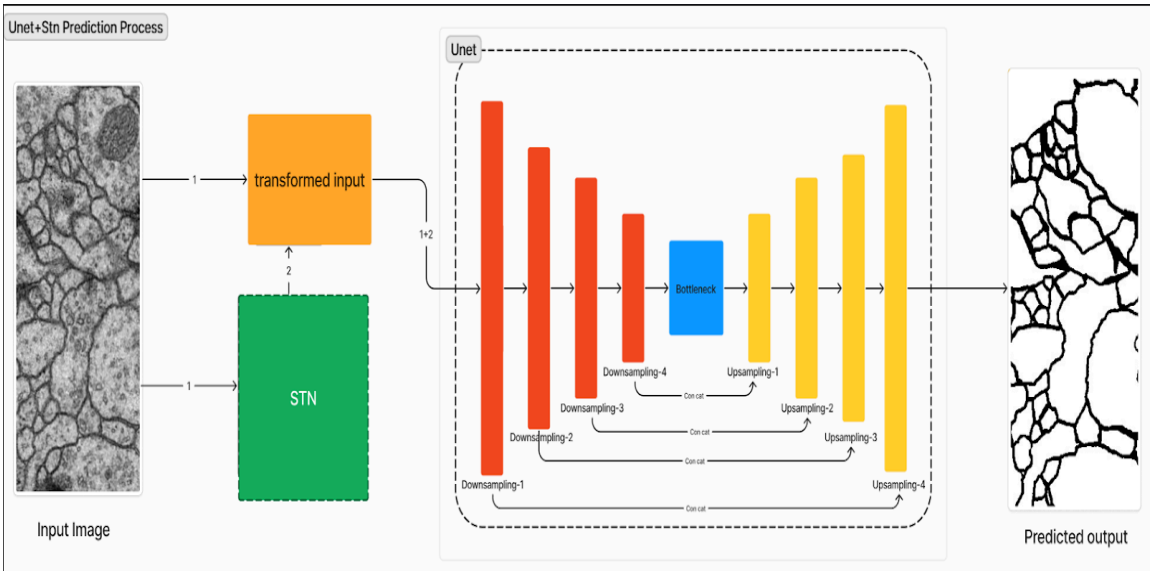


Fig 3.10 (Architecture)

Input Layer:

We start with an input layer that receives grayscale images of size (image_size, image_size, 1), where image_size represents the dimensions of our images.

Spatial Transformer Network (STN):

Our images are then passed through a Spatial Transformer Network (STN). This network consists of convolutional layers followed by max-pooling layers to capture spatial features. These features are then processed by dense layers to predict transformation parameters, such as rotation, scaling, and translation.

The architecture of a Spatial Transformer Network (STN) typically consists of three main components:

***Localization Network*:** This sub-network takes the input data and learns to predict transformation parameters, such as translation, rotation, and scaling, that best align

the input data to a canonical space. It typically consists of convolutional layers followed by fully connected layers to output transformation parameters.

***Grid Generator*:** This component generates a grid of coordinates based on the predicted transformation parameters. The grid represents the spatial mapping from the input image to the canonical space, facilitating the transformation process.

***Sampler*:** The sampler uses the generated grid to perform the actual spatial transformation of the input data. It samples the input image according to the coordinates provided by the grid, effectively applying the learned transformation to the input data.

These components work together to enable the STN to learn spatial transformations directly from the data, allowing neural networks to effectively attend to relevant regions of the input and improve overall performance in various tasks such as image classification, object detection, and geometric data analysis.

Concatenation with Original Input:

Following the transformation process, the resulting images from the Spatial Transformer Network (STN) are merged with the original input images. This strategic fusion ensures that our model benefits from the insights provided by both the unaltered and transformed representations, fostering a more comprehensive learning experience and enhancing its capacity for accurate and adaptable image analysis. This integration ensures that our model not only captures the inherent features of the original images but also learns to adapt to variations introduced by the spatial transformations.

Downsampling Blocks:

Next, the concatenated images go through a series of downsampling blocks. Each block includes two convolutional layers with ReLU activation functions, followed by max-pooling layers. These layers help extract hierarchical features from the images while reducing their spatial dimensions.

***Purpose*:** The downsampling block aims to gradually reduce the spatial dimensions of

the input while increasing the depth of feature maps, allowing the network to capture hierarchical features.

Components:

Two Convolutional Layers: Each convolutional layer performs feature extraction using learnable filters. The first layer extracts basic features, while the second layer captures more complex patterns.

ReLU Activation Function: Applied after each convolutional layer to introduce non-linearity, enabling the model to learn complex representations.

MaxPooling Layer: Downsamples the spatial dimensions of the feature maps by selecting the maximum value within each pooling window. This helps in preserving the most salient features while reducing spatial resolution.

Dropout Layer: Regularizes the network by randomly setting a fraction of input units to zero during training, preventing overfitting.

***Function*:** The downsampling block extracts low-level features from the input image while reducing its spatial dimensions, enabling the network to learn abstract representations of the input.

Bottleneck Block:

After downsampling, the features are passed through a bottleneck block comprising two convolutional layers with ReLU activation functions. This block extracts high-level features from the input, capturing complex patterns in the data.

***Purpose*:** The bottleneck block serves as a bridge between the downsampling and upsampling paths, allowing the network to capture high-level features while maintaining spatial information.

Components:

Two Convolutional Layers: Similar to the downsampling block, these layers extract complex features from the input. However, they maintain the spatial dimensions of the

feature maps.

ReLU Activation Function: Applied after each convolutional layer to introduce non-linearity and enhance the network's expressive power.

Dropout Layer: Helps prevent overfitting by randomly deactivating a fraction of neurons during training.

***Function*:** The bottleneck block condenses the spatial information while extracting high-level features from the input, facilitating better representation learning and feature extraction.

Upsampling Blocks:
The features obtained from the bottleneck block are then upsampled using a series of upsampling blocks. Each block includes two convolutional layers with ReLU activation functions. These layers refine the features while restoring the spatial resolution of the images.

***Purpose*:** The upsampling block aims to reconstruct the original spatial resolution of the input while refining the extracted features from the bottleneck block.

Components:
UpSampling Layer: Increases the spatial dimensions of the feature maps by duplicating the rows and columns.

Concatenation with Skip Connection: Concatenates the upsampled features with the corresponding features from the downsampling path. This skip connection helps in preserving spatial details and overcoming information loss during downsampling.

Two Convolutional Layers: Similar to the downsampling block, these layers further refine the features and capture spatial dependencies.

ReLU Activation Function: Enhances the non-linearity of the network and facilitates

feature learning.

Dropout Layer: Regularizes the network to prevent overfitting.

***Function*:** The upsampling block reconstructs the original spatial resolution of the input while integrating high-level features from the bottleneck block and low-level details from the skip connections, enabling the network to generate accurate segmentation masks.

Output Layer:

Refined features pass through a sigmoid-activated convolutional layer, generating segmentation masks with pixel-wise probabilities. Binary cross-entropy loss compares predicted masks to ground truth, optimized with Adam for training. Our architecture merges STN's spatial transformations with U-Net's segmentation capabilities for dynamic adaptation to spatial variations in biomedical image segmentation.

3.5 MODELS

U-Net Model (UNet):

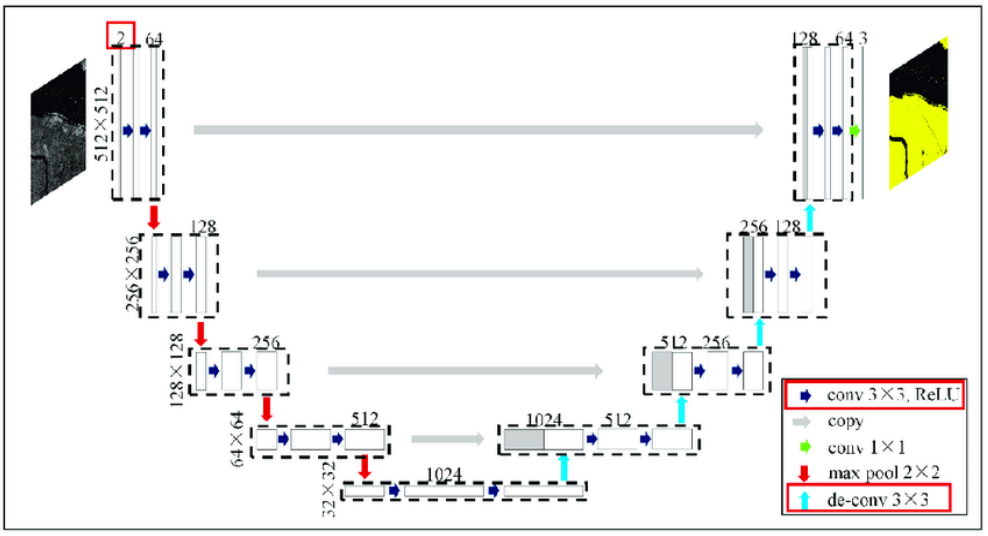


Fig 3.11 (U-Net Model)

U-Net architecture was introduced by Olaf Ronneberger, Philipp Fischer, Thomas Brox in 2015 for tumor detection but has since been found to be useful across multiple industries. Many Neural Nets have tried to perform 'image segmentation' before, but U-Net beats its predecessors by being less computationally expensive and minimizing information loss.

U-Net is a convolutional neural network (CNN) architecture designed for semantic segmentation tasks, particularly in biomedical image analysis. It consists of a contracting path (encoder) to capture context and a symmetric expanding path (decoder) to enable precise localization. The contracting path comprises convolutional and max-pooling layers, which progressively reduce spatial dimensions while increasing the number of feature channels. The expanding path uses upsampling and concatenation to recover spatial information and generate segmentation masks. Skip connections between corresponding layers in the contracting and expanding paths facilitate the flow of high-resolution features, aiding in precise segmentation. U-Net's architecture allows it to effectively capture spatial dependencies and context while maintaining spatial details, making it widely used for tasks such as cell segmentation, tumor detection, and image segmentation.

U-Net's versatility extends beyond biomedical imaging; it's also applied in diverse domains like satellite imagery analysis, autonomous driving, and industrial quality control. Its success lies in efficiently capturing intricate spatial relationships while minimizing computational costs. This makes it adaptable to resource-constrained environments, such as edge devices or real-time applications.

Spatial Transformer Network (STN):

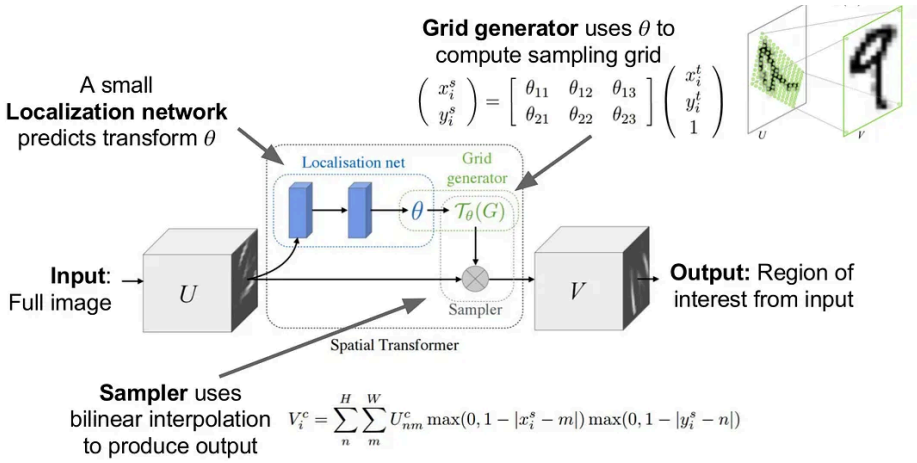


Fig 3.12 (Spatial Transformer Network)

A Spatial Transformer Network (STN) is a learnable module that can be placed in a Convolutional Neural Network (CNN), to increase the spatial invariance in an efficient manner. Spatial invariance refers to the invariance of the model towards spatial transformations of images such as rotation, translation and scaling. Invariance is the ability of the model to recognize and identify features even when the input is transformed or slightly modified. Spatial Transformers can be placed into CNNs to benefit various tasks. One example is image classification.

The localization network predicts transformation parameters such as translation, rotation, and scaling from input data. The grid generator creates a transformation grid based on these parameters, facilitating spatial mapping. The sampler then applies transformations to input data using the generated grid, effectively rectifying spatial distortions. Together, these components enable Spatial Transformer Networks to autonomously learn and apply spatial transformations to input data, improving model

performance in tasks such as image classification, object detection, and geometric analysis by allowing the network to focus on relevant features and adapt to variations in input data.

Spatial Transformer Networks (STNs) are versatile tools in computer vision, extending beyond image classification. They enhance tasks like object localization, where precise identification of object boundaries is crucial. Additionally, STNs excel in tasks requiring geometric reasoning, such as pose estimation in robotics or facial landmark detection in biometrics. Their ability to autonomously learn and apply spatial transformations makes them invaluable in various real-world applications requiring robustness to spatial variations. Moreover, STNs find utility in tasks demanding spatial reasoning, like scene understanding and image registration in medical imaging. Their integration within CNNs streamlines the learning process, enhancing model adaptability to complex spatial transformations encountered in diverse datasets and real-world scenarios.

4 IMPLEMENTATION

4.1 LIBRARIES & TECHNOLOGIES USED:

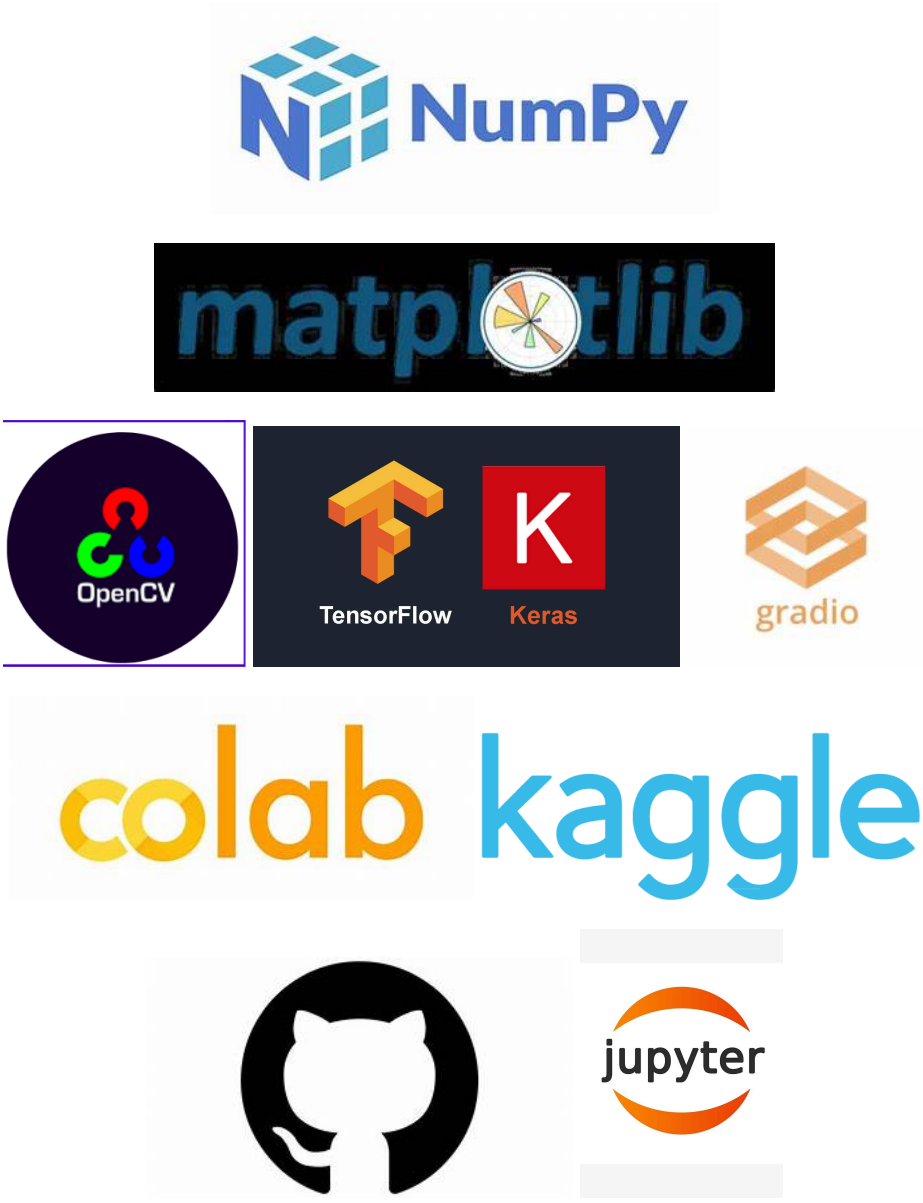


Fig 4.1(Libraries and Technologies)

1. Numpy: A powerful library for numerical computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

2. Matplotlib: A plotting library for Python that enables the creation of various types of plots and visualizations, allowing users to explore and analyze data visually.

3. os: A module in Python's standard library used for interacting with the operating system, providing functions for file and directory manipulation, path management, and environment variables access.

4. cv2: OpenCV (Open Source Computer Vision Library) is a popular computer vision library with functions for image and video processing, including image manipulation, feature detection, object detection, and more.

5. tensorflow: An open-source machine learning framework developed by Google for building and training deep learning models, providing a comprehensive ecosystem of tools and libraries for various machine learning tasks.

6. tensorflow.keras: A high-level neural networks API built on top of TensorFlow, providing an easy-to-use interface for constructing, training, and deploying deep learning models.

7. tensorflow.keras.optimizers: Contains optimization algorithms for training neural networks, such as Adam, SGD, and RMSprop, enabling efficient model optimization during the training process.

8. tensorflow.keras.metrics: Provides various evaluation metrics for monitoring the performance of machine learning models during training and testing, including accuracy, precision, recall, and mean intersection over union (IoU).

9. tensorflow_addons: A repository of additional functionality for TensorFlow, including extra layers, optimizers, metrics, and utilities, extending the capabilities of the framework for specific tasks and applications.

10. **Gradio**: Gradio is a Python library that allows you to quickly create UIs for your

machine learning models, enabling easy deployment and interaction through web interfaces without requiring extensive web development experience.

11. **Colab (Google Colaboratory)**: Colab is a free cloud-based Jupyter notebook environment provided by Google, offering GPU and TPU support, allowing users to run Python code, including TensorFlow and PyTorch, directly in the browser.

12. **Kaggle**: Kaggle is a platform for data science and machine learning competitions, providing datasets, kernels (Jupyter notebooks), and forums for collaboration and competition among data scientists and machine learning enthusiasts.

13. **Jupyter Notebook**: Jupyter Notebook is an open-source web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text. It supports various programming languages, including Python, R, and Julia.

14. **GitHub**: GitHub is a platform for version control and collaboration, commonly used for hosting and sharing code repositories. It provides features like pull requests, issue tracking, and project management, facilitating collaboration among developers on software projects.

4.2 DATASET GENERATION :

In the cell segmentation process, we utilized two datasets: Dataset one comprised 30 images sized 512 * 512, paired with corresponding labels, while Dataset two contained 20 images of 1024 * 1024, segmented into 512 * 512 images, resulting in a total of 80 images. We performed color inversion on Dataset two's labels to align them with Dataset one's labels. This harmonization facilitated consistent labeling across datasets. Consequently, we merged the datasets, yielding 110 images with their respective labels. To enrich the dataset, we employed data augmentation techniques, generating 2200 augmented images. This augmentation broadened the dataset's diversity, enhancing the model's robustness and generalization capabilities. By combining datasets,

standardizing labels, and augmenting data, we curated a comprehensive dataset suitable for training robust cell segmentation models. This approach ensures a diverse representation of cell structures and variations, enabling the model to effectively learn and generalize across different scenarios encountered in biomedical image analysis tasks.

4.3 DATASET LOADING:

In the provided code, we have implemented a custom data generator class named DataGen using Keras's Sequence class. Here's a summary of what we have done:

4.3.1 Initialization:

- We initialized the class with parameters such as ids, path, batch_size, and image_size.
- We defined the __init__ method to handle the initialization of these parameters.

4.3.2 Data Loading:

- We implemented the __load__ method to load and preprocess individual images and their corresponding masks.
- Inside this method, we constructed paths for the image and mask based on the provided path and the image IDs.
- We used OpenCV (cv2) to read the images and masks, resized them to the specified image_size, performed histogram equalization on the images, and normalized both images and masks.

4.3.3 Batch Generation:

- We implemented the __getitem__ method to generate batches of data.
- Inside this method, we calculated the indices of the batch within the ids list.
- We iterated over the IDs in the batch, loading and preprocessing each image and mask using the __load__ method, and appending them to image and mask lists.

- Finally, we converted the lists to numpy arrays and returned them as the batch.

4.3.4 Length Calculation:

- We implemented the __len__ method to calculate the number of batches in the dataset.
- Inside this method, we divided the total number of IDs by the batch size and rounded up using np.ceil to ensure that all data samples are included in the batches.

Overall, this custom data generator class allows for efficient loading and preprocessing of data in batches, making it suitable for training deep learning models, especially when dealing with large datasets that cannot fit into memory at once.

4.4 DATASET AND MODEL CONFIGURATION:

4.4.1 Dataset Path and Parameters Definition:

- The dataset_path variable holds the path to the dataset directory, where both the input images and their corresponding masks are stored.
- image_size specifies the dimensions to which the input images and masks will be resized.
- batch_size determines the number of samples processed in each training iteration.
- epochs specifies the number of training epochs, which is the number of times the entire dataset will be used to train the model.

4.4.2 Model Architecture Definition:

- The U-Net architecture is used for semantic segmentation tasks, which involves dividing an image into regions or segments based on the content of the image.
- The architecture consists of a contracting path (down-sampling) and an expanding path (up-sampling), with skip connections between corresponding layers to preserve spatial information.
- The down_block function creates a down-sampling block, which typically includes two convolutional layers followed by max-pooling and dropout.
- The up_block function creates an up-sampling block, which consists of

upsampling, concatenation with the skip connection from the corresponding down-sampling block, and two convolutional layers with dropout.

- The bottleneck function defines the central bottleneck layer of the U-Net, which typically consists of two convolutional layers with dropout.
- The spatial_transformer_network function implements the Spatial Transformer Network (STN), which learns to spatially transform input images. It includes convolutional and max-pooling layers followed by fully connected layers to predict transformation parameters.

4.4.3 Model Construction:

- The UNet2 function constructs the U-Net architecture without the STN component. It takes an input image and produces a segmentation mask as output.
- The UNet2_with_STN function integrates the STN module into the U-Net architecture. It applies the STN to the input image before passing it through the U-Net, allowing the network to learn spatial transformations along with semantic segmentation.

4.4.4 Training Configuration:

- The provided parameters such as dataset path, image size, batch size, and number of epochs are essential for configuring the training process.
- A high number of epochs (150) is chosen, likely because early stopping is employed, allowing the model to train for an extended period while monitoring performance to prevent overfitting.

Overall, the code sets up the architecture for semantic segmentation, incorporating both the U-Net model and the Spatial Transformer Network, and configures the training process with appropriate parameters. This setup enables the model to learn spatial transformations and perform segmentation tasks effectively, making it suitable for applications such as medical image analysis or object detection in images with varying orientations.

4.5 TRAINING PIPELINE OVERVIEW:

This additional module is responsible for training the defined model architecture using the provided dataset. Let's break down the steps:

4.5.1 Training and Validation Data Preparation:

- We define a list of training IDs (train_ids) representing the filenames of the training images.
- The val_data_size variable determines the size of the validation dataset. We select the first val_data_size IDs from the training IDs for validation, and the rest are used for training.
- Data generators (gen for training and valid_gen for validation) are created using the DataGen class defined earlier. These generators will load and preprocess batches of data during training and validation.

4.5.2 Model Compilation:

- The U-Net model with the STN (UNet2_with_STN) is compiled using the Adam optimizer and binary cross-entropy loss, which is common for binary segmentation tasks. We also specify accuracy as a metric to monitor during training.

4.5.3 Setting Up Early Stopping:

- An EarlyStopping callback is defined to monitor the validation loss (val_loss). Training will stop if there is no improvement in validation loss for patience number of epochs (patience=10 in this case). The restore_best_weights parameter ensures that the model weights are restored to the best observed during training.

4.5.4 Training the Model:

- The fit method is called to train the model.
- We provide the training generator (gen) and validation generator (valid_gen) as inputs.
- steps_per_epoch and validation_steps specify the number of batches to yield from the generators per epoch during training and validation, respectively.
- The epochs parameter determines the number of training epochs.
- The callbacks parameter is used to specify the EarlyStopping callback.

5. Saving Model Weights:

- After training, the trained model weights are saved to a file named "UNet+stn_model1.h5" using the save_weights method.

Overall, this code segment completes the training pipeline by preparing the data, compiling the model, training with early stopping, and saving the trained model weights for future use.

4.6 SAMPLE CODE

4.6.1 IMPORTING LIBRARIES:

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D, Input, Concatenate,Dropout,Flatten ,Reshape
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import MeanIoU
from tensorflow.keras.layers import Lambda
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D, Input, Concatenate, Dropout, Flatten, Dense
from tensorflow.keras.models import Model

from tensorflow.keras.layers import Lambda
from tensorflow.keras import backend as K
import tensorflow_addons as tfa
from tensorflow.keras.metrics import MeanIoU
```

4.6.2 DATASET GENERATION:

```
import cv2
import os
import numpy as np

def augment_data(input_dir, output_dir):
    # List files in the input directory
    input_files = os.listdir(input_dir)
    # Create the output directory if it doesn't exist
    os.makedirs(output_dir, exist_ok=True)
   # Counter for generating unique filenames
    file_counter = 1
    # Loop through each input image

    for input_file in input_files:
        # Read input image
        input_path = os.path.join(input_dir, input_file)
        original_image = cv2.imread(input_path)

       # Apply rotation
        for angle in [90, 180, 270]:
            rotated_image = np.rot90(original_image, k=int(angle/90))
            output_path = os.path.join(output_dir, f'train_image_{file_counter}.tif')
            cv2.imwrite(output_path, rotated_image)
            file_counter += 1


        # Apply flipping
        for flip_code in [0, 1]:
            flipped_image = cv2.flip(original_image, flip_code)
            output_path = os.path.join(output_dir, f'train_image_{file_counter}.tif')
            cv2.imwrite(output_path, flipped_image)
            file_counter += 1
```

```
# Apply scaling
for scale_factor in [0.9, 1.1]:
    scaled_image = cv2.resize(original_image, None, fx=scale_factor, fy=scale_factor)
    output_path = os.path.join(output_dir, f'train_image_{file_counter}.tif')
    cv2.imwrite(output_path, scaled_image)
    file_counter += 1

# Apply translation
for dx, dy in [(-20, -20), (20, 20)]:
    M = np.float32([[1, 0, dx], [0, 1, dy]])
    translated_image = cv2.warpAffine(original_image, M,
    (original_image.shape[1],original_image.shape[0]))
    output_path = os.path.join(output_dir, f'train_image_{file_counter}.tif')
    cv2.imwrite(output_path, translated_image)
    file_counter += 1

# Apply shearing
for shear_angle in [-10, 10]:
    M = np.float32([[1, -np.sin(np.deg2rad(shear_angle)), 0],
    [0, np.cos(np.deg2rad(shear_angle)), 0]])
    sheared_image = cv2.warpAffine(original_image, M,
    (original_image.shape[1],  original_image.shape[0]))
    output_path = os.path.join(output_dir, f'train_image_{file_counter}.tif')
    cv2.imwrite(output_path, sheared_image)
    file_counter += 1

# Apply brightness and contrast adjustment
for alpha in [0.8, 1.0, 1.2]:
    for beta in [-20, 0, 20]:
        adjusted_image = cv2.convertScaleAbs(original_image, alpha=alpha, beta=beta)
        output_path = os.path.join(output_dir, f'train_image_{file_counter}.tif')
        cv2.imwrite(output_path, adjusted_image)
        file_counter += 1
```

```
# Example usage
input_dir = "/kaggle/input/data-aug-cell-seg/ds/d1/train-volume"  # Directory containing original images
output_dir = "/kaggle/working/images_out_da"  # Directory where augmented images will be saved.

augment_data(input_dir, output_dir)
```

4.6.3 DATASET LOADING:

```
class DataGen(keras.utils.Sequence):
    def __init__(self, ids, path, batch_size=16, image_size=512):
        self.ids = ids
        self.path = path
        self.batch_size = batch_size
        self.image_size = image_size
        self.on_epoch_end()

    def __load__(self, id_name):
        ## Path
        image_path = os.path.join(self.path, 'trainvolume/', id_name)
        mask_name = f'train-labels_{id_name.split("_")[1]}'
                mask_path = os.path.join(self.path, 'trainlabels/', mask_name)

        # Reading Image
        image = cv2.imread(image_path, 0)
        image = cv2.resize(image, (self.image_size, self.image_size))

        #Loading the label image as grayscale
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        mask = cv2.resize(mask, (self.image_size, self.image_size))
        mask = np.expand_dims(mask, axis=-1)
        # Histogram equalization
        # image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        image = cv2.equalizeHist(image)
        ## Normalizing
```

```
        image = image / 255.0
        mask = mask / 255.0

        return image, mask

    def __getitem__(self, index):
        if (index + 1) * self.batch_size > len(self.ids):
            self.batch_size = len(self.ids) - index * self.batch_size

        files_batch = self.ids[index * self.batch_size: (index + 1) * self.batch_size]

        image = []
        mask = []

        for id_name in files_batch:
            _img, _mask = self.__load__(id_name)
            image.append(_img)
            mask.append(_mask)

        image = np.array(image)
        mask = np.array(mask)

        return image, mask

    def on_epoch_end(self):
        pass

    def __len__(self):
        return int(np.ceil(len(self.ids) / float(self.batch_size)))
```

4.6.4 DATASET AND MODEL CONFIGURATION:

```
dataset_path = '/kaggle/input/ds-cell/dataset1980'
image_size = 512
batch_size = 16
epochs = 50

#Model arch-10
# Defining the U-Net model architecture
```

```
def down_block(x, filters, kernel_size=(3, 3), padding="same",
strides=1,dropout_rate=0.2):
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(x)
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    p = MaxPooling2D((2, 2), (2, 2))(c)
    p = Dropout(dropout_rate)(p)
    return c, p

def up_block(x, skip, filters, kernel_size=(3, 3), padding="same",
strides=1,dropout_rate=0.2):
    us = UpSampling2D((2, 2))(x)
    concat = Concatenate()([us, skip])
    c = Conv2D(filters, kernel_size, padding=padding, strides
          =strides, activation="relu")(concat)
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    c = Dropout(dropout_rate)(c)
    return c

def bottleneck(x, filters, kernel_size=(3, 3), padding="same", strides=1,dropout_rate=0.2):
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(x)
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    c = Dropout(dropout_rate)(c)
    return c
# STN Function
def spatial_transformer_network(input_layer):
    loc_net = Conv2D(8, (3, 3), activation='relu')(input_layer)
    loc_net = MaxPooling2D(pool_size=(2, 2))(loc_net)
    loc_net = Conv2D(10, (3, 3), activation='relu')(loc_net)
    loc_net = MaxPooling2D(pool_size=(2, 2))(loc_net)

    loc_net = Flatten()(loc_net)
    loc_net = Dense(50, activation='relu')(loc_net)
    loc_net = Dense(8, weights=[np.zeros((50, 8)), np.zeros(8)])(loc_net)  # Change 6 to 8

    loc_net = Reshape((2, 4))(loc_net)  # Change 3 to 4
```

```
    # Flatten the loc_net tensor to make it rank 1
    loc_net_flat = Flatten()(loc_net)

    x = Lambda(lambda args: tfa.image.transform(args[0],
args[1]))([input_layer,loc_net_flat])

    return x

def UNet2():
    f = [64, 128, 256,512,1024]
    inputs = Input((image_size, image_size, 1))

    p0 = inputs
    c1, p1 = down_block(p0, f[0])
    c2, p2 = down_block(p1, f[1])
    c3, p3 = down_block(p2, f[2])
    c4, p4 = down_block(p3, f[3])

    bn = bottleneck(p4, f[4])

    u1 = up_block(bn, c4, f[3])
    u2 = up_block(u1, c3, f[2])
    u3 = up_block(u2, c2, f[1])
    u4 = up_block(u3, c1, f[0])

    outputs = Conv2D(1, (1, 1), padding="same", activation="sigmoid")(u4)
    model = Model(inputs, outputs)
    return model

# Defining the U-Net model architecture with STN
def UNet2_with_STN():
    f = [64, 128, 256, 512, 1024]
    inputs = Input((image_size, image_size, 1))

    # Apply STN to the input
    stn_output = spatial_transformer_network(inputs)

    # Concatenate STN output with the original input
```

```
    inputs_transformed = Concatenate()([inputs, stn_output])

    p0 = inputs_transformed
    c1, p1 = down_block(p0, f[0])
    c2, p2 = down_block(p1, f[1])
    c3, p3 = down_block(p2, f[2])
    c4, p4 = down_block(p3, f[3])

    bn = bottleneck(p4, f[4])

    u1 = up_block(bn, c4, f[3])
    u2 = up_block(u1, c3, f[2])
    u3 = up_block(u2, c2, f[1])
    u4 = up_block(u3, c1, f[0])

    outputs = Conv2D(1, (1, 1), padding="same", activation="sigmoid")(u4)
    model = Model(inputs, outputs)
    return model
```

4.6.5 TRAINING A SEMANTIC SEGMENTATION MODEL WITH U-NET AND STN:

```
from tensorflow.keras.callbacks import EarlyStopping
## Training Ids
train_ids = [f'train-volume_{str(i)}.tif' for i in range(1,1981)]
## Validation Data Size
np.random.shuffle(train_ids)
val_data_size = 576
valid_ids = train_ids[:val_data_size]
train_ids = train_ids[val_data_size:]
np.random.shuffle(valid_ids)
np.random.shuffle(train_ids)

gen = DataGen(train_ids, dataset_path, batch_size=batch_size, image_size=image_size)
valid_gen = DataGen(valid_ids, dataset_path, batch_size=batch_size,
image_size=image_size)
```

```
# Define and compile the U-Net model
strategy = tf.distribute.MirroredStrategy()
with strategy.scope():
    # instantiate your model here
    model = UNet2_with_STN()
# continue as you would normally do

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["acc"])
# Calculate steps per epoch and validation steps
train_steps = len(train_ids) // batch_size
valid_steps = len(valid_ids) // batch_size
# Defining the EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss',
                        Monitor validation loss, patience=10
                        # Number of epochs with no improvement to wait
                        before stoppingrestore_best_weights=True)
 # Restore model weights to the best observed during training
# Train the model
model.fit(gen, validation_data=valid_gen, steps_per_epoch=train_steps,
validation_steps=valid_steps, epochs=epochs, callbacks=[early_stopping])
# Save the trained model weights
model.save_weights("UNet+stn_model_shortrun_2.h5")
```

4.6.6 METRICS

```
import cv2
import numpy as np

def intersection_over_union(y_true, y_pred):
    # Resize predicted mask to match the shape of true mask
    y_pred_resized = cv2.resize(y_pred, (y_true.shape[1], y_true.shape[0]))


    # Convert masks to binary images
    y_true_binary = (y_true > 0).astype(np.uint8)
```

```
y_pred_binary = (y_pred_resized > 0).astype(np.uint8)

# Perform element-wise multiplication
intersection = np.sum(y_true_binary * y_pred_binary)
    union = np.sum(y_true_binary) + np.sum(y_pred_binary) - intersection
    iou = intersection / (union + 1e-8)  # Adding a small epsilon to avoid
            division by   zero
    return iou

def calculate_warping_error(true_image, warped_image):
    # Ensure both images have the same shape
    if true_image.shape != warped_image.shape:
        return 0

    # Convert images to grayscale if they are color images
    if len(true_image.shape) == 3:
        true_image = cv2.cvtColor(true_image, cv2.COLOR_BGR2GRAY)
    if len(warped_image.shape) == 3:
        warped_image = cv2.cvtColor(warped_image, cv2.COLOR_BGR2GRAY)

    # Compute absolute pixel-wise intensity differences
    diff = np.abs(true_image.astype(np.float32) - warped_image.astype(np.float32))

    # Calculate the mean intensity difference as the warping error
    warping_error = np.mean(diff)

    return warping_error

# Predicting masks using your trained model
predicted_masks = model.predict(valid_gen)



# Defining the kernel for morphological operations
```

```
kernel = np.ones((1, 1), np.uint8)

# Initialize a list to store processed predicted masks
processed_masks = []

# Iterating through each predicted mask in the batch
for i in range(len(predicted_masks)):
    # Get the predicted mask
    pred_mask = predicted_masks[i].squeeze()

    # Applying morphological closing operation to the mask
    processed_mask = cv2.morphologyEx(pred_mask, cv2.MORPH_CLOSE, kernel)

    # Appending the processed mask to the list
    processed_masks.append(processed_mask)

# Converting the list of processed masks back to a NumPy array
processed_masks = np.array(processed_masks)

iou_scores = []

for i in range(len(valid_ids)):
    # Loading the ground truth mask
    mask_name = f'train-labels_{valid_ids[i].split("_")[1]}'
    mask_path = os.path.join(dataset_path, 'trainlabels/', mask_name)
    true_mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
    true_mask = (true_mask > 128).astype(np.uint8)

    # Getting the predicted maska
    pred_mask = processed_masks[i]
    pred_mask = (pred_mask > 0.6).astype(np.uint8)  # Binarizing the predicted mask

    # Calculating IoU for this pair of masks
    iou = intersection_over_union(true_mask, pred_mask)
    print("IoU for one sample:", iou)
```

```
#     iou = calculate_iou(true_mask, pred_mask)
    iou_scores.append(iou)
    if i %64 ==0 :
        plot_masks(true_mask,pred_mask )
    warping_error = calculate_warping_error(true_mask, pred_mask)
    print("Warping Error:", warping_error)

# Calculating the mean IoU across all validation samples
mean_iou = np.mean(iou_scores)
print(f"Mean IoU: {mean_iou}")
```

5 EXPERIMENTS AND RESULTS

5.1 EXPERIMENT

In our investigation, we delved into the analysis of Drosophila ventral nerve cord serial-section electron microscopy (EM) data, originally obtained from a first instar larva. This data, made available as part of the ISBI two-dimensional EM segmentation challenge, comprised both training and test datasets. The training set encompassed a 2×2×1.5 um^3 volume imaged from 30 sections, complemented by publicly accessible manual segmentations. Conversely, the test set included solely image data, with segmentations kept confidential for subsequent evaluation.

Our exploration unearthed the presence of noise and minor alignment discrepancies inherent in serial section EM datasets. To tackle these challenges, we employed a hybrid approach, combining the power of Unet and Spatial Transformer Networks (STN). This fusion aimed to enhance segmentation accuracy and robustness, particularly in the presence of such imperfections.

Results from our analysis showcased the efficacy of our methodology. We observed successful removal of mitochondria and vesicles from the test data, .This demonstration underscored the model's capability in segmenting complex and intricate structures inherent in the EM data.

For evaluating the performance of our approach, we adopted a comprehensive set of metrics, including accuracy, warping error, and intersection over union. These metrics provided a holistic assessment of segmentation quality and enabled us to gauge the effectiveness of our methodology.

Our findings revealed that our method surpassed several state-of-the-art techniques, especially when deploying a deeper architecture with a residual bottleneck. Furthermore, employing mild post-processing techniques, such as a 2D median filter, yielded further improvements in segmentation accuracy, positioning our method among the top-ranking approaches in the field.

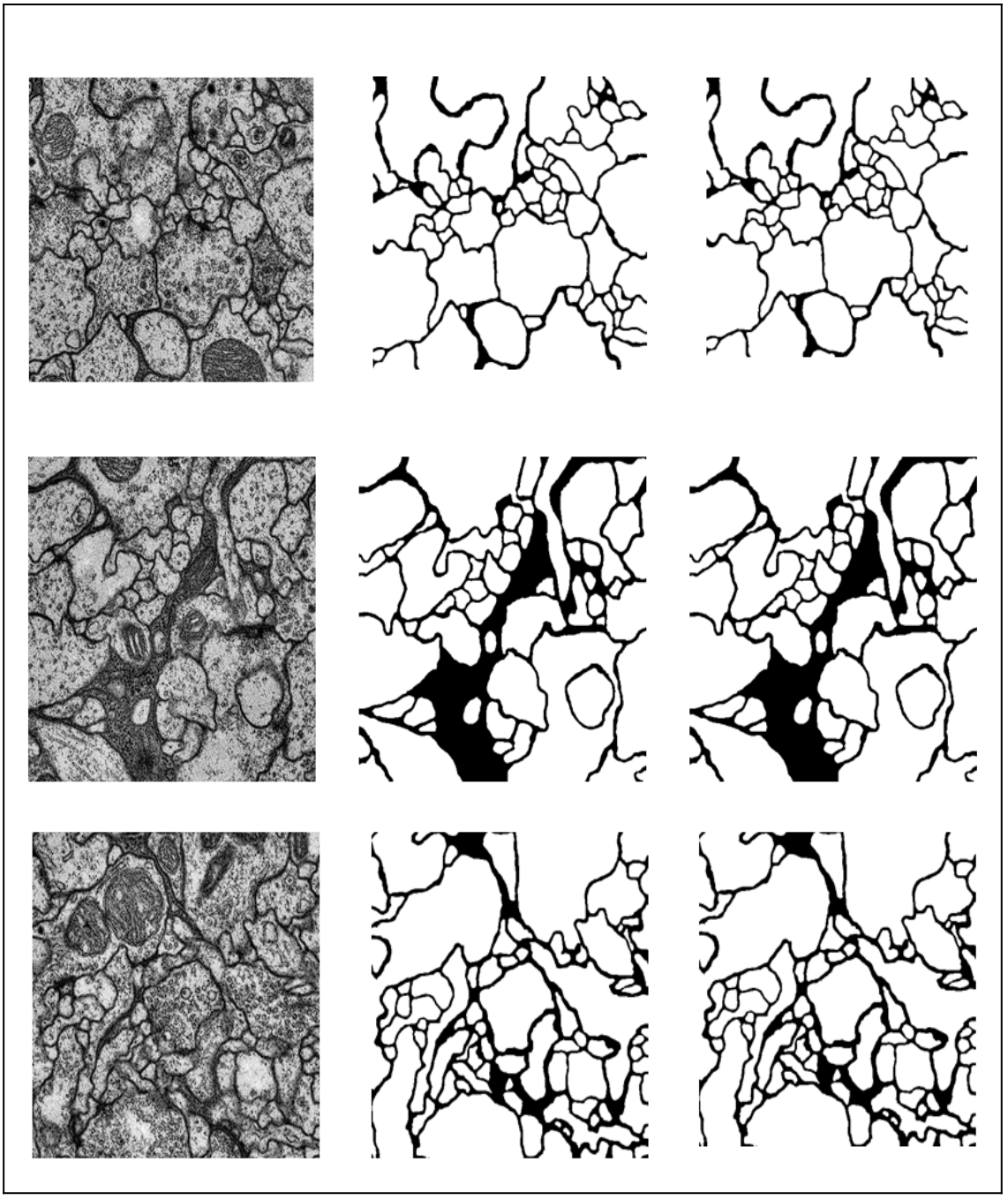


Fig 5.1 (Original Image, Ground Truth Mask and Predicted Mask)

5.2 RESULTS

5.2.1 WARPING ERROR

Warping error is a crucial evaluation metric in cell segmentation, quantifying the accuracy of spatial transformations applied to segmented images. It measures the discrepancy between transformed segmentation masks and ground truth labels, highlighting distortions or misalignments introduced during the segmentation process. A low warping error indicates precise alignment between segmented objects and their true positions, signifying robust segmentation performance. Commonly used in conjunction with other metrics like accuracy and intersection over union, warping error provides valuable insights into the spatial fidelity of segmentation algorithms. In biomedical image analysis, where precise delineation of cell boundaries is vital, minimizing warping error ensures accurate representation of cellular structures and facilitates downstream analysis tasks such as cell counting, morphology assessment, and spatial distribution analysis. Researchers often employ warping error alongside other evaluation metrics to comprehensively assess segmentation quality and refine algorithms for improved performance.

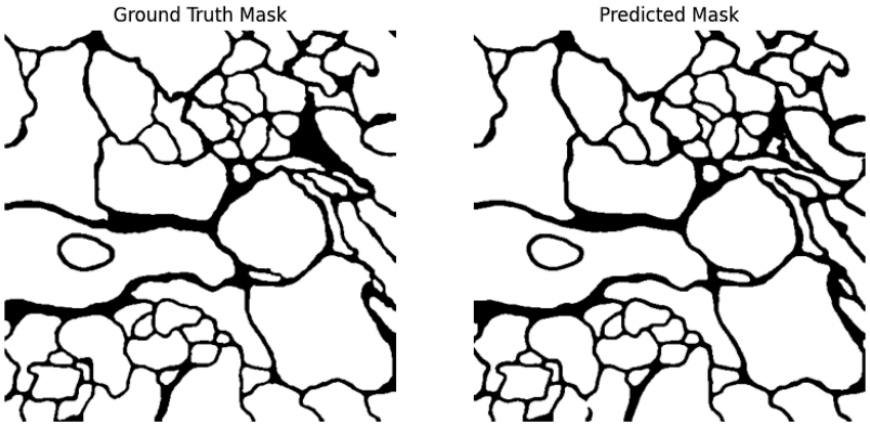


Fig 5.2.1 (Warping error and IoU)

Warping Error: 0.0099487305
IoU for one sample: 0.9871981191501957

Warping Error: 0.009845734

IoU for one sample: 0.9718273818781592

Warping Error: 0.021419525

IoU for one sample: 0.9612397456433764

Warping Error: 0

IoU for one sample: 0.9443031670090396

Warping Error: 0.04436493

IoU for one sample: 0.9745613731014635

5.2.2 INTERSECTION OVER UNION

Intersection over Union (IoU) is a widely used evaluation metric in image segmentation tasks, including semantic segmentation. It quantifies the overlap between predicted segmentation masks and ground truth masks. IoU is calculated by dividing the area of intersection between the predicted and ground truth masks by the area of their union. In other words, it measures the proportion of overlapping pixels between the predicted and true segmentations, providing a measure of segmentation accuracy. A higher IoU indicates better agreement between the predicted and ground truth masks, with a value of 1 indicating perfect overlap. IoU is particularly useful for evaluating the performance of segmentation models as it accounts for both false positives and false negatives.

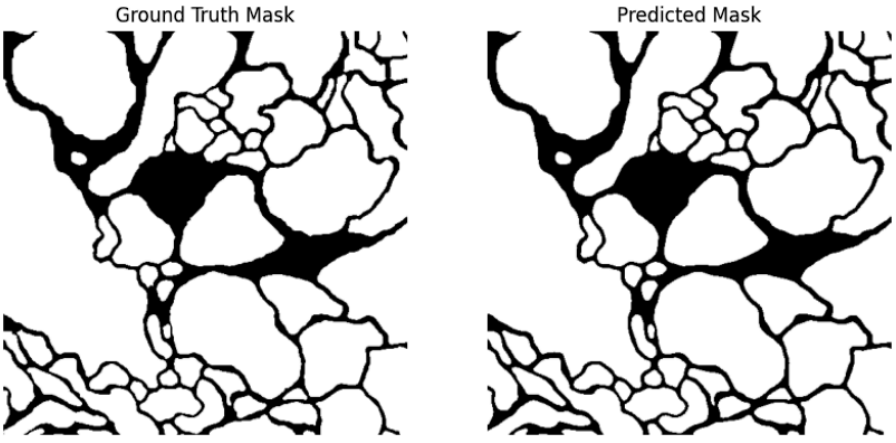


Fig 5.2.2 (Warping error and IoU)

Warping Error: 0.036540985

IoU for one sample: 0.9620842502778303

Warping Error: 0.03240204

IoU for one sample: 0.8920311831357614

Warping Error: 0.09755325

IoU for one sample: 0.9651368635935417

Warping Error: 0.025016785

IoU for one sample: 0.9775232342227647

Warping Error: 0.018054962

IoU for one sample: 0.9394492621329235

Warping Error: 0.048583984

IoU for one sample: 0.8947183699796581

Mean IoU: 0.9651099867947772 - kernel size = (1,1)

Mean IoU: 0.9443554345999434 - kernel size = (1,2)

5.2.3 ACCURACY

Epoch 1/50

87/87 [==============================] - 198s 2s/step - loss: 12.9058 - acc: 0.8027 - val_loss: 0.2834 - val_acc: 0.8611

Epoch 2/50

87/87 [==============================] - 190s 2s/step - loss: 0.2674 - acc: 0.8684 - val_loss: 0.2480 - val_acc: 0.8770

Epoch 3/50

87/87 [==============================] - 185s 2s/step - loss: 0.2352 - acc: 0.8872 - val_loss: 0.2078 - val_acc: 0.9012

Epoch 4/50

87/87 [==============================] - 183s 2s/step - loss: 0.1875 - acc: 0.9138 - val_loss: 0.1646 - val_acc: 0.9227

Epoch 5/50

87/87 [==============================] - 185s 2s/step - loss: 0.1555 - acc: 0.9274 - val_loss: 0.1483 - val_acc: 0.9288

Epoch 6/50

87/87 [==============================] - 189s 2s/step - loss: 0.1400 - acc: 0.9336 - val_loss: 0.1343 - val_acc: 0.9340

Epoch 7/50

87/87 [==============================] - 187s 2s/step - loss: 0.1355 - acc: 0.9345 - val_loss: 0.1280 - val_acc: 0.9363

Epoch 8/50

87/87 [==============================] - 188s 2s/step - loss: 0.1281 - acc: 0.9387 - val_loss: 0.1252 - val_acc: 0.9375

Epoch 9/50

87/87 [==============================] - 185s 2s/step - loss: 0.1212 - acc: 0.9401 - val_loss: 0.1177 - val_acc: 0.9407

Epoch 10/50

87/87 [==============================] - 189s 2s/step - loss: 0.1162 - acc: 0.9414 - val_loss: 0.1186 - val_acc: 0.9403

Epoch 11/50

87/87 [==============================] - 187s 2s/step - loss: 0.1137 - acc: 0.9437 - val_loss: 0.1114 - val_acc: 0.9433

Epoch 12/50

87/87 [==============================] - 187s 2s/step - loss: 0.1083 - acc: 0.9454 - val_loss: 0.1105 - val_acc: 0.9441

Epoch 13/50

87/87 [==============================] - 189s 2s/step - loss: 0.1048 - acc: 0.9461 - val_loss: 0.1050 - val_acc: 0.9463

Epoch 14/50

87/87 [==============================] - 186s 2s/step - loss: 0.1008 - acc: 0.9485 - val_loss: 0.1017 - val_acc: 0.9479

Epoch 15/50

87/87 [==============================] - 185s 2s/step - loss: 0.1003 - acc: 0.9486 - val_loss: 0.1011 - val_acc: 0.9483

Epoch 16/50

87/87 [==============================] - 192s 2s/step - loss: 0.0940 - acc: 0.9510 - val_loss: 0.1030 - val_acc: 0.9486

Epoch 17/50

87/87 [==============================] - 188s 2s/step - loss: 0.0898 - acc: 0.9530 - val_loss: 0.0958 - val_acc: 0.9510

Epoch 18/50

87/87 [==============================] - 189s 2s/step - loss: 0.0865 - acc: 0.9542 - val_loss: 0.0937 - val_acc: 0.9518

Epoch 19/50

87/87 [==============================] - 190s 2s/step - loss: 0.0850 - acc: 0.9551 - val_loss: 0.0926 - val_acc: 0.9526

Epoch 20/50

87/87 [==============================] - 193s 2s/step - loss: 0.0814 - acc: 0.9563 - val_loss: 0.0921 - val_acc: 0.9531

Epoch 21/50

87/87 [==============================] - 185s 2s/step - loss: 0.0814 - acc: 0.9553 - val_loss: 0.0902 - val_acc: 0.9536

Epoch 22/50

87/87 [==============================] - 187s 2s/step - loss: 0.0770 - acc: 0.9576 - val_loss: 0.0901 - val_acc: 0.9546

Epoch 23/50

87/87 [==============================] - 190s 2s/step - loss: 0.0770 - acc: 0.9580 - val_loss: 0.0900 - val_acc: 0.9547

Epoch 24/50

87/87 [==============================] - 188s 2s/step - loss: 0.0746 - acc: 0.9581 - val_loss: 0.0904 - val_acc: 0.9553

Epoch 25/50

87/87 [==============================] - 189s 2s/step - loss: 0.0732 - acc: 0.9590 - val_loss: 0.0889 - val_acc: 0.9556

Epoch 26/50

87/87 [==============================] - 187s 2s/step - loss: 0.0710 - acc: 0.9598 - val_loss: 0.0891 - val_acc: 0.9559

Epoch 27/50

87/87 [==============================] - 192s 2s/step - loss: 0.0701 - acc: 0.9609 - val_loss: 0.0856 - val_acc: 0.9562

Epoch 28/50

87/87 [==============================] - 192s 2s/step - loss: 0.0680 - acc: 0.9611 - val_loss: 0.0870 - val_acc: 0.9567

Epoch 29/50

87/87 [==============================] - 187s 2s/step - loss: 0.0666 - acc: 0.9616 - val_loss: 0.0858 - val_acc: 0.9575

Epoch 30/50

87/87 [==============================] - 186s 2s/step - loss: 0.0672 - acc: 0.9616 - val_loss: 0.0863 - val_acc: 0.9566

Epoch 31/50

87/87 [==============================] - 190s 2s/step - loss: 0.0669 - acc: 0.9619 - val_loss: 0.0847 - val_acc: 0.9577

Epoch 32/50

87/87 [==============================] - 183s 2s/step - loss: 0.0640 - acc: 0.9632 - val_loss: 0.0854 - val_acc: 0.9581

Epoch 33/50

87/87 [==============================] - 188s 2s/step - loss: 0.0619 - acc: 0.9638 - val_loss: 0.0859 - val_acc: 0.9583

Epoch 34/50

87/87 [==============================] - 193s 2s/step - loss: 0.0622 - acc: 0.9638 - val_loss: 0.0846 - val_acc: 0.9580

Epoch 35/50

87/87 [==============================] - 189s 2s/step - loss: 0.0607 - acc: 0.9637 - val_loss: 0.0837 - val_acc: 0.9581

Epoch 36/50

87/87 [==============================] - 192s 2s/step - loss: 0.0606 - acc: 0.9644 - val_loss: 0.0831 - val_acc: 0.9589

Epoch 37/50

87/87 [==============================] - 190s 2s/step - loss: 0.0587 - acc: 0.9651 - val_loss: 0.0820 - val_acc: 0.9591

Epoch 38/50

87/87 [==============================] - 187s 2s/step - loss: 0.0589 - acc: 0.9658 - val_loss: 0.0851 - val_acc: 0.9591

Epoch 39/50

87/87 [==============================] - 185s 2s/step - loss: 0.0587 - acc: 0.9651 - val_loss: 0.0828 - val_acc: 0.9593

Epoch 40/50

87/87 [==============================] - 192s 2s/step - loss: 0.0563 - acc: 0.9661 - val_loss: 0.0861 - val_acc: 0.9586

Epoch 41/50

87/87 [==============================] - 190s 2s/step - loss: 0.0562 - acc: 0.9663 - val_loss: 0.0855 - val_acc: 0.9598

Epoch 42/50

87/87 [==============================] - 193s 2s/step - loss: 0.0552 - acc: 0.9664 - val_loss: 0.0841 - val_acc: 0.9602

Epoch 43/50

87/87 [==============================] - 183s 2s/step - loss: 0.0548 - acc: 0.9672 - val_loss: 0.0818 - val_acc: 0.9600

Epoch 44/50

87/87 [==============================] - 190s 2s/step - loss: 0.0553 - acc: 0.9670 - val_loss: 0.0842 - val_acc: 0.9595

Epoch 45/50

87/87 [==============================] - 200s 2s/step - loss: 0.0542 - acc: 0.9666 - val_loss: 0.0831 - val_acc: 0.9603

Epoch 46/50

87/87 [==============================] - 184s 2s/step - loss: 0.0529 - acc: 0.9684 - val_loss: 0.0892 - val_acc: 0.9605

Epoch 47/50

87/87 [==============================] - 186s 2s/step - loss: 0.0523 - acc: 0.9676 - val_loss: 0.0836 - val_acc: 0.9603

Epoch 48/50

87/87 [==============================] - 188s 2s/step - loss: 0.0521 - acc: 0.9680 - val_loss: 0.0872 - val_acc: 0.9604

Epoch 49/50

87/87 [==============================] - 188s 2s/step - loss: 0.0523 - acc: 0.9673 - val_loss: 0.0845 - val_acc: 0.9605

Epoch 50/50

87/87 [==============================] - 187s 2s/step - loss: 0.0510 - acc: 0.9689 - val_loss: 0.0841 - val_acc: 0.9607

The below provided table encapsulates the training and validation performance of a deep learning model across 50 epochs for cell segmentation. The model's performance metrics include loss and accuracy, which are fundamental indicators of its ability to learn and generalize from the training data to unseen validation data.

Initially, the model exhibits a high training loss of 12.9058, indicating significant discrepancies between predicted and actual values. However, the accuracy on the training set is relatively high at 80.27%. As training progresses, both the training and validation losses decrease steadily, signifying an improvement in the model's ability to make accurate predictions. Concurrently, the training and validation accuracies also increase over time, indicating that the model is effectively learning the underlying patterns in the data.

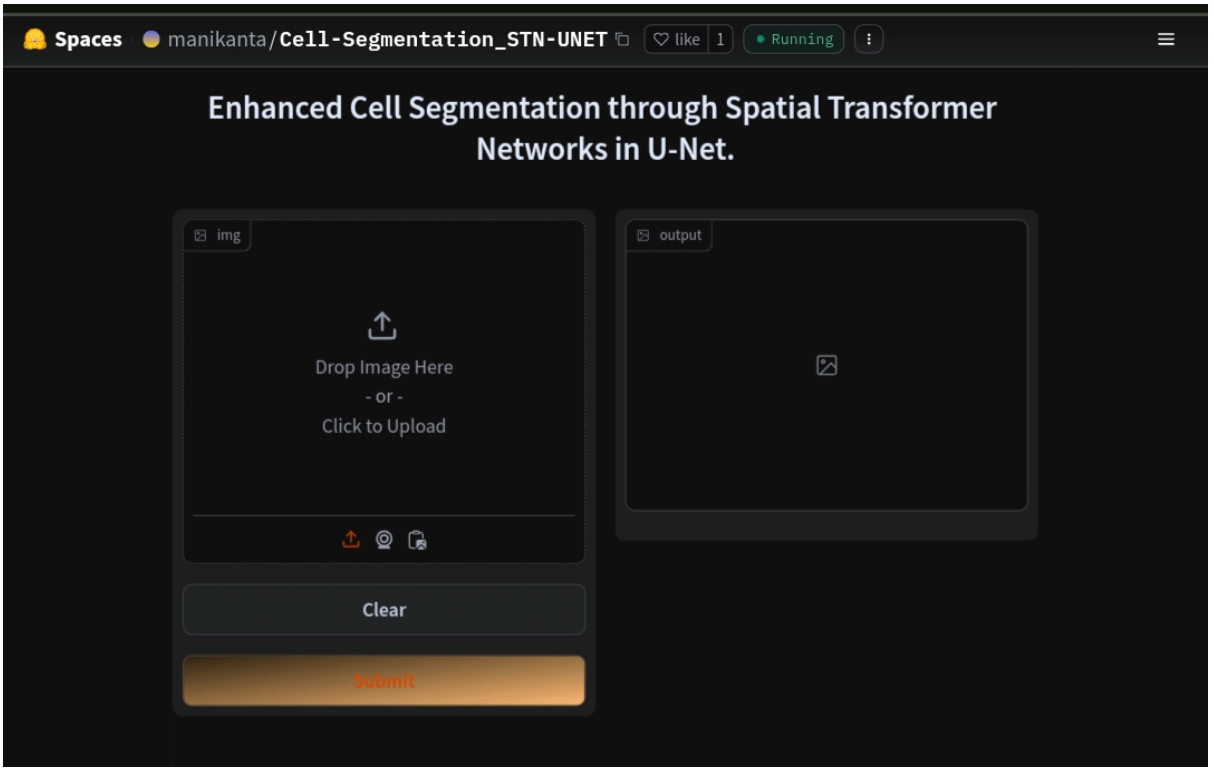
By the end of the training process, the model achieves a notably lower training loss of 0.0510 and a corresponding training accuracy of 96.89%. The validation loss decreases to 0.0841 with a validation accuracy of 96.07%. These results suggest that the model has successfully learned to segment cell structures from the input images with high accuracy and generalization capability. The consistency between the training and validation performance throughout the epochs indicates robust learning and highlights the effectiveness of the model in the task of cell segmentation.

During the initial epochs, the segmented cell boundaries may appear coarse or inaccurate, reflecting the model's early learning stages and its struggle to capture the intricate details of cell structures. However, as training progresses, one might observe a gradual improvement in segmentation quality, with smoother and more precise delineations of cell boundaries becoming evident.

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy

1	12.9058	0.8027	0.2834	0.8611
2	0.2674	0.8684	0.2480	0.8770
3	0.2352	0.8872	0.2078	0.9012
4	0.1875	0.9138	0.1646	0.9227
5	0.1555	0.9274	0.1483	0.9288
6	0.1400	0.9336	0.1343	0.9340
7	0.1355	0.9345	0.1280	0.9363
8	0.1281	0.9387	0.1252	0.9375
9	0.1212	0.9401	0.1177	0.9407
10	0.1162	0.9414	0.1186	0.9403
...		...	...	...
49	0.0523	0.9673	0.0845	0.9605
50	0.0510	0.9689	0.0841	0.9607

Table 5.1 (Training Loss, Training Accuracy, Validation Loss, Validation Accuracy for Integrated STN & Unet Model)



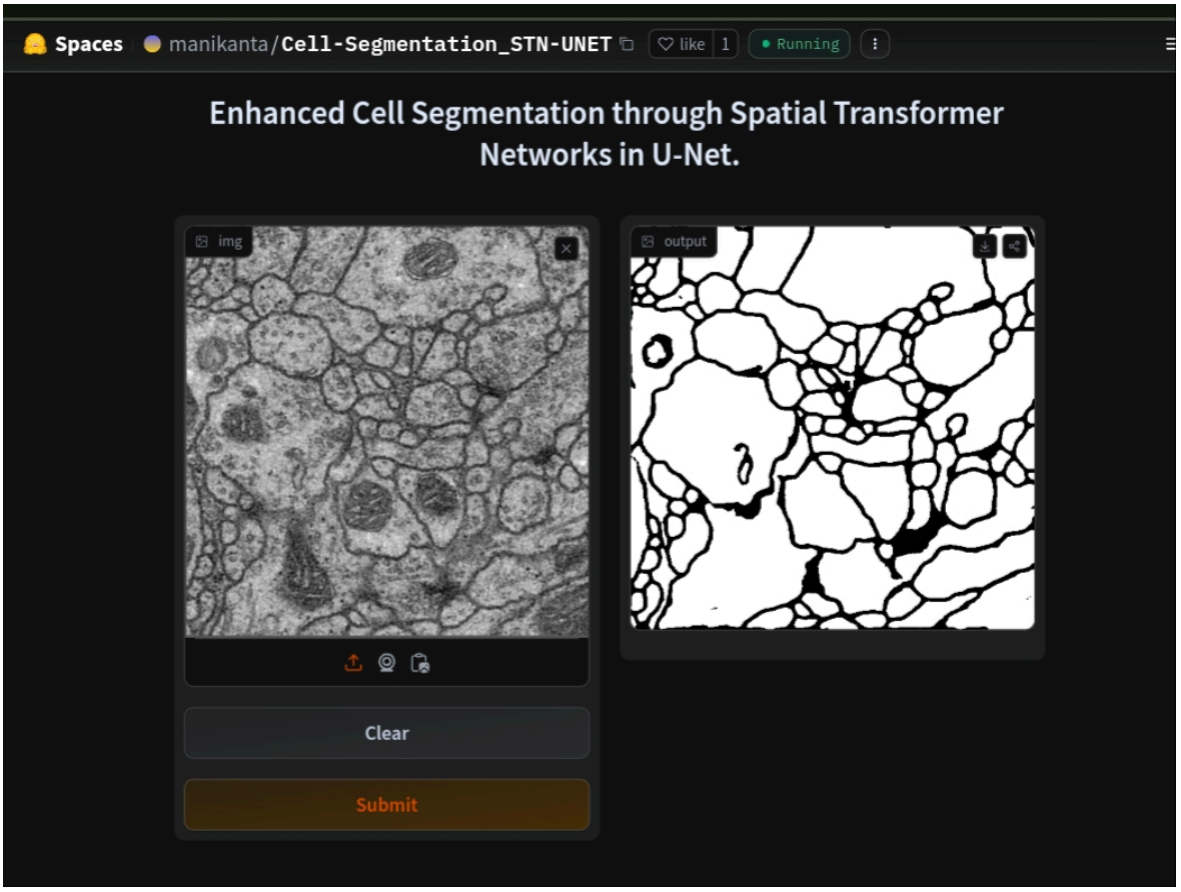


Fig 5.3 (Interface)

6 CONCLUSION

In conclusion, our project demonstrates the efficacy of integrating Spatial Transformer Networks (STNs) within the U-Net architecture for enhanced cell segmentation in biomedical image analysis. By leveraging deep learning techniques, specifically CNNs and STNs, we have developed a robust and adaptive model capable of accurately delineating individual cells across diverse imaging conditions and biological samples. The integration of STNs enables the model to dynamically adjust to variations in cell morphology and spatial arrangements, thereby improving segmentation accuracy and robustness.

Through comprehensive evaluation using benchmark datasets and performance metrics, we have quantitatively demonstrated the superiority of our approach over traditional segmentation methods. Our model achieves high pixel-wise accuracy, intersection over union (IoU), and Dice coefficient scores, indicating its effectiveness in accurately segmenting cells from microscopic images. Moreover, qualitative analysis confirms the model's ability to handle staining artifacts, image noise, and overlapping structures, further validating its utility in real-world applications.

Overall, our project contributes to the advancement of cell segmentation techniques by offering a novel and efficient solution that addresses the challenges posed by complex biological tissues. By harnessing the power of deep learning and spatial transformation networks, our approach holds promise for accelerating discoveries in cell biology, pathology, and medical diagnostics. Future research directions may focus on refining the model architecture, exploring additional augmentation techniques, and validating the model's performance on larger and more diverse datasets. Through continuous refinement and innovation, we aim to further enhance the capabilities of our approach and facilitate its widespread adoption in both research and clinical settings.

7 FUTURE SCOPE

Our work provides a platform for the development of automated diagnostic systems based on cell separation and analysis, with the goal of covering multiple medical conditions and integrating real-time diagnostic capabilities. Future work will focus on increasing accuracy by combining advanced algorithms and real-time feedback techniques for faster healthcare interventions. The benefits of deep learning algorithms and reinforcement learning will greatly improve the performance and robustness of these automated diagnostic systems.

The integration of advanced machine learning techniques into cell separation algorithms is critical to improve accuracy and efficiency. Future efforts will go deeper into applying deep learning algorithms, reinforcement learning, and transfer learning to deal with cellular structure complexity and variability to increase the overall performance of classification algorithms

Our work also emphasizes integrating data from multiple imaging modalities, such as fluorescence microscopy and confocal imaging, to improve the accuracy and detail of cell separation. Future research will focus on optimizing fusion algorithms, developing robust methods to solve multimodal data integration challenges, exploring new imaging techniques for advanced cell analysis and diagnosis. The development of tools for quantitative analysis are important aspects to be explored further to improve the analytical and diagnostic capabilities of biomedical image analysis.

8 BIBLIOGRAPHY

[1] Interpretation and Visualization Techniques for Deep Learning Models in Medical Imaging. Daniel T.Huff, Amy J. Weisman and Robert Jeraj

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation in International Conference.

[3] Max Jaderberg, Karen Simonyan, Andrew Zisserman, Koray Kavukcuoglu Spatial Transformer Networks , arxiv.org/abs/1506.02025.

[4] U-Net-STN: A Novel End-to-End Lake Boundary Prediction Mode by Lirong Yin,Lei Wang, Tingqiao Li,Siyu Lu, Zhengtong Yin, Xuan Liu, Xiaolu Li and Wenfeng Zheng.

[5] Christoffer Edlund, Timothy R Jackson, Nabeel Khalid, Nicola Bevan, Timothy Dale, Andreas Dengel, Sheraz Ahmed, Johan Trygg, and Rickard Sj¨ogren. Livecell—a large-scale dataset for label-free live cell segmentation. Nature methods, 18(9):1038– 1045, 2021. 2, 3, 4

[6] Deep Learning in Cell Image Analysis Junde Xu, Donghao Zhou, Danruo Deng, Jingpeng Li, Cheng Chen, Xiangyun Liao, Guangyong Chen, and Pheng Ann Heng.

[7] Noah F Greenwald, Geneva Miller, Erick Moen, Alex Kong, Adam Kagel, Thomas Dougherty, Christine Camacho Fullaway, Brianna J McIntosh, Ke Xuan Leow, Morgan Sarah Schwartz, et al. Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning. Nature biotechnology, pages 1–11, 2021. 2

[8] Kaiming He, Georgia Gkioxari, Piotr Doll´ar, and Ross Girshick. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017. 3

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770– 778, 2016. 3

[10] Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. An energy and gpu computation efficient backbone network for real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2019.

[11] Youngwan Lee and Jongyoul Park. Centermask: Real-time anchor-free instance segmentation. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 13906–13915, 2020. 3