Crack Detection in Pipes Using Deep Learning

Girish Rajani, Khushi Ruparel

B.E. (Artificial Intelligence and Data Science) 4th Year
Thadomal Shahani Engineering College, University of Mumbai

Report Submitted on

Completion of Practical Training at

SECTD-BARC

January 2024

Contents

List of Figures

List of Tables

Synopsis

Traditional methods to identify emerging cracks and damages in pipes are proven to be challenging and even unreliable at times. Ensuring the integrity of these pipelines is crucial, as cracks can emerge due to various factors, necessitating thorough inspections. By leveraging the recent improvements and rapid development in the fields of deep learning and image processing, Neural Networks can be utilized to optimize and automate the process of crack identification and detection.

The project aims to develop a deep learning model for crack tip detection in pipes using images obtained through destructive testing. Semantic Segmentation was used to solve this problem. A model was trained on a dataset comprising diverse crack types, severity levels, and real-world conditions. The U-Net architecture was employed to develop the model, considering all the parameters and their relative relevance. The training of the model was split to include a validation step, where the model was further optimized. An improvement in the model's performance was noticed, as more data augmentation was applied. The successful implementation of this model has the potential to enhance the reliability of pipeline inspections by improving crack detection accuracy and efficiency. The learning-based approach is necessary to handle complex crack patterns, lighting, and camera movements.

1. Introduction

1.1 Problem Statement

Pipelines crucial to industries such as oil and gas, water supply, and chemical processing are susceptible to cracks that may emerge due to various factors, necessitating thorough inspections. Traditional inspection methods, especially those involving destructive testing, provide valuable insights but come with inherent challenges. Deep learning presents an opportunity to enhance the precision and efficiency of crack detection in pipes by analysing images derived from destructive testing methods.

This project aims to develop a sophisticated deep learning model specifically designed for crack tip detection in pipes, utilizing images obtained through destructive testing procedures. The model's training process will leverage a dataset comprising images of pipes subjected to destructive testing, encompassing diverse crack types, severity levels, and conditions reflective of real-world scenarios.

Challenges:

i. **Complex Crack Patterns from Destructive Testing:** Cracks generated through destructive testing may exhibit intricate patterns, making it challenging for the model to discern between genuine crack features and testing-induced artefacts. The model must account for these complexities to ensure accurate detection.

ii. **Varying Lighting Conditions:** The images captured during inspections may be subjected to fluctuating lighting conditions, impacting the visibility of cracks. The model needs to be robust, ensuring consistent performance across diverse lighting scenarios.

iii. **Limited Availability of Destructive Testing Data:** Destructive testing data can be limited due to the high cost and risk associated with such testing procedures. Strategies to handle a potentially limited dataset, including data augmentation, are crucial for robust model training.

1.2 Motivation

The successful implementation of a deep learning model for crack tip detection in pipes using images from destructive testing holds significant promise for enhancing the reliability of pipeline inspections. This approach aims to improve the accuracy and efficiency of crack detection.

Since intricate crack patterns are difficult to describe mathematically. Therefore, learning based approach is adopted to learn crack features from the data itself. Additional complexity arises from lighting and camera pan movements. Learning based approach is required to handle such complexities.

2. Literature Review

Image segmentation involves dividing an image into distinct regions, typically based on pixel characteristics. Unlike image classification or object recognition, it doesn't require prior knowledge of specific objects. Object classification assigns predefined labels like "horse" or "car," whereas image segmentation focuses on grouping pixels.

There are two primary types of image segmentation: semantic segmentation and instance segmentation. Semantic segmentation assigns a single class label to all objects of the same type, while instance segmentation provides separate labels for similar objects.

Semantic segmentation involves breaking down an unfamiliar image into different parts and identifying objects within it, such as a beach, ocean, sun, dog, or swimmer. Unlike object recognition, segmentation goes deeper and doesn't require recognizing objects beforehand. It assigns a category label to each pixel in the image, offering detailed category information at the pixel level.

This pixel-level semantic information is crucial for various real-world applications like self-driving vehicles, pedestrian detection, defect identification, therapy planning, and computer-aided diagnosis. It enables intelligent systems to understand spatial positions and make important judgments. Semantic segmentation stands out from other computer vision tasks, like object classification, which requires annotating the entire image with one or more labels, and object detection, which demands knowledge of where target objects are in the scene.

When presented with a new image, an image segmentation algorithm is expected to identify and group together the pixels that share semantic relevance.

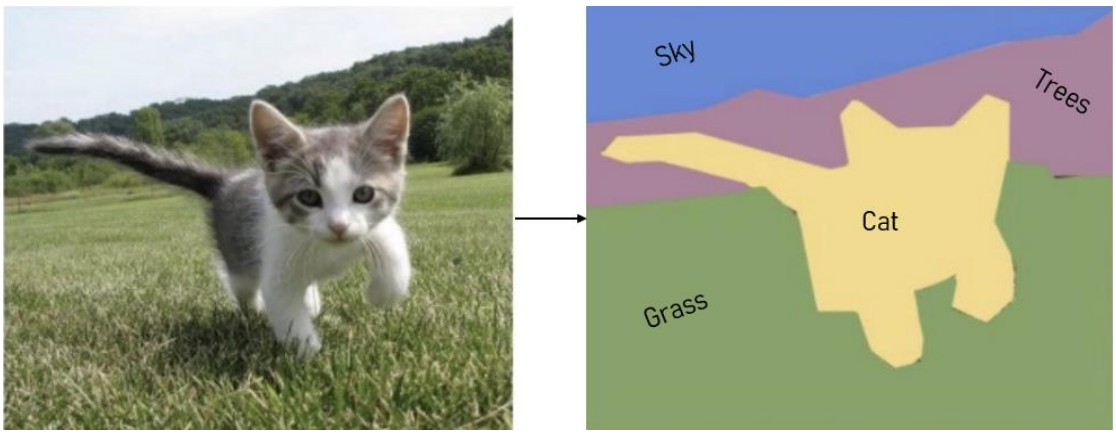


Figure 1: Semantic Segmentation

The issue at hand involves classifying each pixel in an image into one of several possible categories. The objective is to input an image and generate category decisions for individual pixels in the output.

2.1 Naïve approach for semantic segmentation

One approach to address this challenge is the Naïve Approach, treating it as a classification problem with a sliding window. In this method, the input image is divided into multiple crops of the same size. Each crop undergoes processing by a Convolutional Neural Network (CNN) to classify the category for that specific crop. Taking crops at the pixel level enables the classification of each pixel, with a sliding window of dimensions like 3 × 3 or 7 × 7 focusing on classifying the centre pixel.

Despite its potential effectiveness, this approach has notable drawbacks. Firstly, it is inefficient as it neglects to reuse shared features between patches. Secondly, it inadequately utilizes spatial information, and the use of different sliding window sizes may result in varied accuracy performances depending on the image's domain. Smaller windows may overlook the broader picture, while larger ones may fail to capture fine details. [1]

2.2 Fully Convolutional Networks

The alternative approach involves applying fully convolutional networks (FCNs) to generate predictions for all pixels. This model employs various blocks of convolution and max-pool layers to initially compress an image to $1/32^{nd}$ of its original size. At this reduced scale, a class prediction is made, followed by utilizing up-sampling and deconvolution layers to restore the image to its original dimensions.

These models are characterized by the absence of fully connected layers. The down-sampling steps are designed to capture semantic and contextual information, while the up-sampling steps aim to recover spatial details. Additionally, there are no limitations on image size, and the final image retains the same dimensions as the original image. To restore fine-grained spatial information lost during down-sampling, skip connections are employed. These connections, bypassing at least one layer, facilitate the transfer of information from the down-sampling to the up-sampling step. The incorporation of features from various resolution levels aids in combining context information with spatial details, effectively addressing the limitations of the naive method.

Although these types of convolutional neural networks outperform the naive approach and operate more efficiently, they may still encounter performance issues. Specifically, if the training data includes images with high-resolution convolutions, maintaining the original resolution throughout every hidden layer can be computationally expensive [2].

2.3 Mask R-CNN

The Faster RCNN algorithm is widely recognized for effective object detection. It operates in two stages, starting with the Region Proposal Network (RPN) suggesting potential object bounding boxes. The second stage, Fast R-CNN, utilizes RoIPool to extract features from each candidate box, performing classification and bounding-box regression. The features are shared between both stages for faster inference.

A related algorithm, Mask R-CNN, builds on Faster R-CNN by introducing a third branch for object masks. While Faster R-CNN outputs class labels and bounding-box offsets, Mask R-CNN adds a distinct output for object masks, requiring finer spatial layout information. This is achieved using the Fully Convolution Network (FCN).

In the Mask R-CNN architecture, object classification and localization involve bounding boxes and semantic segmentation, classifying each pixel into specific categories. Every region of interest is assigned a segmentation mask, with the final output producing a class label and bounding box. This architecture extends the capabilities of Faster R-CNN, combining a deep convolutional network for region proposals with a detector utilizing these regions.

Mask R-CNN merges the Faster RCNN and FCN networks into a unified architecture. It excels in both object detection and instance segmentation, accurately delineating object boundaries at the pixel level through the use of Feature Pyramid Network (FPN) and Region of Interest Align (ROIAlign), ensuring strong performance and accuracy.

Despite its strengths, Mask R-CNN has limitations, including computational complexity and memory usage challenges during training and inference. It may face difficulties accurately segmenting very small objects or handling heavily occluded scenes. Acquiring a substantial amount of annotated training data can be demanding, and fine-tuning the model for specific domains may necessitate careful parameter tuning [3].

2.4 U-Net

The U-Net architecture, derived from the Fully Convolutional Network (FCN), has been tailored to excel with minimal training data, producing more accurate segmentations. It distinguishes itself from FCN through two key differences: its symmetric structure and the use of concatenation, instead of summation, in the skip connections between the down sampling and up sampling paths. The symmetric design, coupled with skip connections, facilitates the transfer of information, and contributes to a large number of feature maps in the up-sampling path.

Comprising of two parts, the U-Net architecture involves a contracting path and an expansive path. The contracting path captures context, while the expansive path aids in precise localization, ultimately generating a segmentation map. The skip connections from the contracting path play a crucial role in retaining spatial information lost during contraction, enhancing the accuracy of feature localization by the decoder layers.

The U-Net architecture proves advantageous for image segmentation tasks in several ways. Firstly, its U-shaped design facilitates the fusion of low-level and high-level features, enhancing object localization. Secondly, the skip connections contribute to preserving spatial information, enabling more precise segmentation. Lastly, the U-Net architecture boasts a relatively small number of parameters, making it computationally efficient compared to alternative architectures.

In applications with limited training images, U-Net stands out as the most suitable architecture, demonstrating effectiveness in capturing context, maintaining spatial information, and enhancing segmentation accuracy.

3. Implementation

3.1 Data Preparation

Data pre-processing involves the crucial task of preparing raw data to make it suitable for a deep learning model. This initial step is pivotal, as the quality of the data directly influences the quality of the model.

For the scope of the project, data preparation encompasses two essential steps. Firstly, it involves preparing the labelled dataset. Additionally, the data is processed through an input pipeline as part of the overall data preparation process.

3.1.1 Creating Labelled Input

To prepare the labelled dataset, it is necessary to have a segmented mask for each image within the training data. This mask distinguishes the crack from the background by highlighting the relevant area, where the crack is represented by $1s$ and the background by $0s$ in binary form.

The Image Segmenter tool in MATLAB was utilized for creating the segmentation mask for each image. This tool incorporates various automatic algorithms that facilitate the effective generation of segmentation masks.

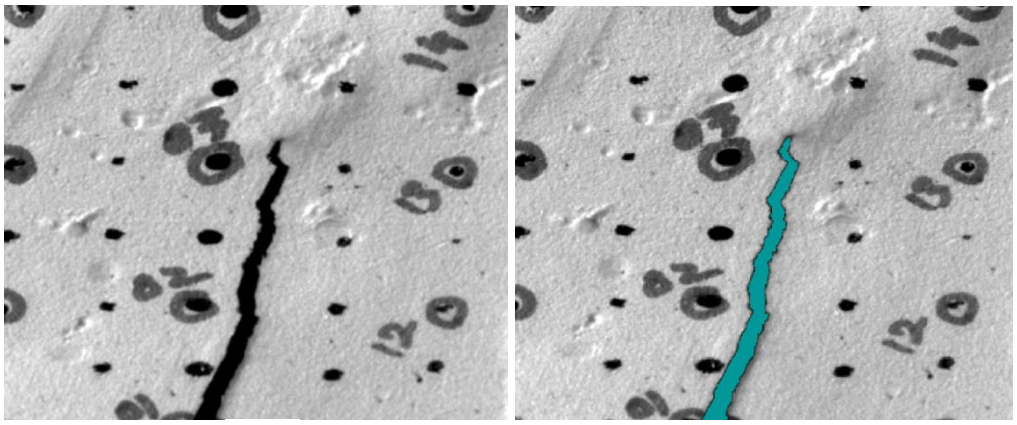
Image segmentation involves dividing an image into distinct parts or regions based on pixel characteristics. For instance, one approach is to identify regions by detecting abrupt discontinuities in pixel values, indicating edges that define these regions. Alternatively, methods may use colour values or texture to divide the image into regions.

The Image Segmenter accomplishes image segmentation by refining these regions. Algorithms utilized to create the segmentation masks include:

i. **Flood Fill:**

   Flood fill is a fundamental technique employed in image segmentation, particularly for applications like creating binary masks for a given image. The primary objective of flood fill is to identify and isolate distinct regions of interest within an image. Floodfill is advantageous in scenarios where regions of interest are characterized by homogenous intensity levels. Its application facilitates the automatic and efficient segmentation of images.

   Flood Fill was used to identify and capture fully connected, visibly discernible cracks characterized by continuous regions and distinct intensity levels. This method proved highly effective in generating binary masks for images that had extensive cracks that were precisely outlined.



(a) (b)

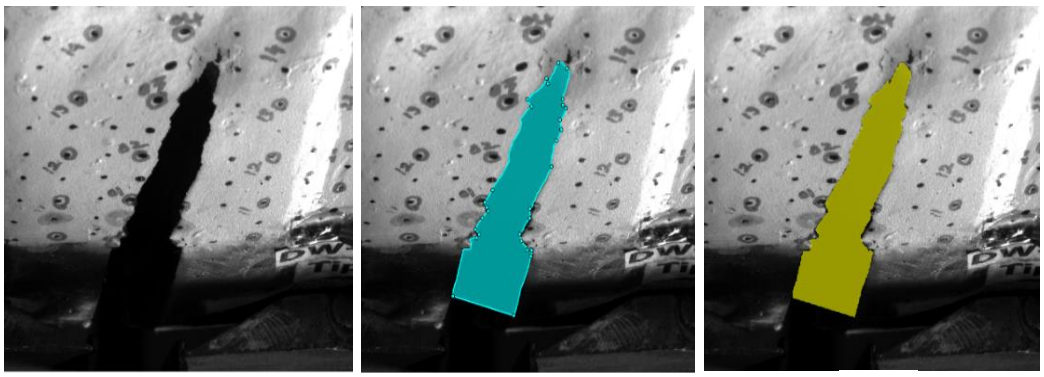
Figure 2: Flood Fill

Fig 2 (a), shows the image before flood fill, and Fig 2 (b), shows image after flood fill in Image Segmenter.

ii. Draw ROIs:

The Region of Interest (ROI) drawing tool in image segmentation, allows users to define specific areas for analysis or manipulation. This tool is particularly useful for selecting regions within an image that are of particular interest for further processing or creating masks. Using Draw ROI different shapes can be drawn or plotted like rectangles, polygons, or even freehand regions onto the displayed image. These drawn ROIs serve as masks or selections enabling easy analysis.

Draw ROI was employed in this particular use case to tackle problems of unclear images and cracks that were unevenly shaped or positioned for the Flood Fill algorithm to identify or create masks.



(a) (b) (c)

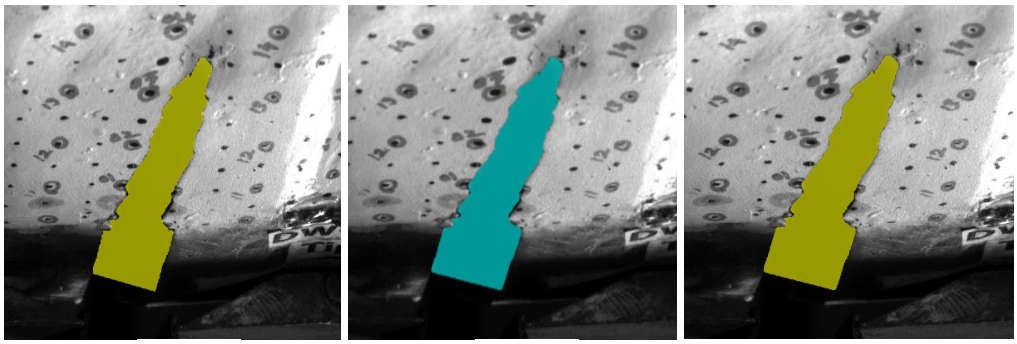
Figure 3: Draw ROI

Fig 3 (a), shows the image before ROI, and Fig 3 (b) shows the drawn boundaries for the mask using Draw ROI, and Fig 3 (c) shows the applied mask on the image in Image Segmenter.

iii. Active Contours:

The active contours method is an automatic and iterative technique for marking locations in an image. Active contours operate based on minimizing an energy function that defines the contour's position, adapting to image features such as edges or intensity gradients. A contour can be initiated near the object of interest, and the algorithm iteratively evolves the contour to align with prominent features in the image. This process allows for precise and fine-tuned adjustments to capture the desired object boundaries accurately. The Active Contours tool is particularly beneficial when dealing with complex object shapes or when precise boundary delineation is essential.

The Active Contour algorithm was utilized for refining the segmentation results by addressing gaps and making minor adjustments between manually drawn or selected regions of interest and the actual boundaries of cracks.



(a) (b) (c)

Figure 4: Active Contours

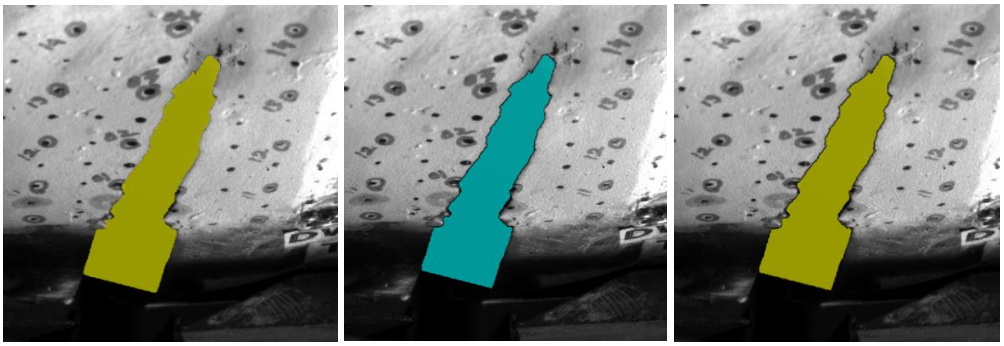
Fig 4 (a), shows the masked image after Draw ROI, Fig 4 (b), shows the expanded region based active contour on the image, and Fig 3 (c) shows the applied mask on the image in Image Segmenter.

iv. Morphology:

The Morphology tool is a primary tool in image segmentation applications, offering powerful operations for analysing and manipulating the shape and structure of objects within images. Morphology encompasses a wide range of image processing techniques that operate based on shapes. During a morphological operation, each pixel's adjustment depends on the values of the neighbouring pixels. Morphological operations, such as dilation, erosion, opening, and closing, are commonly applied to enhance or suppress certain features, refine object boundaries, and extract relevant information.

The erosion operation of morphology was used to modify the shape and structure of the object. It was employed to define shapes, and refine and fine-tune the segment mask boundaries. The erode morphological operation was used to

eliminate pixels at the object boundaries. This operation proved to be beneficial in closely fitting the mask to the crack boundaries, followed by active contours.



(a) (b) (c)

Figure 5: Morphology

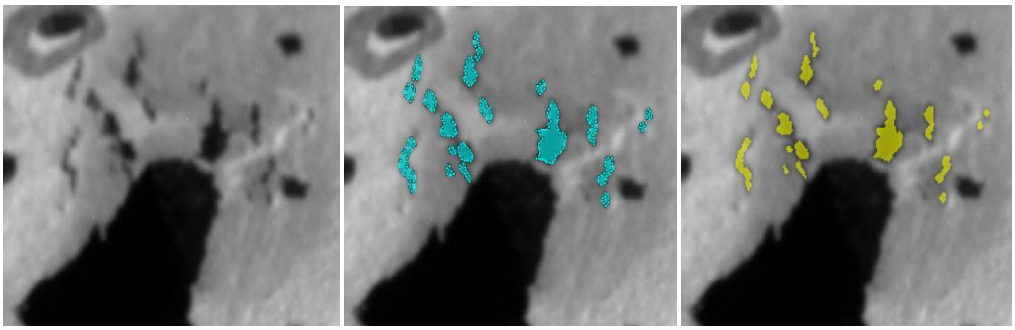
Fig 5 (a) shows the masked image after active contour, Fig 5 (b) shows the erode mask morphology on the image, and Fig (c) shows the applied mask on the image in Image Segmenter.

These multiple segmentation methods offered by Image Segmenter were used in an iterative process to generate highly precise segmentation masks for each image. Several methods were layered to achieve the desired results.

Additionally, the app allows users to view the binary image of the created mask, making it easier to analyse and understand as it represents the intended output.

Regarding the objective of the project, the segmentation mask for each image comprises of two parts: the primary crack, including the tip of the crack and the secondary cracks.

Flood fill and Draw ROIs are the two algorithms utilized to mask the secondary cracks in the images.



(a) (b) (c)

Figure 6: Secondary Cracks

Fig 6 (a) shows the image before masking, Fig 6 (b) shows the drawn boundaries for the mask using Draw ROI, and Fig 6 (c) shows the applied mask on the image in Image Segmenter.

3.1.2 Creating Input Pipeline

The purpose of the input pipeline is to link each image with its segmented mask and convert the data into a suitable format for processing.

Input pipelines are crucial for any deep learning experiment because learning models need data in a TensorFlow-friendly format. Creating efficient pipelines is straightforward with the tf.data.Dataset abstraction, a component of the tf.data API, as it represents a sequence of elements from a dataset in a simple format. [4]

The pipeline starts by reading each input image and its corresponding masked image from the relevant file directories. These data values are then transformed into tensors for smooth pre-processing and training. Subsequently, each image is paired with its respective masked image for easy access.

To enhance the size and feature variance of the dataset, each image is repeated 16 times using the repeat method, which duplicates tensors a specified number of times in the dataset.

The two main operations of the pipeline involve cropping and data augmentation.

i. Cropping:

To manage the computational load of the model and address class imbalance, the input data is cropped. Image moments are employed to crop each image around its centroid. Utilizing the crop function, each image is transformed into a fixed size of 1024×1024 for standardization.

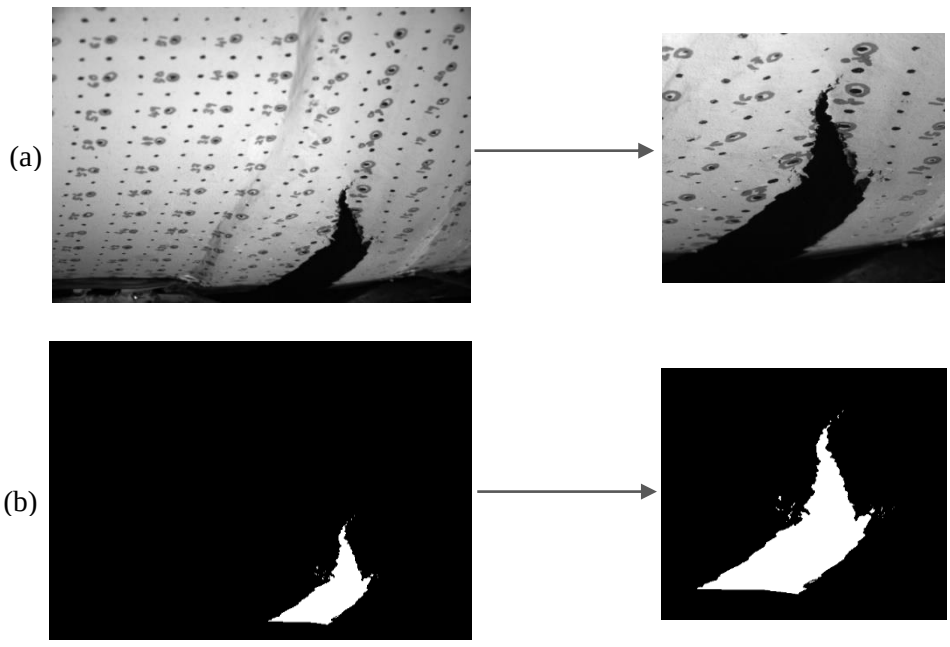
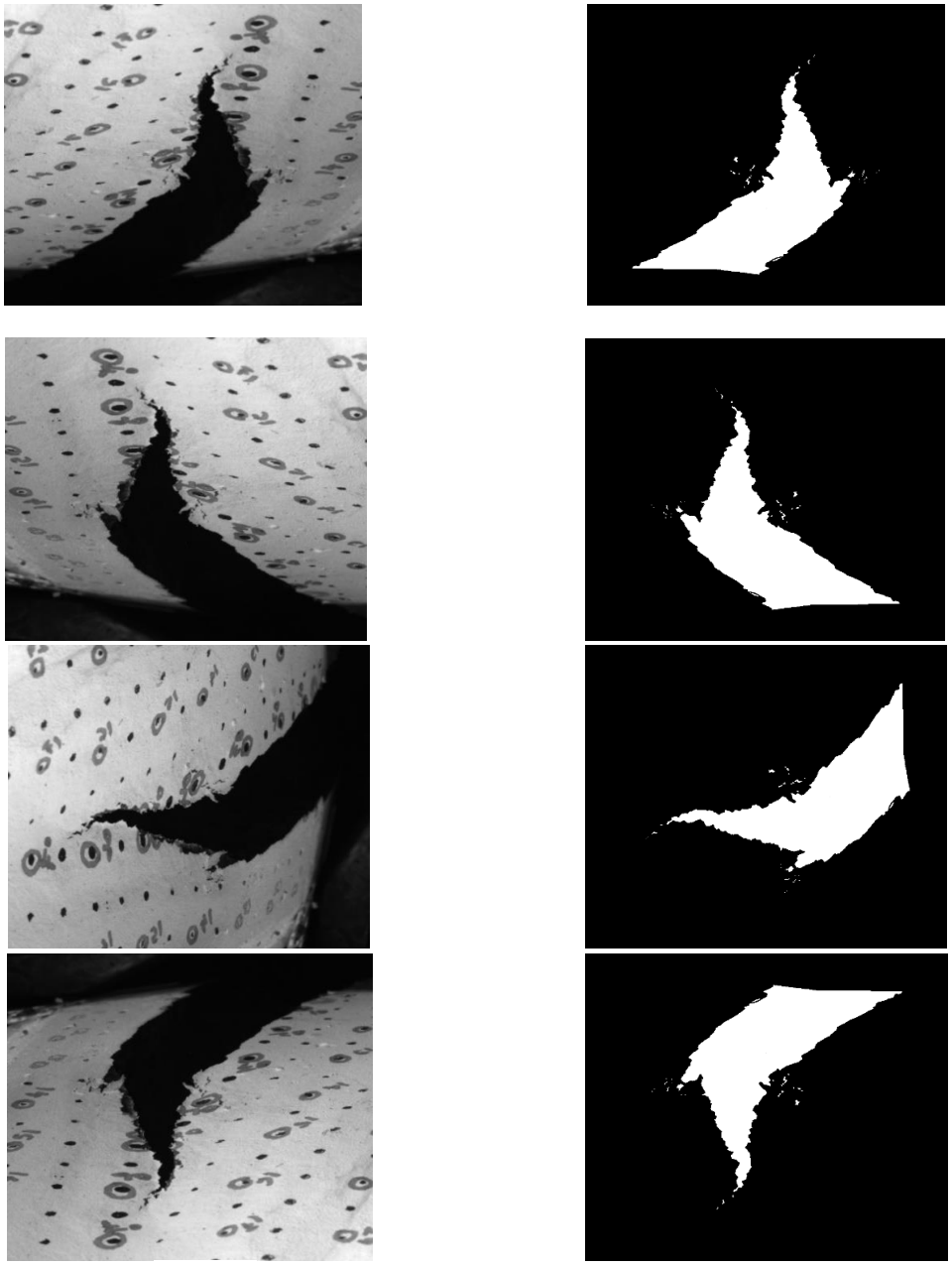


Figure 7: Cropping

Fig 7 (a) shows the original and its crop for the input image, Fig 7 (b) shows the original and its crop for the mask image.

ii. Data Augmentation:

Data augmentation plays a crucial role in instilling the desired invariance and robustness properties in the network, especially when there are limited training samples available. To introduce augmentation in the repeated dataset, each image undergoes random flipping and/or rotation.



(a) (b)

Figure 8: Data Augmentation

Fig 8 (a) shows the original and its augmentations for the input image, Fig 8 (b) shows the original and its augmentations for the mask image.

In the final step, the data values undergo batching and prefetching to enhance the efficiency of TensorFlow input pipelines. Batching involves grouping multiple data elements into a single batch, while prefetching involves retrieving data from disk or a database in the background during model training.

Batching and prefetching contribute to improved model performance by ensuring that the data is readily available when needed. Prefetching specifically streamlines the process by overlapping the pre-processing and model execution of a training step. As the model executes training step s, the input pipeline concurrently reads the data for step $s + 1$. This approach minimizes the step time to the maximum, rather than the sum, of the training and data extraction times.

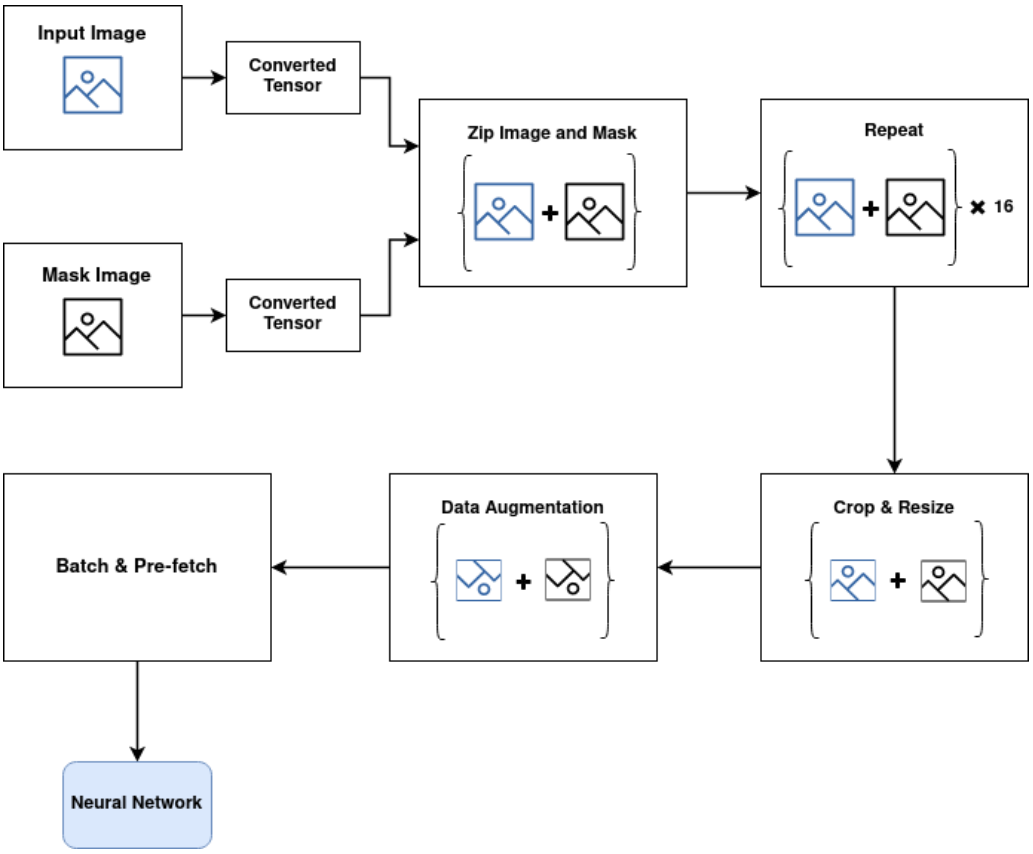


Figure 9: Data Preparation Input Pipeline

3.2 Network Architecture

The U-Net architecture, designed for image segmentation applications, employs a dual-path approach comprising an encoding (contracting) path and a decoding (expanding) path. The unique U-shaped design enhances the network's ability to capture both local and global features, resulting in precise segmentation outcomes.

In the contracting path, convolutional layers and max pooling operations are used to capture detailed, low-level characteristics by progressively reducing the spatial dimensions of the input image. The expanding path utilizes transposed convolutions for up-sampling, allowing the reconstruction of a dense segmentation map.

Key components of the U-Net architecture include skip connections, which link corresponding layers between paths to gather both local and global data. Concatenation is used during up-sampling, combining feature maps from the encoding path with up-sampled feature maps from the decoding path to leverage multi-scale information.

The absence of fully connected layers and the use of exclusively convolutional layers make U-Net adaptable to images of varying sizes while preserving spatial information. The contracting path plays a vital role in extracting high-level information from the input image and gradually shrinking spatial dimensions. The network architecture, as depicted in Figure 10, is composed of two main components: a contracting path (left side) and an expanding path (right side). The contracting path follows the standard structure of a convolutional network, featuring a series of repeated steps. Each step involves consecutive 3×3 convolutions, each followed by a rectified linear unit (ReLU). Additionally, a 2×2 max pooling operation with a stride of 2 is applied for down-sampling. Notably, the number of feature channels is doubled at each down-sampling step. Every step in the expansive path consists of an up-sampling of the feature map followed by a 2×2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly copied feature map from the contracting path, and two 3×3 convolutions, each followed by a ReLU. At the final layer of the network, a 1×1 convolution is employed to map each 2-component feature vector to the desired number of classes. In total, the network has 11 convolutional layers [5].

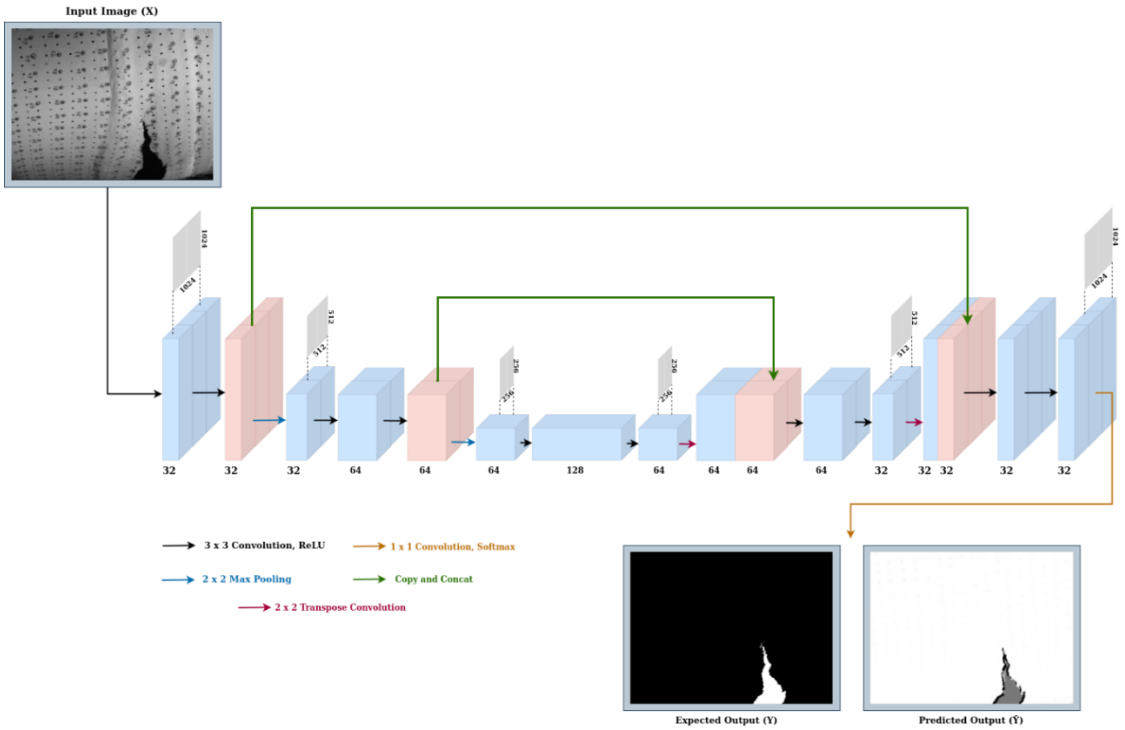


Figure 10: Network Architecture

3.2.1 Activation Functions

In this architecture the input images and their corresponding segmentation maps are used to train the network.

The Rectified Linear Unit (ReLU) activation function is applied after each convolutional layer to enhance the network's ability to learn complex patterns in data. ReLU introduces non-linearity, crucial for capturing intricate relationships, and is computationally efficient, making it faster during training. It also encourages sparsity in activation, reducing computational load. ReLU addresses the vanishing gradient problem, promoting faster convergence in deep networks.

The ReLU activation is defined as:

$$f(x) = \max(0, x)$$

Where, $f(x)$ is the output of the ReLU function for a given input x, and $\max(0, x)$ denotes that the output is the maximum of zero and the input x. Thus, if the input x is positive or zero, the output is equal to x. However, if the input x is negative, the output becomes zero.

This simple yet effective function replaces negative values with zero, facilitating quicker training and improved learning rates

The final layer of the architecture utilizes the soft-max activation function. To calculate the energy function, a pixel-wise soft-max is applied to the last feature map, and this outcome is then combined with the sparse categorical cross-entropy loss function. The soft-max activation is defined as:

$$p_k(x) = \exp(a_k(x)) / \left(\sum_{k'}^{K} \exp(a_{k'}(x))\right)$$

Where, $a_k(x)$ denotes the activation in feature channel k at the pixel position $x \in \Omega$ with $\Omega \subset \mathbb{Z}^2$. K is the number of classes and $p_k(x)$ is the approximated maximum-function. I.e. $p_k(x) \approx 1$ for the k that has the maximum activation $a_k(x)$ and $p_k(x) \approx 0$ for all other k.

3.2.2 Parameters

The U-Net-based architecture is trained on a set of 353,250 parameters, showcasing its capacity to learn intricate patterns and representations. [6] One of its notable features is its adaptability to process images of varying sizes for generating outputs.

Layer (type)	**Output Shape**	**No of Filters**	**No. of Parameters**
Input Layer	(None, 1024, 1024, 1)		0
Convolution Layer (Con2D)	(None, 1024, 1024, 32)	32	320
Convolution Layer (Con2D)	(None, 1024, 1024, 32)	32	9248

Max Pooling Layer (MaxPooling2d)	(None, 512, 512, 32)		0
Convolution Layer (Con2D)	(None, 512, 512, 64)	64	18496
Convolution Layer (Con2D)	(None, 512, 512, 64)	64	36928
Max Pooling Layer (MaxPooling2d)	(None, 256, 256, 64)		0
Convolution Layer (Con2D)	(None, 256, 256, 128)	128	73856
Convolution Layer (Con2D)	(None, 256, 256, 64)	64	73792
Transposed convolution layer (Con2DTranspose)	(None, 512, 512, 64)	64	16448
Concatenate Layer (Concatenate)	(None, 512, 512, 128)		0
Convolution Layer (Con2D)	(None, 512, 512, 64)	64	73792
Convolution Layer (Con2D)	(None, 512, 512, 32)	32	18464
Transposed convolution layer (Con2DTranspose)	(None, 1024, 1024, 32)	32	4128
Concatenate Layer (Concatenate)	(None, 1024, 1024, 64)		0
Convolution Layer (Con2D)	(None, 1024, 1024, 32)	32	18464
Convolution Layer (Con2D)	(None, 1024, 1024, 32)	32	9248
Convolution Layer (Con2D)	(None, 1024, 1024, 2)	2	66

Total number of trainable parameters: 353,250

3.3 Training Experiment

3.3.1 Training Scheme

In order to train a highly effective and efficient neural network, careful consideration was given to the selection of optimization techniques and training strategies used, to ensure optimal performance. The selection of Adam Optimizer was particularly justified

given its suitability for large-scale problems, making it well-suited for the challenges posed by datasets with a substantial volume of data and parameters.

Implemented within a custom training loop, the model went through an iterative training over the course of 30 epochs. This training loop contained dynamic adjustments to the learning rate and periodic evaluations on the validation dataset. Carefully observing and evaluating how the model performed at each epoch provided valuable insights, allowing for a greater understanding of how the model training unfolded.

For the evaluation of the model's effectiveness in the challenging classification tasks, the Sparse Categorical Cross Entropy (Sparse CCE) Loss function was adopted. This loss function extension, was a great choice for datasets where output labels are encoded in a sparse matrix format, and provided a comprehensive metric for assessing the model's classification accuracy and performance.

To ensure a thorough examination of the model's generalization abilities, a strategic random allocation of 16% of the input data was designated for validation during the training process. This practice allowed for an unbiased assessment of the model's performance on a set of data not used in the training phase, contributing to a more reliable understanding of its capabilities.

The adoption of mini-batch Gradient Descent with a batch size of 32 was a key aspect of the training methodology. This approach enhanced training efficiency by processing data in smaller, more manageable batches rather than tackling the entire dataset in one go. This was particularly beneficial for large datasets, as it allowed for quicker convergence and more efficient updates to the model's parameters.

A crucial component of the training scheme involved Learning Rate Scheduling, a dynamic technique used to adjust the learning rate during the training process. Implemented using the Piecewise Constant Decay class in Keras, the learning rate was systematically adjusted over the 30 epochs. Specifically, it started at $1e^{-3}$ for the initial 20 epochs, followed by reductions to $1e^{-4}$ for epochs $21 - 25$, and a further decrease to $1e^{-6}$ for epochs $26 - 30$. This strategic modulation of learning rates addresses challenges such as convergence issues, oscillations, and overshooting, contributing to a more stable and efficient optimization process.

To sum up, the holistic training approach, characterized by careful selection of optimizers, thoughtful choice of loss functions, implementation of mini-batch Gradient Descent, and dynamic adjustments in learning rates, helped in training the neural network with the adaptability and efficiency needed in order to in achieving superior classification results, particularly on extensive and intricate datasets.

3.3.2 Optimizer

Optimization algorithms are used in the training phase of deep learning, where the model is presented with a batch of data, the gradients are calculated, and the weights and biases are updated using an optimization algorithm. Once the model has been trained, it can then be used for inference on new data.

Stochastic gradient descent (SGD) algorithms are the main optimization techniques used to train deep neural networks. These algorithms can be divided into two categories: adaptive learning rate methods (e.g., Adam and AdaGrad) and accelerated schemes

(e.g., Nesterov momentum). SGD is the dominant optimization algorithm in deep learning, which is simple and performs well across many applications [7].

Adam is a stochastic gradient descent algorithm based on the estimation of 1st and 2nd-order moments. The algorithm estimates the 1st-order moment (the gradient mean) and 2nd-order moment (element-wise squared gradient) of the gradient using the exponential moving average, and corrects its bias. The final weight update is proportional to the learning rate times the 1st-order moment divided by the square root of the 2nd-order moment. [8]

Adam was deployed as the optimizer while training the model as it is computationally efficient, has little memory requirements, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters., it inherits the good features of RMSProp and other optimizer algorithms. The results of the Adam optimizer are generally better than every other optimization algorithm and have faster computation time. As a result, Adam is recommended as the default optimizer for most of the applications.

Adam served as the optimizer during model training, offering computational efficiency, low memory requirements, being invariant to diagonal rescaling of gradients and compatibility with large-scale data/parameter problems. It inherits positive features from RMSProp and other optimization algorithms. Generally, the results obtained using the Adam optimizer outperform those of other optimization algorithms, showcasing its reliability and faster computational times. Consequently, Adam is recommended as the default optimizer for a variety of applications.

3.3.3 Loss Function

The loss function serves as a metric to quantify the disparity between the predictions generated by the model and the desired or expected outcomes. It essentially measures how well the model represents the data in its current state.

The model's effectiveness was assessed using the Sparse Categorical Cross Entropy (Sparse CCE) Loss function, a standard metric for evaluating classification tasks. It is an extension of the Categorical Cross Entropy (CCE) Loss and is particularly well-suited for datasets where output labels are encoded in a sparse matrix format.

In Cross-Entropy Loss, each probability of the predicted class is compared with the actual class and loss is calculated which penalizes the probability based on how far it is from the actual expected value. The penalty is logarithmic in nature yielding a large score for large differences close to 1 and small score for small differences tending to 0. A perfect model would have a cross-entropy loss of 0. Cross Entropy is defined as:

$$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i)\,, for\ n\ classes$$

Where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

In Categorical Cross Entropy, the target labels are expected to be in one-hot encoded format, where each label is represented as a vector with all elements being zero except for the index corresponding to the true class, which is set to one. However, in some cases, especially when dealing with large number of classes, representing labels in a

one-hot encoded format can be computationally inefficient and memory-intensive. Sparse Categorical Cross Entropy addresses this by allowing the target labels to be provided as integers, where each integer corresponds to the class index. This is particularly useful when the output space is large and most of the entries in the target vector are zero. The main difference between CCE and Sparse CCE is on how labels are defined.

Categorical cross-entropy is used when we have to deal with the labels that are one-hot encoded, for example, we have the following values for 3-class classification problem [1,0,0], [0,1,0] and [0,0,1].

In sparse categorical cross-entropy, labels are integer encoded, for example, [1], [2] and [3] for 3-class problem.

By directly working with integers, Sparse CCE streamlines the storage requirements and computational processing, resulting in a more resource-efficient approach.

3.3.4 Validation

Model validation is the process of evaluation of a trained model using a dedicated validation dataset, derived from the training dataset. This validation dataset is randomly extracted from the original training dataset, for assessing the model's generalization capabilities. Essentially, it offers an unbiased evaluation that aids in estimating the model's proficiency. This evaluation takes continuously by modifying the models hyperparameters.

Partitioning data for validation is a crucial step, as the validation set enables the model to gain insights into previously unseen data. While calculating training loss assesses the model's prediction capabilities on the training or already seen data, validation loss is systematically calculated at each step, to see model's prediction capabilities on new or unfamiliar data. Advancements in both training and validation losses are crucial and essential during the training process. [9]

Ideally, validation data splits typically range between 10% to 20% of the training dataset. In this particular case, 16% of the input data was randomly allocated for validation during the training process. Subsequently, the selected validation subset along with the training dataset was used to calculate the training and validation loss, as well as training and validation accuracy at the completion of each epoch.

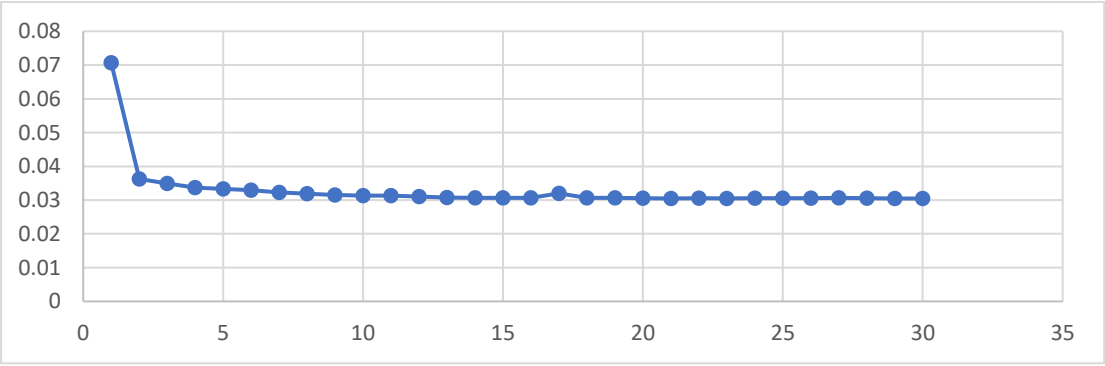


Figure 11: Training Loss

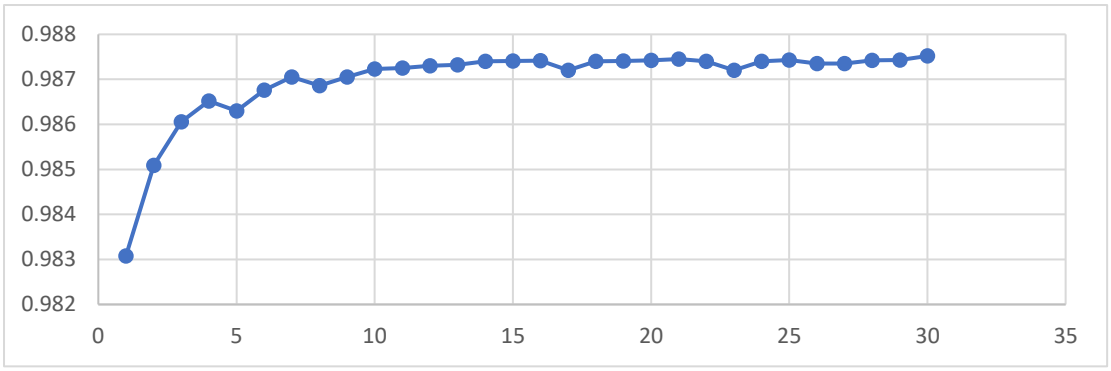


Figure 12:Training Accuracy

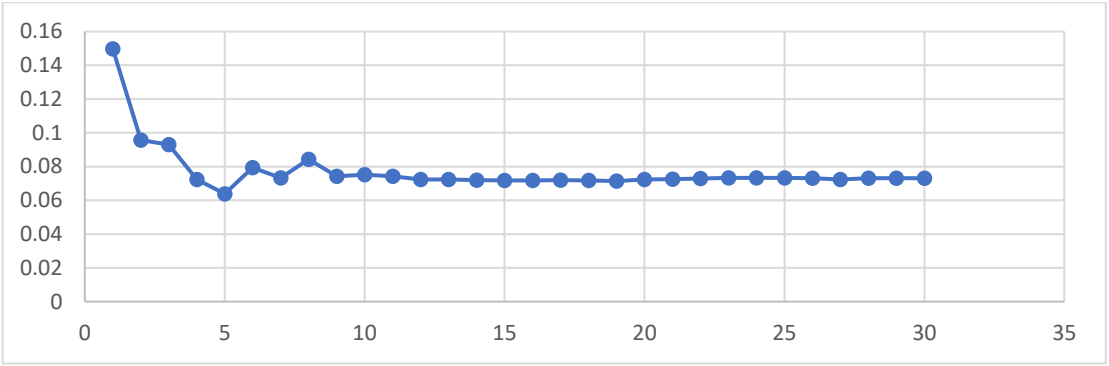


Figure 13: Validation Loss

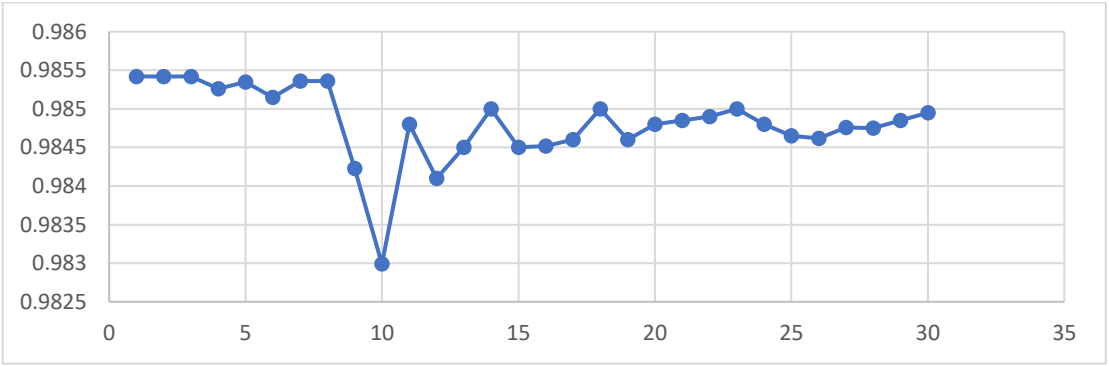


Figure 14: Validation Accuracy

3.3.5 Batch Size

The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.

A batch can be thought of as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables, and an error is calculated. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient.

A training dataset can be divided into one or more batches. When all training samples are used to create one batch, the learning algorithm is called batch gradient descent.

When the batch is the size of one sample, the learning algorithm is called stochastic gradient descent. When the batch size is more than one sample and less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent.

For the trained model, Mini Gradient Descent mode is used, with the batch size being 1. Batch is taken as 1, as the model and data size are computationally heavy.

3.3.6 Learning Rate

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process.

The learning rate is one of the most important hyperparameters when configuring and training a neural network, therefore it is vital to know how to investigate the effects of the learning rate on model performance and to build an intuition about the dynamics of the learning rate on model behaviour. In gradient descent learning algorithms the learning rate parameter can have a significant effect on generalization accuracy. In particular, decreasing the learning rate below that which yields the fastest convergence can significantly improve generalization accuracy, especially on large, complex problems. The learning rate also directly affects the training speed [10].

During the training process, Learning Rate scheduling is used to customize the learning rate after a certain number of epochs. Learning rate scheduling is a technique used in training machine learning models to dynamically adjust the learning rate during the training process. The learning rate is a hyperparameter that determines the step size at which the model's parameters are updated during optimization. An optimal learning rate is crucial for efficient and effective model training. Learning rate scheduling helps address challenges such as convergence issues, oscillations, and overshooting that can occur when using a fixed learning rate. This is achieved by using the Piecewise Constant Decay class in Keras. [11]

For epochs, 1 to 20 the Learning Rate was set to $1e^{-3}$, for 21- 25 it was set to $1e^{-4}$ and for $26 - 30$ it was set to $1e^{-6}$.

4. Discussion and Conclusion

4.1 Challenges Faced

While developing the Sematic Segmentation model using the U-Net Architecture several challenges were encountered that had notable effects on the model's predictive performance, training requirements and its overall capabilities.

i. **Pixel-Intensive Image Processing:**
Handling images with an extensive number of pixels poses computational challenges, leading to increased memory usage and potential performance bottlenecks. In this case, all the input images were of size 2048×2560, resulting in slowing down the overall workflow and affecting the performance.

ii. **Resource-Intensive Model Architecture:**
Models with a huge number of parameters require large number of computational resources, resulting in straining the current hardware and increasing the training time and deployment complexities. As a consequence, there is a need to strategically optimize the model input and its parameters. The initially proposed U-Net architecture had over 1.5 million parameters making it completely necessary to use powerful workstations for effective execution.

iii. **Class Imbalance Bias:**
Uneven class distribution in the training data can cause biased model predictions, with the minority classes being underrepresented, resulting in an inaccurate model. In this case, the input data had a greater number of background pixels, when compared to the foreground pixels. This resulted in training data class imbalance affecting the overall model performance.

Solutions Applied:

Initially, to address the problem extensive number of parameters, the model architecture was modified by reducing the feature channels of all the convolution layers by half, which in return resulted in reducing the total number of parameters from over 1.5 million to $353{,}250$. This drastically helped in improving the model's performance and training time, by putting less strain on the available computational resources.

To tackle the problems of large image size and unbalanced classes, a comprehensive solution was applied, which involved reducing the image size and cropping it based on the centroid of the segment mask with the help of image moments.

In computer vision and image processing, image moments are often used to characterize the shape of an object in an image. These moments capture basic information such as the area of the object, the centroid (i.e. the centre (x, y)-coordinates of the object), the orientation, and other desirable properties. The centroid for a given image is simply the arithmetic mean position of all the points in a given object (segment mask in this case).

In terms of image moments centroid is calculated as:

$$Centroid\ \{\bar{x}\ ,\bar{y}\ \} = \left\{\frac{M_{10}}{M_{00}}\ ,\frac{M_{01}}{M_{00}}\right\}$$

Where,

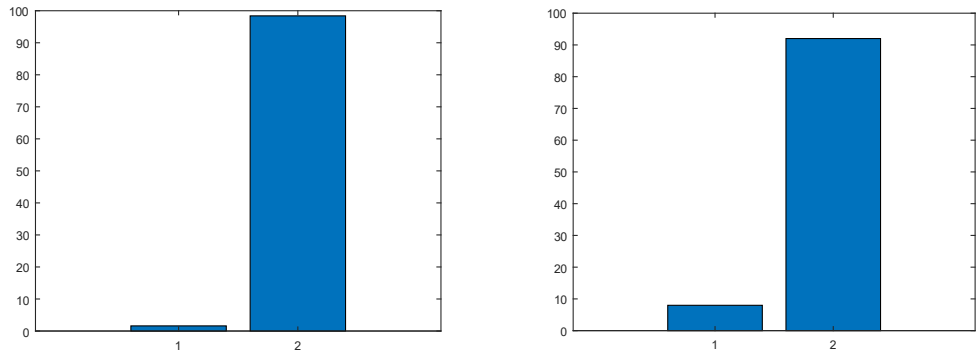
$$M_{ij} = \sum_{x} \sum_{y} x^i y^i \; I(x, y)$$

Here, x, y refers to the row and column index and $I(x, y)$ refers to the intensity at the location (x, y).

The centroid, calculated using moments and pixel intensities, serves as the centre of mass, guiding the spatial location for cropping to ensure a more balanced representation.

The calculated values of cx and cy or the $(\bar{x}, \bar{y})$ determine the centre point of the segment mask serving as the starting coordinates for the cropping process. Bounding-box crop technique was used to apply crop and then resize the segment masks and their corresponding input images to a standardized size of 1024×1024. This method ensured the consistency in dimensions, and uniform processing throughout the dataset.

By cropping images based on their centroids, the class imbalance is effectively reduced as shown in Figure 15 & 16, ensuring a better representation during model training. Additionally, this process aids in resizing images to lower dimensions, which not only contributes to effective and low memory utilization but also plays a vital role in enabling the resource-intensive model to execute seamlessly, even when dealing with a high number of parameters.



(a) (b)

Figure 15: Class Imbalance

Fig 15 (a) shows Class Imbalance before cropping and Fig 15 (b), shows Class Imbalance after cropping.

Effectively and strategically cropping and resizing the images using centroid enables us to address major challenges faced and improve the overall adaptability and robustness of the model.

4.2 Results

The U-Net neural network architecture was specifically designed for semantic segmentation tasks, with a focus on identifying cracks on industrial pipes through CNN (Convolutional Neural Networks). The model underwent training, involving over 300 thousand parameters, leading to an impressive average accuracy of 97.62 *per cent*.

It's worth noting that the model exhibits varying performance on different images, yielding diverse results based on the input provided. The accuracy scores for a selection of randomly chosen images are detailed in Table 1.

Table 1: Accuracy Comparison Table

Image Index	**Accuracy**	**Accuracy in per cent**
Image 1	0.9995924	99.59
Image 172	0.9877025	98.77
Image 218	0.7829588	78.29

Figure 16: Input, Mask and Prediction for Image Index 1

Figure 17: Input, Mask and Prediction for Image Index 172

Figure 18: Input, Mask and Prediction for Image Index 218

In conclusion, the U-Net-based neural network demonstrates a robust capability for semantic segmentation in identifying cracks on industrial pipes, achieving a great average accuracy across a substantial parameter set. The varied performance on different images suggests the model's adaptability to diverse scenarios, showcasing its versatility in real-world applications. As illustrated in Table 1 and the corresponding result images, the model exhibits a commendable performance in detecting and delineating cracks, contributing to its potential as an effective tool in industrial inspection and quality assurance. Further refinements and fine-tuning may enhance the model's generalization across a broader spectrum of images and scenarios, fostering its continued utility in addressing challenges related to pipeline integrity and maintenance.

5. Future Work

Building upon the current implementation of Sematic Segmentation using the U-Net Architecture for identifying cracks, several directions can be taken to approve and enhance the model's predictions and address the current problems and limitations.

- Expanding the dataset is essential to enhance the model's generalization capabilities. This includes acquiring additional input images and utilizing data augmentation techniques to introduce diversity, thereby improving the overall robustness of the model.
- Explore the influence of varying batch sizes on training efficiency. Larger batches can enhance computational efficiency by processing more data in parallel, leading to faster updates of the model's parameters.
- Fine-tuning the model by hyperparameter optimization. It includes tweaking of parameters, learning rate and numbers of filters. This adjustment is aimed at figuring out how well the model can pick up on the finer details of the image, for accurate segmentation tasks.
- Prioritizing the mitigation of class imbalance. Trying techniques like class-weight adjustments during training
- Implementing multi-GPU training to scale up the model's capacity for handling larger datasets and more complex segmentation tasks. By distributing the computational load across multiple GPUs, the training process can be significantly accelerated, allowing for faster computation and improved efficiency. Additionally, optimizing the communication between GPUs and addressing any potential challenges in synchronization will be crucial for harnessing the full potential of parallel processing in training the U-Net architecture for semantic segmentation.

References

[1] Y. Z. Y. G. Shijie Hao, “A Brief Survey on Semantic Segmentation with Deep Learning,” Neurocomputing, 2020.

[2] E. S. T. D. Jonathan Long, “Fully Convolutional Networks for Semantic Segmentation,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[3] G. G. P. D. R. G. Kaiming He, “Mask R-CNN,” in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017.

[4] D. Paper, “Build TensorFlow Input Pipelines,” in State-of-the-Art Deep Learning Models in TensorFlow, 2021.

[5] O. Ronneberger, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” Medical Image Computing and Computer-Assisted Intervention, vol. 9351, 2015.

[6] C. M. H. e. al, “Deep learning for undersampled MRI reconstruction,” Physics in Medicine & Biology, vol. 63, 2018.

[7] A. Mortazi, “Selecting the best optimizers for deep learning–based medical image segmentation,” Artificial Intelligence in Radiology , 2023.

[8] J. B. Diederik P. Kingma, “Adam: A Method for Stochastic Optimization,” International Conference on Learning Representations, 2014.

[9] H. Z. Haiying Wang, “Model Validation, Machine Learning,” Encyclopedia of Systems Biology, 2013.

[10] T. M. D.R. Wilson, “The need for small learning rates on large problems,” International Joint Conference on Neural Networks, vol. 01, 2001.

[11] Z. L. Aston Zhang, “Learning Rate Scheduling,” in Dive into Deep Learning.