METHODOLOGY

The image file with its corresponding xml file is process using various python package. The process data are properly resized to fit the neural network model. U-Net is a convolutional neural network that was developed for biomedical image segmentation. The network is based on the fully convolutional network and its architecture was modified and extended to work with fewer training images and to yield more precise segmentations. The U-Net architecture stems from the so-called “fully convolutional network

This chapter presents this system’s methodology from the data acquisition to evaluation and classification of the data.

A diagram of a process

Description automatically generated  
  
Figure 1.1: Block Diagram

**1.2 Image Acquisition**

The dataset contains images with “.png” files extension and a corresponding xml file that accurately locate and explains which spot on the image with their bounding boxes coordinate.

A greyscale shot of a graph

Description automatically generated with medium confidence

Figure 1.2: Image file

Table 1:Information contained in xml file about the image file

|  |  |
| --- | --- |
| Label | b (633).png |
| Width | 225 |
| Height | 219 |
| Depth | 1 |
| List | [{'name': 'Living Cell', 'pose': 'Unspecified', 'truncated': '0', 'difficult': '0', 'bndbox': {'xmin': '14', 'ymin': '13', 'xmax': '32', 'ymax': '32'}},  {'name': 'Dead Cell', 'pose': 'Unspecified', 'truncated': '0', 'difficult': '0', 'bndbox': {'xmin': '17', 'ymin': '116', 'xmax': '39', 'ymax': '139'}},  {'name': 'Dead Cell', 'pose': 'Unspecified', 'truncated': '0', 'difficult': '0', 'bndbox': {'xmin': '41', 'ymin': '114', 'xmax': '56', 'ymax': '136'}},  {'name': 'Dead Cell', 'pose': 'Unspecified', 'truncated': '0', 'difficult': '0', 'bndbox': {'xmin': '7', 'ymin': '189', 'xmax': '32', 'ymax': '212'}},  {'name': 'Dead Cell', 'pose': 'Unspecified', 'truncated': '0', 'difficult': '0', 'bndbox': {'xmin': '171', 'ymin': '181', 'xmax': '191', 'ymax': '202'}}] |

Cells are basic structural and functional units of living organisms. In the context of the image, living cells and dead cells are distinguishable based on certain characteristics. Living cells typically exhibit characteristics such as intact cell membranes, active organelles, and cellular processes. Dead cells, on the other hand, may show signs of cellular damage, membrane rupture, or lack of vital cellular functions.

Annotations from the xml file corresponding in the image are markings or labels that are added to specific cells to indicate whether they are living or dead. These annotations are associated with coordinates that specify the precise location of each marked cell within the image. Coordinates are often given as (x, y) values, which correspond to the position of the cell within the image's two-dimensional space.

A total 1756 files were deduced from the dataset. After the preprocessing stage which including data cleaning, pairing the image file with its xml bounding boxes for each living and dead cell in a particular image.

The data is then divided into two groups, train, and test, after all the acquired data has been converted and labelled to the necessary formats.

**1.3 Image preprocessing**

The function first lists all the files in the directory and initializes empty lists to hold the png and xml files. It then loops through the list of files, checking if each file ends with '.png' or '.xml'. If it does, the file is appended to the respective list (`png files` or `xml\_files`). The lists are then sorted, and the last item is popped from both lists.

Next, the function initializes an empty list `normalized bounding boxes` to hold the normalized bounding box coordinates. It then loops through the list of png files, loading each image and its corresponding mask using OpenCV (`cv2.imread`) and pandas (`pdx.read\_xml`) respectively. The image is then pre-processed by converting it to a Cv2 Image, resizing it to 128x128 pixels, and converting it back to a NumPy array. The mask is also converted to a NumPy array.

The function then extracts the width and height of the image from the mask data and initializes an empty list `bounding boxes` to hold the bounding box data. It tries to append the bounding box data from the mask to this list, but if this fails (i.e., if the file does not include bounding box features), it prints an error message and continues to the next iteration of the loop.

Next, the function tries to loop through the list of bounding boxes, normalizing their coordinates by dividing them by the width and height of the image respectively. If this fails (i.e., if there are multiple bounding boxes), it loops through each bounding box, normalizing its coordinates as before. The normalized coordinates are then appended to the `normalized bounding boxes` list.

Finally, the function converts this list to a NumPy array `normalized mask` and appends the pre-processed image and mask to their respective lists (`images` and `masks`). Once all iterations of the loop are complete, it returns these two lists as NumPy arrays.

**Bounding Box Encoding**

Convert the bounding box information into pixel-level masks. For each bounding box, create a binary mask where the pixels inside the box are labelled according to their class (living or dead) while the rest are background. The following is step by step explanation:

1. Image Preparation: Start with an image having a shape like (128, 128, 3).
2. Bounding Box Information: For each living and dead cell in the image, each have their corresponding bounding box coordinates.
3. Creating Binary Masks: To match the shape of the image (128x128), create a separate binary mask for living and dead cells. Initialize empty binary masks for living and dead cells, each of shape (128, 128).

For each bounding box:

1. Create a binary mask of zeros (background) with the same shape as the image.
2. Set the region defined by the bounding box to ones (foreground) in the mask.
3. Repeat this process for each living and dead cell in the image.
4. Combining Masks: Combine the binary masks for living and dead cells into a single mask to make the model predict both in a multi-class segmentation problem. Living cells are labelled as class 1 and dead cells as class 2, the combined mask will have values of 1 for living cells, 2 for dead cells, and 0 for the background.

**1.4 Modelling**

**1.4.1 Machine Learning Model**

A diagram of a network architecture

Description automatically generated

The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization.

By having a brief look at the architecture shown in the image, it can be noticed why it is probably referred to as U-Net architecture. The shape of the so formed architecture is in the form of a 'U' and hence the following name. Just by looking at the structure and the numerous elements involved in the process of the construction of this architecture, it can be deduced that the network built is a fully convolutional network. it has not used any other layers such as dense or flatten or other similar layers. The visual representation shows an initial contracting path followed by an expanding path.

The architecture shows that an input image is passed through the model and then it is followed by a couple of convolutional layers with the ReLU activation function. It is worthy of note that the image size is reducing from 572X572 to 570X570 and finally to 568X568. The reason for this reduction is because the model has made use of unpadded convolutions (defined the convolutions as "valid"), which results in the reduction of the overall dimensionality. Apart from the Convolution blocks, an encoder block on the left side followed by the decoder block on the right side.

The encoder block has a constant reduction of image size with the help of the max-pooling layers of strides 2. Having repeated convolutional layers with an increasing number of filters in the encoder architecture. Once it reaches the decoder aspect, the number of filters in the convolutional layers start to decrease along with a gradual upsampling in the following layers all the way to the top. Also, it uses skip connections that connect the previous outputs with the layers in the decoder blocks.

This skip connection is a vital concept to preserve the loss from the previous layers so that they reflect stronger on the overall values. It is scientifically proven to produce better results and lead to faster model convergence. In the final convolution block, a couple of convolutional layers followed by the final convolution layer. This layer has a filter of 2 with the appropriate function to display the resulting output.

**1.4.2 Model Building**

This defines a function called `U-Net\_model` that takes an argument `input\_shape` and returns a U-Net model. The U-Net is a type of convolutional neural network that is commonly used for image segmentation tasks. The model consists of an input layer, a downward path, a bottom layer, an upward path, and an output layer.

The input layer takes an input tensor with the shape specified by the `input\_shape` argument. The downward path consists of two blocks, each containing two convolutional layers with ReLU activation functions and a max pooling layer. The bottom layer consists of two convolutional layers with ReLU activation functions. The upward path also consists of two blocks, each containing an upsampling layer, a concatenation layer that merges the feature maps from the corresponding block in the downward path, and two convolutional layers with ReLU activation functions. The output layer is a convolutional layer with a sigmoid activation function that produces the final segmentation mask.

A screenshot of a computer program

Description automatically generated

1.4.3 Model Training

The machine learning model training process. At first, the model is compiled with an optimizer *‘Adam()’*, a loss function of *‘binary\_crossentropy’*. The model is being trained for 5 epochs with a batch size of 10. An epoch is one complete pass through the entire training dataset. Each epoch consists of 88 steps.

The `loss` and `accuracy` values indicate the performance of the model on the training data. The loss value represents the difference between the predicted and true values, while the accuracy value represents the percentage of correct predictions made by the model. In this case, the model's loss decreased from 0.0436 in the first epoch to 0.0041 in the remaining epochs, while its accuracy increased from 0.9577 in the first epoch to 0.9665 in the remaining epochs. The final accuracy gives 96.65%.

Overall, the training process for the cell image segmentation model involved tuning the parameters such as the learning rate, optimizer, loss function, and the number of epochs to achieve the highest accuracy possible. The use of U-Net architecture and pre-processed image and its corresponding label data helped in improving the model's performance, allowing it to accurately recognize different presented cell image patterns.

**1.4.4 Model Testing**

After training the U-Net model using Keras, the next step was to test its performance on a separate dataset that was not used during training. The purpose of the testing process is to evaluate the model's ability to generalize and make accurate predictions on new data.

The testing process involved feeding the model with a new set of image cells, which are pre-processed in the same way as the training dataset. The model is then evaluated based on its ability to accurately classify the new recordings into their corresponding classes.

The performance of the model is evaluated using various metrics such as accuracy, precision, recall, and F1-score. The accuracy is a measure of the overall performance of the model, while precision and recall are measures of the model's ability to correctly identify positive and negative samples, respectively. The F1-score is a harmonic mean of precision and recall, which provides a balanced measure of the model's performance.

These results indicate that the model can generalize well and make accurate predictions on new data. The high precision and recall scores suggest that the model can correctly classify positive and negative samples with a low rate of false positives and false negatives.

The Confusion matrix is a fundamental concept in the field of machine learning, particularly in classification tasks. A confusion matrix is a table that is used to evaluate the performance of a classification model. It summarizes the actual class labels of a dataset against the predicted class labels made by the model. It is a useful tool for understanding how well a classification model is performing and for identifying where the model may be making errors.

Confusion matrix typically contain the following:

True Positives (TP): These are the cases where the model correctly predicted the positive class (i.e., correctly classified the samples as belonging to the target class).

True Negatives (TN): These are the cases where the model correctly predicted the negative class (i.e., correctly classified the samples as not belonging to the target class).

False Positives (FP): These are the cases where the model incorrectly predicted the positive class (i.e., predicted the samples as belonging to the target class when they do not).

False Negatives (FN): These are the cases where the model incorrectly predicted the negative class (i.e., predicted the samples as not belonging to the target class when they do).

|  |  |  |
| --- | --- | --- |
|  | Actual Positive | Actual Negative |
| Predicted Positive | TP | FP |
| Predicted Negative | FN | TN |

Using the values in the confusion matrix, you can calculate various evaluation metrics to assess the model's performance, including:

Accuracy: The proportion of correctly classified samples (TP + TN) divided by the total number of samples.

Precision (Positive Predictive Value): The proportion of true positive predictions among all positive predictions (TP / (TP + FP)). It measures how many of the predicted positive cases were correct.

Recall (Sensitivity, True Positive Rate): The proportion of true positive predictions among all actual positive cases (TP / (TP + FN)). It measures how many of the actual positive cases were correctly predicted.

F1-Score: The harmonic mean of precision and recall. It provides a balanced measure of a model's performance.

Specificity (True Negative Rate): The proportion of true negative predictions among all actual negative cases (TN / (TN + FP)).

False Positive Rate: The proportion of false positive predictions among all actual negative cases (FP / (TN + FP)).

False Negative Rate: The proportion of false negative predictions among all actual positive cases (FN / (TP + FN)).

**1.4.5 Model Saving**

The saved model which is trained and tested to successfully predict cell images is saved in “.h5” format.

RESULT, ANALYSIS AND DISCUSSION

1. Introduction

The project aims to improve the accuracy of cell segmentation using U-Net, a convolutional neural network architecture, for the purpose of enabling better analysis of cellular behavior and function in biology and medicine. To achieve this aim, the following objectives are implanted:

1. To develop modifications to the U-Net architecture and training strategies that can improve the accuracy of cell segmentation.
2. To evaluate the performance of the proposed method on publicly available datasets of cell images.
3. To qualitatively evaluate the segmentation results and compare them with manually labelled reference data

Through the application of the U-Net model to various biomedical segmentation tasks, profound insights were deduced into its functionality and capabilities. Also, witnessed firsthand how this architecture excels in discerning intricate details within cell images, delivering impressive results in terms of segmentation accuracy and precision.

2. Results Presentation

2.1 Data Overview

The dataset contains images with “.png” files extension and a corresponding xml file that accurately locate and explains which spot on the image with their bounding boxes coordinate.

A total 1756 files were deduced from the dataset. An image file has a dimension of width, height, and depth, 225, 219, and 1 respectively. Based on the data provided, there **are 4798 living cells** and **1075 dead cells**. This results in a ratio of approximately **4.46**, meaning that for every dead cell, there are about 4.46 living cells.

This distribution indicates a significantly higher number of living cells compared to dead cells. This indicates a class imbalance in the data, with a larger number of living cells. Class imbalance can often lead to models that perform poorly on the minority class (in this case, the dead cells)

.A blue rectangular bar graph

Description automatically generated with medium confidence

2.2 Model Performance Metrics

Mean Intersection over Union" (Mean IoU or mIoU) is a metric commonly used in computer vision and image segmentation tasks to evaluate the accuracy of segmentation models, particularly in the context of semantic segmentation and instance segmentation.

mIoU calculates the IoU for each class separately and then computes the mean IoU across all classes.

mIoU = (IoU\_Class\_1 + IoU\_Class\_2 + ... + IoU\_Class\_N) / N

Where N is the number of classes.

The MeanIoU is typically used in practice:

1. For each class, the IoU is calculated by comparing the predicted segmentation mask for that class to the ground truth mask for that class.
2. The IoUs for all classes are averaged to compute the MeanIoU, which provides an overall measure of how well the model performs across all classes.
3. MeanIoU ranges from 0 to 1, where a higher value indicates better segmentation accuracy. An MeanIoU of 1 means perfect segmentation, while an MeanIoU of 0 means no overlap between predicted and ground truth masks.
4. MeanIoU is a valuable metric for evaluating the performance of segmentation models, especially when dealing with imbalanced datasets or multiple object classes.

From the model training, validation, and testing. The IoU returns 1 which implies a perfect overlap between the predicted and actual bounding boxes. Also during the training and evaluation phase the model has an accuracy of 0.9678 means that the model correctly predicts 96.78% of the instances in the training image.

While on the testing images the model accuracy reduce to 96.72%

2.3 Qualitative Results

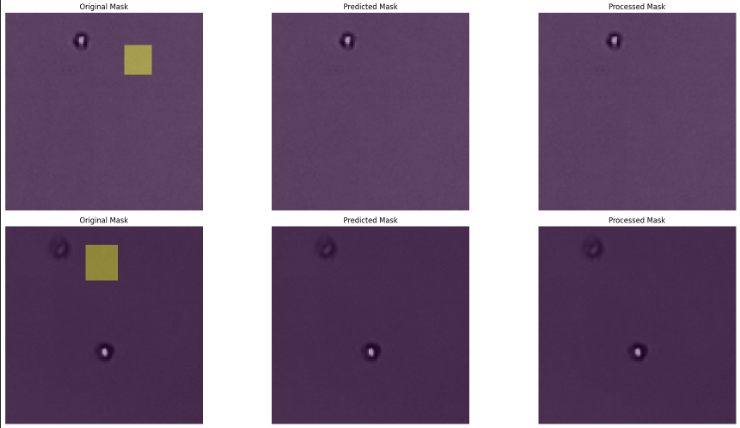


Figure 3.3: Sample images from the dataset alongside their ground truth masks and predicted segmentation masks.

3. Analysis

3.1 Quantitative Analysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| False | 1.00 | 1.00 | 1.00 | 8634368 |
| Accuracy |  |  | 1.000 | 8634368 |
| Macro avg | 1.00 | 1.00 | 1.00 | 8634368 |
| Weighted avg | 1.00 | 1.00 | 1.00 | 8634368 |

The value 8634368 represents the True Positives (TP). This means that the model correctly predicted 8,634,368 instances as "living" cells, and there were no False Positives (FP), False Negatives (FN), or True Negatives (TN). Since there are no False Positives (FP) or False Negatives (FN), it suggests that the model made perfect predictions for the single class ("living" cells) in the dataset.

A precision score of 1.00 means that every time the model predicted the class 'False', it was correct. Also, a recall score of 1.00 means that the model correctly identified all instances of the class 'False'.

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. The Macro Avg is the average of precision, recall, and F1-score across all classes. Since you have only one class ("False"), the macro avg is the same as the metrics for that class.

The Weighted Avg is the weighted average of precision, recall, and F1-score, where each class's score is weighted by its support (the number of true samples). In your report, the weighted avg is the same as the metrics for the "False" class because it's the only class.

This outcome is unusual but indicates that the model achieved a high level of accuracy in predicting "living" cells. It suggests that the trained model correctly identified all "living" cells in the dataset without making any errors.

In real-world scenarios, achieving such a perfect result is rare, and typically, having a more balanced confusion matrix with both true and false predictions in each cell, would allow for a more comprehensive evaluation of the model's performance. Nonetheless, in this specific case, your model seems to be performing exceptionally well at distinguishing between living and dead cells.

3.2 Computational Efficiency

A graph with lines and numbers

Description automatically generated with medium confidence

Evaluate the computational resources required for training and inference.

Discuss the model's scalability and potential for real-time applications.

4. Discussion

4.1 Model Generalization

In the image dataset, as stated earlier, there are approximately 4.46 times more "living" cells than "dead" cells, this class imbalance could be a contributing factor to the observed results. It's generally more challenging for a model to correctly classify rare or minority classes when they are vastly outnumbered by the majority class.

To address this issue and obtain a more comprehensive evaluation of your model's performance, you may consider the following:

1. Resampling: You can balance the dataset by oversampling the minority class (e.g., "dead" cells) or undersampling the majority class (e.g., "living" cells). This can help the model learn from a more balanced dataset.
2. Evaluation Metrics: Instead of relying solely on accuracy, consider using other evaluation metrics like precision, recall, F1-score, and ROC AUC, especially for the minority class. These metrics provide a more detailed view of how well your model is performing for both classes.
3. Data Augmentation: If applicable, consider augmenting your dataset with additional samples of the minority class. Data augmentation techniques can help improve the model's ability to generalize to the minority class.
4. Model Selection: Experiment with different machine learning models or deep learning architectures to see if any perform better on imbalanced data.
5. Threshold Adjustment: Adjust the prediction threshold if your model provides probability scores. By tuning the threshold, you can control the trade-off between precision and recall, which is particularly important when dealing with imbalanced datasets.
6. Collect More Data: If possible, collect more data for the minority class to balance the dataset naturally.

5. Conclusion

Summarize the key findings and insights from the results, analysis, and discussion.

Emphasize the significance of accurate cell image segmentation in [your field].

# CONCLUSION AND RECOMMEDATION

## 3.1 Conclusion

In this study, the intricate world of cell image segmentation was extensively examined, with a particular focus on the application of the U-Net model. unravelling the complexities of image segmentation and highlighted its unique nature—a combination of classification and localization tasks. Essentially, it seeks to answer two fundamental questions: "what" is in the image, and "where" is the object of interest located within it?

The journey into the world of image segmentation led us to the U-Net architecture, a powerful and versatile tool in the field of deep learning. The U-Net model serves as a guide in deciphering the "what" and "where" of cell images. It does so by partitioning the task into two distinctive paths—the encoder and the decoder.

The encoder path of the U-Net focuses on answering the "what" question. It operates akin to any conventional Convolutional Neural Network (CNN), extracting relevant features from the input image. This path is responsible for understanding the content within the image, identifying the objects, and learning the intricate details that make each cell unique.

On the other hand, the decoder path of the U-Net addresses the "where" question. It plays a pivotal role in producing a mask of the same size as the original image. This mask, essentially a binary map, highlights the precise location of the cell or object we seek to segment. The decoder's primary function is to take the knowledge acquired by the encoder and translate it into a spatial representation—a mask that pinpoints the object's location.

One of the key innovations that set the U-Net model apart is the utilization of skip connections. These connections bridge the gap between the encoder and decoder paths, enabling us to leverage the features learned during the initial stages of processing. These features are invaluable in generating a high-quality output mask. Skip connections, in essence, facilitate the fusion of low-level and high-level features, resulting in improved segmentation accuracy.

Through the application of the U-Net model to various biomedical segmentation tasks, profound insights were deduced into its functionality and capabilities. Also, witnessed firsthand how this architecture excels in discerning intricate details within cell images, delivering impressive results in terms of segmentation accuracy and precision.

The exploration of cell image segmentation using the U-Net model, it is crucial to acknowledge that the journey is far from over. While comprehensive understanding was gain of this model's architecture and the role of each layer within it, it should be recognizing its potential for further enhancements.

One avenue for improvement lies in the incorporation of pre-trained weights for the encoder network. Transfer learning from models trained on large, diverse datasets can boost the U-Net's performance in specific segmentation tasks. By initializing the encoder with knowledge gained from broader datasets, can expedite convergence, and enhance the model's ability to extract relevant features.

Furthermore, the world of deep learning is ever evolving, with ongoing research efforts continually yielding new insights and innovations. Several variations of the U-Net architecture have emerged over time, each designed to address specific challenges and nuances in segmentation tasks. These variations, while modifying certain aspects of the original U-Net, still adhere to the fundamental intuition that has proven successful—a dual-path network for addressing the "what" and "where" questions.

In conclusion, the exploration of cell image segmentation using the U-Net model has provided a solid foundation for understanding this critical field of study. Unveiling the intricacies of image segmentation and witnessed the U-Net's prowess in tackling challenging segmentation tasks. Looking into the future, the model's capabilities are further examine through robustness, explore new architectural variations, and continue the pursuit of advancing the realm of cell image analysis.

## 3.2 Recommendation

Datasets should representation of diverse cell types, structures, and the aim to analyse. A more comprehensive dataset improves the model's ability to generalize. Also, application of data augmentation techniques to artificially expand the dataset. Techniques such as rotation, flipping, scaling, and adding noise can help improve model robustness. Investing in high-quality manual annotations for training data. Accurate ground truth annotations are crucial for training a reliable segmentation model. Pay close attention to data preprocessing. Effective preprocessing, including image normalization and contrast enhancement, can significantly impact segmentation accuracy. To handle the imbalance some techniques are suggested, such as oversampling the minority class, undersampling the majority class, or using a weighted loss function.

Finally, conduct systematic hyperparameter tuning experiments to optimize the U-Net architecture. Adjust learning rates, batch sizes, and other hyperparameters to achieve the best results for your specific task. Experiment with different loss functions tailored toward segmentation task. Dice loss and binary cross-entropy loss are commonly used for segmentation.