Deep Learning for Crack Segmentation in Infrastructure

Jeevi Scarozza
University of Geneva: Physics Applications of AI

June 2024

Chapter 1

Aim

The aim of this project is to develop a deep learning model for crack segmentation in metal images using a simple U-Net architecture. The goal is to accurately identify and cracks in images surfaces to aid in preventive maintenance of different infrastructure. U-Net is a type of convolutional neural network (CNN). The goal of this project was to develop a solid understanding of this architecture. What distinguishes the U-Net architecture from other CNNs is the unique structure, which includes a contracting path (encoder) to downsample, and a symmetric expanding path (decoder) for upsampling. The U-Net architecture is particularly known to be suitable for image segmentation tasks, making it the right model for this project.

Chapter 2

The Dataset

The dataset used in this project is the CrackSeg9k dataset, which consists of a combination of 10 smaller datasets preprocessed and resized to 400x400 pixels.These smaller datasets include Crack500, Deepcrack, Sdnet, Cracktree, Gaps, Volker, Rissbilder, Noncrack, Masonry, and Ceramic, however it is further resized in this version.

The CrackSeg9k dataset contains images of cracks in various materials, including concrete, building walls, roads, masonry, and ceramic. This diversity in material may present both opportunities and challenges for crack segmentation models. While training on a diverse dataset can help the model generalize better to different types of cracks, it may also give variations that could affect performance. One possibility for future work could be to train the model specifically on metal images if the primary focus is on detecting cracks in metal materials. This targeted approach may lead to better performance and accuracy for crack segmentation tasks in metal, such as the bellows of the Large Hadron Collider (LHC), which is what has inspired this project.

2.1 Exploration and Visualization

Exploration and Visualization Dataset Visualization Preprocessing Notebook: To understand the dataset and visualize the images, a preprocessing.ipynb notebook was created. This notebook allowed the exploration of the data by providing visual insights into the images and their corresponding ground truth masks. This allows us to display several examples of input images alongside their ground truth masks. It also enabled inspection of the quality and variety of the dataset, ensuring the images and masks were correctly aligned. I was also able to identify issues in the dataset, such as image with a missing segmentation mask, which was removed from the training set.

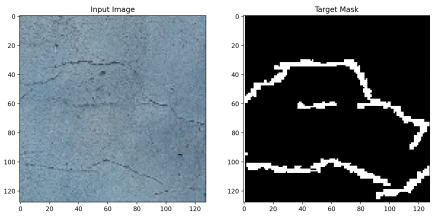


Figure 2.1: Example of Test Image and its Corresponding Ground Truth Mask

2.1.1 Class Imbalance Handling

Class Weight Calculation: The dataset exhibited class imbalance, which could negatively impact model performance by biasing it towards the majority class. To address this, class weights were calculated to balance the loss function, ensuring that the model pays equal attention to both classes.

Calculation Method: The weights were computed based on the frequency of each class in the training set. These weights were then incorporated into the loss function.

2.1.2 Dataset Splitting

Data Splitting: Ensured correct model training and evaluation by dividing the data into train, validation, and test sets. The model should not be evaluated on the same data that it is trained on, as this would induce bias. The data was also split into a train, validation, and test set. The exact size of each is as follows: Train set: 5052 images Validation set: 631 images Test set: 632 images

A comprehensive exploration and visualization phase allowed for subsequent pre-processing, by ensuring that the dataset was well-prepared and understood before model development, and the identification of data issues.

Chapter 3

Method

3.1 UNet Architecture in Detail

The chosen method for crack segmentation is a U-Net architecture implemented using Keras. The U-Net was introduced in 2015 for the purpose of biomedical image segmentation. The model consists of a contracting path to downsample images, for feature extraction, and a symmetric expanding path for upsampling.The U-Net is then connected through what are called skip connections. These skip connections improve model performance and speed for image segmentation tasks, by directly connecting feature maps from the encoder side to the decoder side.

Encoder Path: Consists of a series of convolutional layers with increasing depth. Each convolutional layer is followed by batch normalization, Leaky ReLU activation, and dropout for regularization. MaxPooling layers are used to reduce the spatial dimensions of the feature maps, in order to help identify high level features. The encoder path is responsible for feature extraction.

Decoder Path: Uses transposed convolutional layers for upsampling. Each upsampling layer is followed by concatenation with the corresponding feature map from the encoder path; these are the skip connections. Just like in the encoder, each layer in the decoder is followed by batch normalization, Leaky ReLU activation, and dropout. This side of the U-Net helps with localization.

Final Convolution: A final convolutional layer with a sigmoid activation function produces the output segmentation map, where each pixel value represents the probability of belonging to the crack class. This is a value between 0 and 1. Since this task is a binary classification task, this is the appropriate function to select.

Custom Training Step: The custom training step method (seen at the end of the models source code) was added to allow for computing the customized loss function.

3.2 Pre-processing the Data

Pre-processing steps are essential for preparing the raw data before feeding it into the model. In this project, the pixel values of the images were scaled to a range of 0 to 1. This normalization process is necessary for allowing convergence during model training.

The function get dataset was used to set up the TensorFlow dataset. This function helps in loading and pre-processing the images for training by converting the images into tensors which can be fed into the model.

Handling Missing Data: During the initial setup, one image was found to be missing its corresponding mask. This issue was discovered while running the train.py script. Debugging was necessary to identify the problematic image. The missing mask was handled by removing the image from the dataset to ensure consistency in the training process.

3.3 Model Training

In this project, I carefully selected hyperparameters such as batch size, learning rate, and dropout rate to try to optimize the performance of our crack segmentation model. TensorFlow datasets allows us to train the model on batches of images rather than the entire dataset at once. This batching process not only conserves memory and can result in faster training times if selecting a larger batch size. By adjusting hyperparameters and evaluating model performance on validation data, we can fine tune the model to achieve the best possible results.

Methodology

Optimizer

The U-Net model was compiled using the Adam optimizer, known for its efficiency and effectiveness in training deep learning models. The Adam optimizer combines the advantages of both the AdaGrad and RMSProp algorithms, allowing for adaptive learning rates.

Loss Function

Initial Loss Function

The Binary Cross-Entropy (BCE) loss function was chosen initially for training the U-Net model. BCE is suitable for binary classification problems and is

widely used in segmentation tasks where each pixel is classified as either foreground or background. BCE measures the performance of a classification model whose output is a probability value between 0 and 1. The BCE loss increases as the predicted probability diverges from the actual label.

Choice of Activation Function: Leaky ReLU vs. ReLU

ReLU: The Standard Activation Function. In this case, the Leaky ReLU was selected instead. This is due to the non-negative bias. Since ReLU outputs zero for all negative inputs, it might lead to neurons becoming inactive when a large portion of the input data is negative. Due to the class imbalance favored towards the majority class (non crack), this was a potential solution.

More Modifications

Later in the training process, the loss function was modified to better address the class imbalance. I implemented a custom loss function. This is a modification of the binary cross-entropy, through scaling with weights. A weighted vector was calculated using the following:

$$\text{weight\_vector} = y_{\text{true}} \times \text{class\_weights}[1] + (1 - y_{\text{true}}) \times \text{class\_weights}[0] \quad (3.1)$$

$$\text{weighted\_bce} = \text{weight\_vector} \times \text{bce} \quad (3.2)$$

Evaluation Metrics

Metrics for Evaluating U-Net Model Performance:

Binary Accuracy:

- Measures the proportion of correctly predicted pixels over the total number of pixels.

Mean IoU (Intersection over Union):

- Calculates the average IoU for all classes, which is a robust metric for segmentation tasks, measuring the overlap between the predicted and ground truth masks.

False Negatives and False Positives:

- Provide insights into the types of errors the model makes, crucial for the task, as for this particular situation, it was a challenge to generate any true positives initially, due to the class imbalance.

Precision and Recall:

- Precision measures the accuracy of positive predictions, while recall measures the ability to identify all positive instances.

Custom IoU Metric:

- A custom IoU computation metric was also included to provide additional insights specific to the dataset and task requirements.

3.3.1 Model Compilation

The model compilation command integrates the optimizer, loss function, and evaluation metrics, setting up the model for training with the specified configurations.

Optimizer Selection:

Adam optimizer was chosen due to its straightforwardness to implement, and fast convergence.

Loss Function:

Binary Cross-Entropy was initially selected for its suitability in binary classification tasks, with the modification using class weights.

Weight Decay:

Selected the default of 0.0001 for all models.

Chapter 4

Evaluation of the Method

Hyperparameter Tuning:

To evaluate the performance of the U-Net model, several experiments were conducted by training the network with different sets of hyperparameters. The purpose was to explore the impact of these parameters on the model's performance. Each experiment was run using the command-line interface provided by the script cli.py.

Visuals Generated

Training Loss vs. Validation Loss: Helps visualize how well the model is learning and how it generalizes to unseen data.

Training Accuracy vs. Validation Accuracy: Shows the model's performance over epochs and helps identify overfitting or underfitting issues.

Precision and Recall, and IoU: Precision measures the accuracy of positive predictions made by the model. It answers the question: "Out of all the instances predicted as positive, how many are actually positive?

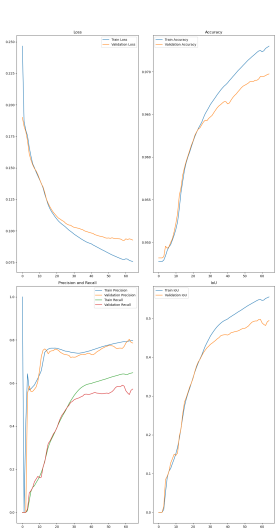
Recall measures the ability of the model to identify all positive instances. It answers the question: "Out of all the actual positive instances, how many did the model correctly identify?"

IoU is discussed in the evaluation section of this report.

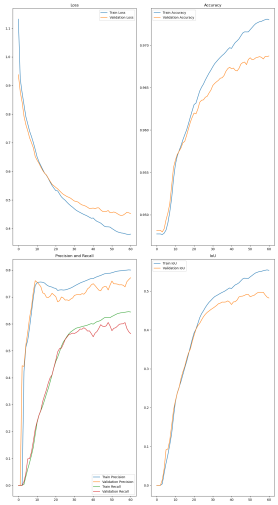
4.1 Graphical Results of Model Training

Training without Weighted Loss Function

Training Observations: The custom loss function resulted in a lower validation loss when compared to the binary cross entropy loss. While the visual



(a) Standard Binary Cross Entropy Loss



(b) Unet 1 Default (Utilized custom weighted loss function)

Figure 4.1: Comparison of Training Histories

results are similar, the custom loss function was kept throughout further training and used as the default.

Training with Binary Cross-Entropy Loss

Comparison of Training with Two Different Learning Rates

Learning Rate: 0.001

Training Observations: Potential risk of overshooting optimal weights due to a higher learning rate.

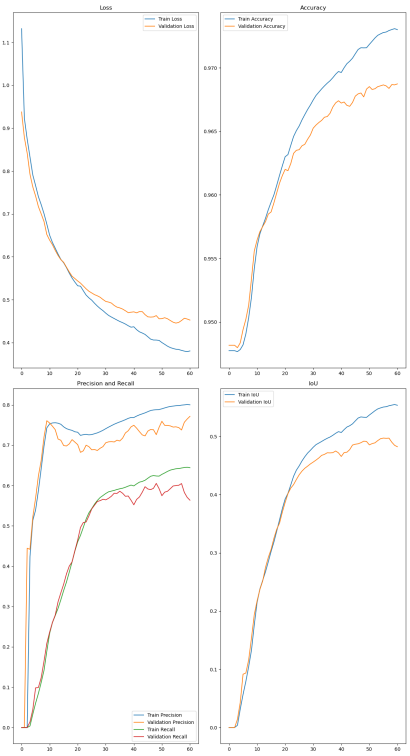
Learning Rate: 0.0001

Training Observations: Slower but smoother convergence with the lower rate.

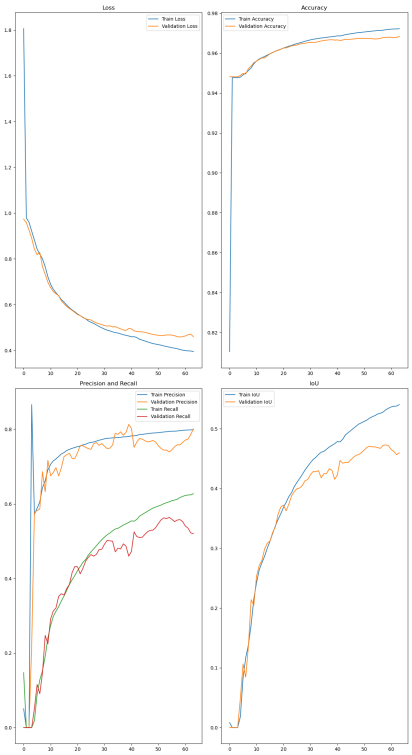
Comparison of Training with Different Dropout Values

Dropout: 0.1, 0.3, and 0.5

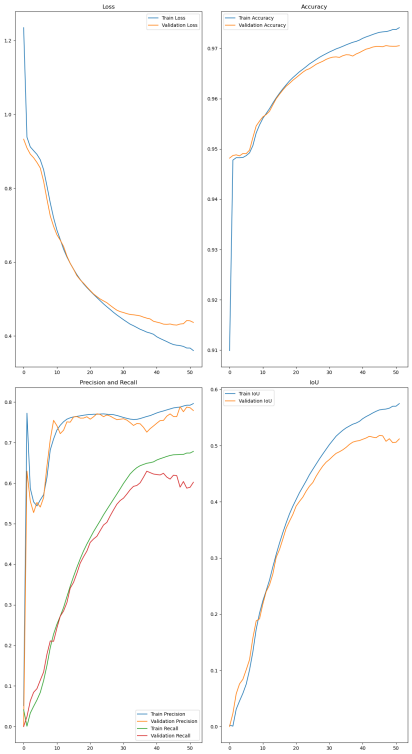
Training Observations: From the quantitative analysis above, we do not observe significant differences in performance between the different dropout values. However, it is important to note that selecting a dropout value too high can result in underlearning. Therefore, we move to qualitative analysis.



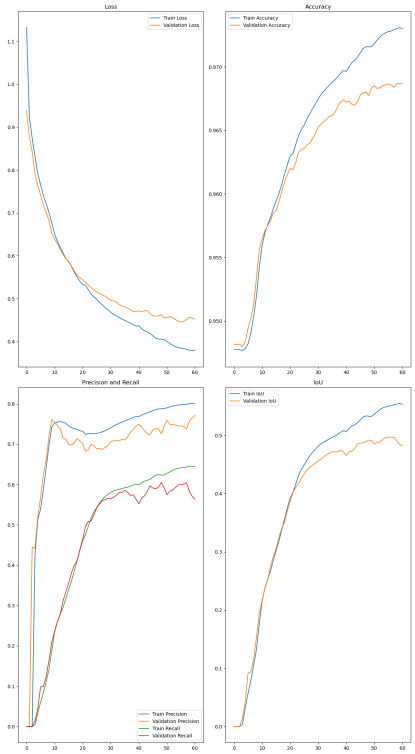
(a) Learning Rate: 0.001 (Unet 1 Default



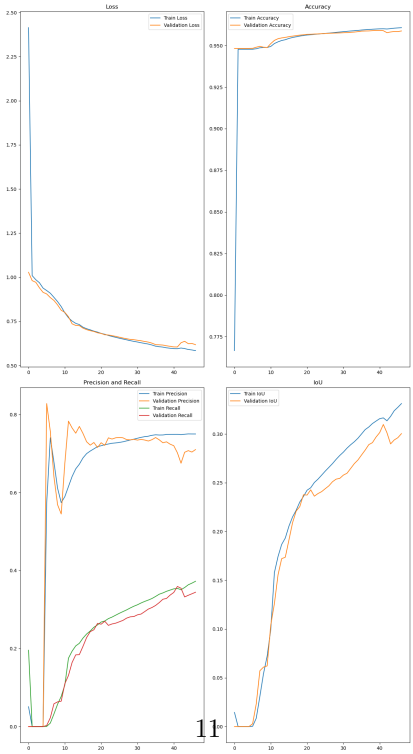
(b) Learning Rate: 0.0001



(a) Batch Size 32

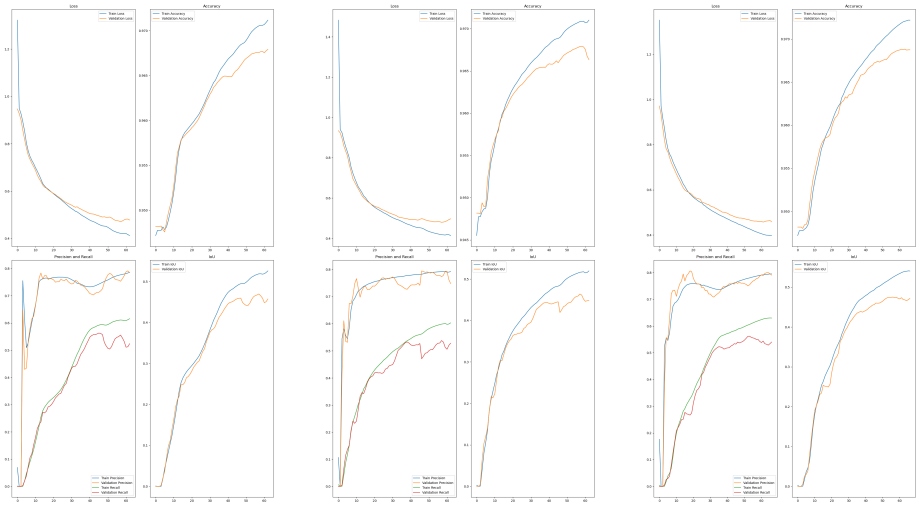


(b) Unet 1 Default (Batch Size 64)



(c) Batch Size 128

Figure 4.3: Training History for Different Batch Sizes



(a) Dropout: 0.1 (b) Dropout: 0.3 (c) Dropout: 0.5

Figure 4.4: Comparison of Training with Different Dropout Values

Better Model Configuration

Training Observations for Model with Batch Size of 32 and a Dropout of 0.1 Combination of smaller batch size and optimal dropout rate to reduce overfitting. However, this model did not achieve the highest IoU score.

4.1.1 Notes about Evaluation

The method was evaluated using standard metrics such as Accuracy and Intersection over Union (IoU). Dice coefficient would be used with more time. Challenges encountered included class imbalance, implementing custom parameters within Keras, and high training times. The model was trained using my computer's CPU. Training for this type of model should typically be done on a GPU, if available. For this type of task, Pytorch may have been a better choice due to the requirements of my task.

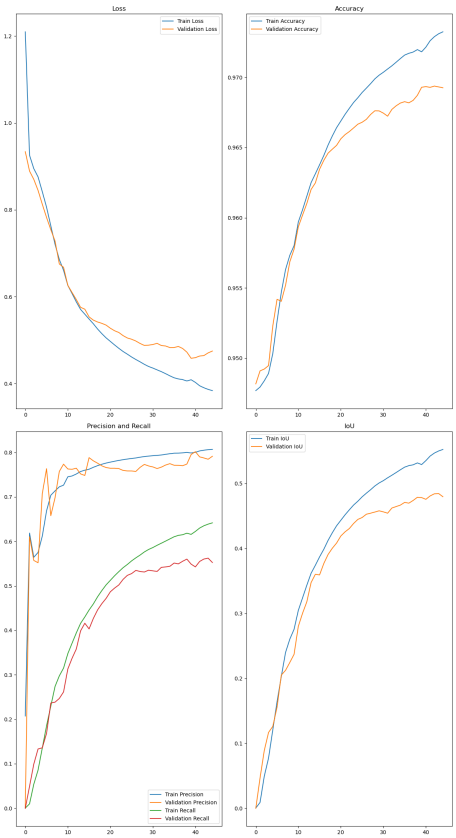


Figure 4.5: Unet with Batch Size of 32 and Dropout of 0.1

Chapter 5

Visual Results on Test Set

5.0.1 The following images show the different model results on a test image. The predicted mask is the model's output.

Visually, we observe that the custom binary cross entropy loss function (Unet 1 Default) produces better results compared with the standard binary cross entropy loss, but the numerical differences are minimal. The Unet 1 had achieved a validation IoU of 0.498 compared to the unweighted binary cross entropy model of 0.495, which is insignificant.

See figure 5.10. The defualt model (and all subsequent models), can correctly identify that there is no crack in the image.

Figure 5.1: Default

Figure 5.2: Unet with Standard Binary Cross Entropy Loss

Figure 5.3: Lower Learning Rate

Figure 5.4: Batch Size 32

Figure 5.5: Batch Size 128

Figure 5.6: Dropout 0.1

Figure 5.7: Dropout 0.3

Figure 5.8: Dropout 0.5

Figure 5.9: Batch Size of 32 and Dropout of 0.1

Figure 5.10: Unet 1 (Default) on Another Test Image

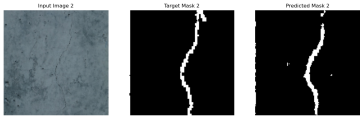


Figure 5.11: Unet with Lower Learning Rate on Another Test Image

Figure 5.12: Unet 1 (Default) Non Crack Image

Chapter 6

Conclusion

6.1 Model Summary

In conclusion, the developed U-Net model demonstrates promising results for crack segmentation in metal images.

$$\text{IoU} = \frac{\text{true\_positives}}{\text{true\_positives} + \text{false\_positives} + \text{false\_negatives}} \tag{6.1}$$

The Intersection over Union (IoU) metric, also known as the Jaccard index measures the overlap between the predicted segmentation mask and the ground truth mask. An IoU score ranges from 0 to 1, where 1 indicates perfect overlap and 0 indicates no overlap at all. This is useful for segmentation tasks because it accounts for both false positives and false negatives, providing a better measure of the model's accuracy. In the context of crack segmentation, a higher IoU indicates that the model is better at correctly identifying the presence and extent of cracks while minimizing both missed detections aka, false negatives, which in this case, was very likely.

Through extensive experimentation, the best-performing U-Net model in this project achieved an IoU value of 0.514. This model was trained with a batch size of 32 and a learning rate of 0.001, which was found to be an adequate balance between the tradeoff of computational efficiency and model performance. Overall, while the achieved IoU score of 0.514 is a significant accomplishment, it also highlights areas for potential improvement.

6.1.1 Comments

It is also important to note, that this project was done using the Keras framework. However, many difficulties were encountered due to the custom evaluation metrics. While the models were saved, some cannot be loaded in the current evaluation script. The model can be evaluated directly in the train script, but the obvious downside is that this is inconvenient to retrain the model when

attempting to just evaluate on new images. In the future, this sort of task with a Unet might be better developed using PyTorch instead. While I initially did not choose Pytorch because of its steeper learning curve, in the end it might have been more suitable.

6.2 Future Work

If given more time, several additional approaches could be explored to further improve this project:

- **Data Augmentation**: Introducing data augmentation techniques such as rotation, scaling, and flipping could help the model generalize better by providing it with more data.
- **Transfer Learning**: Using pre-trained models on similar tasks could be more practical.
- **Grid Search for Hyperparameter Tuning**: Conducting a grid search to find the optimal hyperparameters.
- **Evaluation using the Dice Coefficient**: Using the Dice Coefficient for model evaluation to better capture the overlap between the predicted and ground truth masks.
- **Higher Number of Filers Per Layer**: In this model, I selected 16 as the number of filters per layer based on several considerations. 16 filters per layer were deemed sufficient to capture essential features without excessive computational complexity. However, with a higher number of filters per layer could lead to the model better caputring more detailed features. Due to the computational cost, I did not explore this option.

In conclusion, while the U-Net model achieved reasonable results, there is room for improvement. However, for such a model, there is a strong potential to assist in infrastructure maintenance efforts as intended. The model displayed the ability to identify cracks in a variety of different types of images. The need for improvement lies within the precision of these predictions.

Chapter 7

References

1. Brownlee, J. (2020, August 25). How to reduce overfitting with dropout regularization in Keras. MachineLearningMastery.com. https://machinelearningmastery.com/how-to-reduce-overfitting-with-dropout-regularization-in-keras/

2. Siddharth Sharma, Dhananjay Balakrishnan, Shreyas Kulkarni, Shreyas Singh, Saipraneeth Devunuri, Sai Chowdeswara Rao Korlapati. (2022). *Crackseg9k: A Collection of Crack Segmentation Datasets* (Version V4). Harvard Dataverse. DOI: 10.7910/DVN/EGIEBY. Retrieved from https://doi.org/10.7910/DVN/EGIEB

3. Kulkarni, S., Singh, S., Balakrishnan, D., Sharma, S., Devunuri, S., Korlapati, S. C. R. (2022). *CrackSeg9k: A Collection and Benchmark for Crack Segmentation Datasets and Frameworks.* arXiv preprint arXiv:2208.13054.

4. Gupta, P. (2021, December 17). Understanding skip connections in convolutional neural networks using U-Net Architecture. Medium. https://medium.com/@preeti.gupta02.pg/u skip-connections-in-convolutional-neural-networks-using-u-net-architecture-b31d90f9670a

5. Hasan, S. M. K., Linte, C. A. (2019). U-NetPlus: A Modified Encoder-Decoder U-Net Architecture for Semantic and Instance Segmentation of Surgical Instruments from Laparoscopic Images. In 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) (pp. 7205-7211). Berlin, Germany: IEEE. DOI: 10.1109/EMBC.2019.8856791

6. Ronneberger, O., Fischer, P., Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18 (pp. 234-241). Springer International Publishing.

7. Xu, C., Coen-Pirani, P., Jiang, X. (2023, March 25). Empirical study of overfitting in deep learning for predicting breast cancer metastasis. *Cancers.* DOI: 10.1007/s00464-019-07113-y. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PM