Ingénierie des Systèmes d'Information
Vol. 30, No. 2, February, 2025, pp. 495-503
Journal homepage: http://iieta.org/journals/isi

Encoder-Decoder Architectures for Crack Detection on Surfaces: A Deep Learning Approach

Sakshi[1], Richa Vijay[2], Sachin Lodhi[3], Ajit Noonia[4*], Gagandeep Berar[5], Ajay Kumar[4]

[1] Amity Institute of Information and Technology, Amity University, Noida 201313, India
[2] School of Computer Science and Engineering, IILM University, Greater Noida 201306, India
[3] Department of Computer Science and Engineering, California State University, Fullerton, CA 92821, USA
[4] Department of Computer Science and Engineering, Manipal University Jaipur, Jaipur 303007, India
[5] Department of Computer Science and Engineering, Chitkara University Institute of Engineering and Technology, Chitkara University, Rajpura 140401, India

Corresponding Author Email: ajit.noonia@jaipur.manipal.edu

https://doi.org/10.18280/isi.300221

Received: 23 December 2024
Revised: 15 January 2025
Accepted: 14 February 2025
Available online: 27 February 2025

Keywords:
crack detection, neural networks, civil engineering, image segmentation, deep learning, innovative technology

ABSTRACT

The building and maintenance of structures like roads, expressways, buildings, skyscrapers, and so forth requires a sizable workforce as well as a sizable financial investment in the fields of civil engineering and construction. Following completion, it is necessary to regularly check for deformations such as fractures, the structure's outermost layer peeling off, rusting, etc. To maintain the safety of both human life and the building itself, such deformations must therefore be continuously monitored and repaired. We present a neural network-based method to find cracks in such structures in addition to physical inspection. Using 2500 images as training data, the model had an accuracy of 95.0%; on the validation set, it had a mean IoU score of 83%. The proposed method also demonstrates superior performance, achieving a 15% increase in prediction accuracy when compared to state-of-the-art methods, thereby illustrating its worth in real-time applications.

1. INTRODUCTION

This decade has seen the fastest rate of progress out of all those studied. Numerous buildings have been built and continue to be built. By building things like roads, buildings, dams, and other infrastructure, emerging countries are catching up to the rest of the globe [1]. Each of these structures took a great deal of capital and pure human labor to create. Pumping such resources would only secure the successful completion of the building projects for these structures; however, upkeep is a critical area in which resources should be expanded [2].

These built structures are vulnerable to a variety of variables, including the environment, chemicals, and man-made causes, and they could sustain harm as a result. Damage to these constructions includes not only the loss of previously expended time and money but, in some situations, the loss of lives if the structure suddenly falls and fractures [3]. Therefore, it is necessary to continually find and fix such defects in these structures. The most typical defects in a cement-based structure may be termed the cracks. In most situations, the cracks get wider with time, which makes the structures weaker in the long run and increases the risk of collapse if the cracks are not fixed.

There are manual methods that require labor to find and repair such cracks. Human eyesight is limited in its ability to detect objects or areas of interest (ROI), and its detection range, which ranges from the lower constraint of 25 cm to the upper bound of infinity, is very constrained (depends on various factors like size of the objects) [4]. If the range of deformity extends well beyond these boundaries at both ends, the manual approaches will not work [5].

Crack detection is a critical task in infrastructure maintenance, where early identification of structural damages can prevent catastrophic failures and reduce maintenance costs. Traditional methods, such as manual inspections and rule-based algorithms, often suffer from low accuracy, inconsistency, and inefficiency in large-scale applications. Recent advances in deep learning have revolutionized image-based analysis by providing robust feature extraction and pattern recognition capabilities, making it a promising solution for crack detection. Moreover, deep learning techniques can handle diverse and complex crack patterns while offering real-time processing capabilities, addressing the limitations of existing approaches [6]. As opposed to human vision, which is limited by characteristics like speed, duration, and range of vision, automatic detection systems are relatively resilient to such restrictions [7]. Deep learning has shown promise in crack detection, but it often requires extensive labeled data. The proposed SS-CCDN utilizes a semi-supervised approach with a multi-task model and memory module, enhancing efficiency and accuracy in detecting concrete cracks [8].

Thus, this study contributes to the field of automated deformities detection in structures, ensuring more accurate and precise detection of such deformities. To ensure the correct detection and upkeep of the structures, the article introduces strategies that have been addressed in the civil domain. The cost calculation to restore the damaged architecture is another advantage over the manual technique that this article highlights in addition to the attained accuracy and time savings. This would eliminate the need for human efforts to list all the costs necessary to fix the structure. The described method would save a significant amount of time compared to its counterpart methodology, which is labor-intensive and may still produce some inaccurate results.

2. MOTIVATION

2.1 To prevent mishappenings

The main driving force behind the authors' decision to produce this work is their desire to avoid any accident or misfortune by identifying structural flaws like cracks [9]. Although surface cracks may seem natural, their internal growth could pose a threat to structures including nuclear power plants, dams, bridges, pillars, and others [10]. In the case of the large structure stated earlier, these malformations can be regarded as the most serious ones. The authors' method would identify the critical flaws in advance so that the appropriate safety measures and repair work may be done to avert the potential accident [11].

2.2 Producing more feasible options

The structures, small and large both, are built with complexities in today's world. The addition of the new facilities also contributes to the increased complexity of these structures. Because of such persisting complexity, the manual inspection of such structures is not feasible by sole human efforts [12, 13]. But there are various small-scale devices like small robot cars and other small-sized robots that can go to places inside a structure where human reach is not possible. So, integrating such devices with the method proposed here would facilitate detecting deformities [14]. The proposed work would contribute as another element to the array of feasible options to inspect a structure.

2.3 Workers' safety insurance

The vast reach of the structures makes it very hard to involve humans in various works involved in building and maintaining these structures [15]. The factors like location, weather, and physical condition of the structure make it very hard to ensure the safety of a human participating in the task of repairing work. Putting human efforts to detect, count, and estimate the required expenditure to repair structure is not the most efficient approach but the approach suggested in this paper can be bolstered by other methods to calculate the cost required to repair the system or structure [16]. Various algorithms [17-19] can be integrated with this method to calculate the area of damage and hence the cost needed to repair and fix them.

3. PREDOMINANCE OF PROPOSED APPROACH

The proposed approach in this study makes use of encoder-decoder architectures, especially U-Net and its variants, which are perfectly suitable for image segmentation tasks, such as crack detection for several reasons.

3.1 Fine localization

The encoder-decoder architecture allows the network to capture high-level contextual information from the encoder and fine details from the decoder. This will be used to make accurate crack localization. Cracks tend to include fine, subtle structures and thus require more precise localization than standard classification tasks. While CNNs have been good in classification tasks, they often fail at pinpointing pixels belonging to a crack. Crack detection using deep learning involves employing Convolutional Neural Networks (CNN) to analyze images for structural defects. This methodology achieves high accuracy by classifying images of cracks and non-cracks, utilizing a dataset of concrete crack images for training [20].

3.2 Variability in shape and size of cracks

Encoder-decoders have no problem with the variability in shape and size of cracks. The downsampling within the encoder captures the overall structure and context, and upsampling within the decoder refines the segmentation map to delineate the boundaries of cracks, regardless of their complexity.

Those direct connections that the U-Net is providing between corresponding layers within the encoder and decoder blocks are very helpful as these could bypass the bottleneck in case downsampling by an encoder while preserving finer information that is critical to perfect segmentation. This often ends up being a bottleneck when using simpler encoder-decoder models without skip connections.

3.3 Outputting segmentation maps

Encoder-decoders are designed to output a pixel-wise prediction to generate a segmentation map highlighting regions of crack in the image. This is a near-ideal fit for a task such as crack detection, although other deep models for classification, like a normal CNN, need extra processing to yield segmenting maps.

Although other deep learning models can be used for segmentation, the natural architecture of an encoder-decoder network makes such a model more suitable for tasks like crack detection, where accurate boundary delineation and contextual understanding are critical.

4. RESEARCH METHODOLOGY

The entire method to perform the research consists of mainly 10 steps from the exploration of the dataset to the end step which is the analysis of the results produced as the final output from the entire research methodology [21, 22].

The very initial phase is an exploration of all the datasets available for the fulfillment of the intended objective. The finalization and incorporation of the datasets then succeed. The integrated dataset then is arranged and loaded into the objects in the next phases in such a manner as to support the training phase of the model in upcoming steps. The Model is then initialized and trained on the loaded dataset in succeeding steps. The trained model is then evaluated on unseen data in the next to last step and in the last step, the results produced by all training and evaluation phases are discussed and analyzed. The step-by-step summary of the methodology is as depicted in Figure 1.

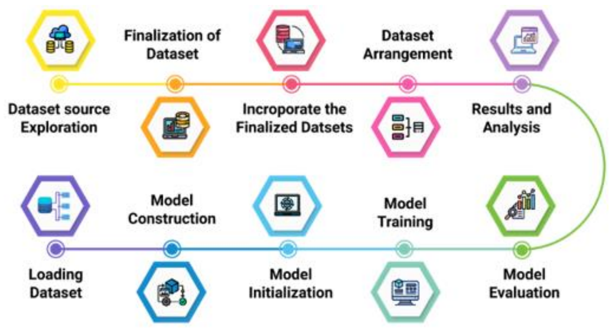

Figure 1. Research methodology

5. EXPERIMENTATION

All the phases of the research methodology are discussed in extended detail in the following sub-section:

5.1 Dataset exploration, diversity and finalization

To fulfill the need for data to train the model the team of authors has performed an intense exploration of the dataset which contains the images of the surfaces with the crack and the corresponding masks for the ground truth. The exploration was performed solely on the internet as the availability of such data cannot be collected locally and it would be feasible to explore various surfaces and get the images.

All the data explored in the previous phase is also checked for diversity and data has been collected from various sources. To create a truly diverse dataset for crack detection, several factors were considered and carefully controlled or varied during data acquisition:

5.1.1 Surface types and materials

Concrete: The authors have included various types of concrete as part of dataset images: smooth, textured, stamped, colored, different aggregate sizes and compositions, and aged concrete exhibiting weathering and discoloration.

Asphalt: Different asphalt types, ages, and conditions (e.g., cracked, potholed, repaired).

Metals: Various metals (steel, aluminum, etc.), finishes (painted, unpainted), and conditions (rust, corrosion).

Other Materials: Wood, brick, stone—any material where cracks might be relevant. This is crucial for genuine generalization.

5.1.2 Crack characteristics

Orientation: Capture images with cracks running horizontally, vertically, diagonally, and at various angles are considered.

Width and Length: The dataset includes cracks of varying sizes, from hairline cracks to wide, extensive cracks.

Depth: While depth is hard to directly capture visually, include cracks in different stages of deterioration to represent different depths.

Types of Cracks: Different crack types like transverse cracks, longitudinal cracks, map cracks, etc were considered in dataset images.

Fillings: We have also included cracks that are filled or repaired, as the model should be robust enough to differentiate these from genuine cracks.

5.1.3 Environmental conditions

Lighting: The dataset images under various lighting conditions: direct sunlight, shade, overcast skies, different times of day, and artificial lighting were considered. Consider adjusting the exposure settings as well to incorporate varying levels of brightness.

Weather: The dataset includes images captured in wet, dry, snowy, or dusty conditions. These conditions significantly change how cracks appear visually.

Shadows: There have been varying angles and intensities of shadow that can obscure cracks, recreating images with different shadow patterns.

5.1.4 Image acquisition

Resolution: Images had high enough resolution to capture fine details of cracks.

Viewpoint: Images were taken from various viewpoints, including close-up shots and more distant images.

Background: Varying background conditions (complex backgrounds, relatively clean backgrounds) to test robustness were considered.

5.1.5 Handling variations and generalization

Data Augmentation: The authors have endeavored to artificially increase dataset size by creating variations of existing images. Standard augmentation techniques rotation, flipping, scaling, cropping, color jittering, and adding noise, have been used. Advanced techniques like GANs that could synthesize new images were also considered for a part of the dataset.

Robust Loss Functions: The authors have employed robust loss functions that are less sensitive to outliers and noisy data. This helps when handling significant variations in lighting or surface quality.

The diversity of the dataset is directly proportional to the model's ability to reliably detect cracks under real-world conditions. So, the diverse dataset was considered and later handled using variation handling techniques and generalization methods.

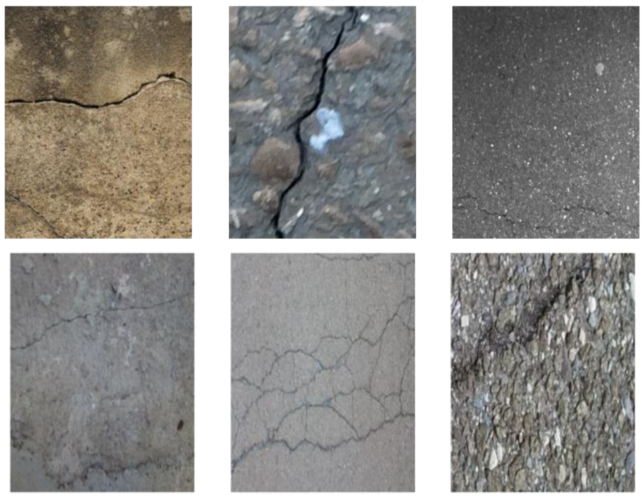

Figure 2. Samples from different datasets

After performing the exploration and collecting the diversity of the dataset, the authors found 4 datasets [23-26]. that have the samples/dataset that is the most suitable for the objective of this article. Figure 2 shows various samples from the diverse datasets.

The Datasets finalized after the intense explorations are then unified under a single entity. Various samples, from various sources, are unified to increase the length of the samples in the dataset. This facilitates the study by increasing the size of the dataset and helps in generalizing the model so the model is less susceptible to noise and hence can be saved from overfitting.

5.2 Arrangement of the incorporated dataset

After unifying the data, the directory arrangement is done in a manner that bolsters the training phase of the model. The input samples received from various sources are then stored in a single directory and renamed and then their mask or ground truth are mapped correctly with their counterpart input images. This would mark the dataset as ready to be loaded for training as each image would be having correct mask mapped to them which would serve as the pair of the input image and ground truth. The arrangements of images and their corresponding masks in the finalized dataset are shown in Figure 3.

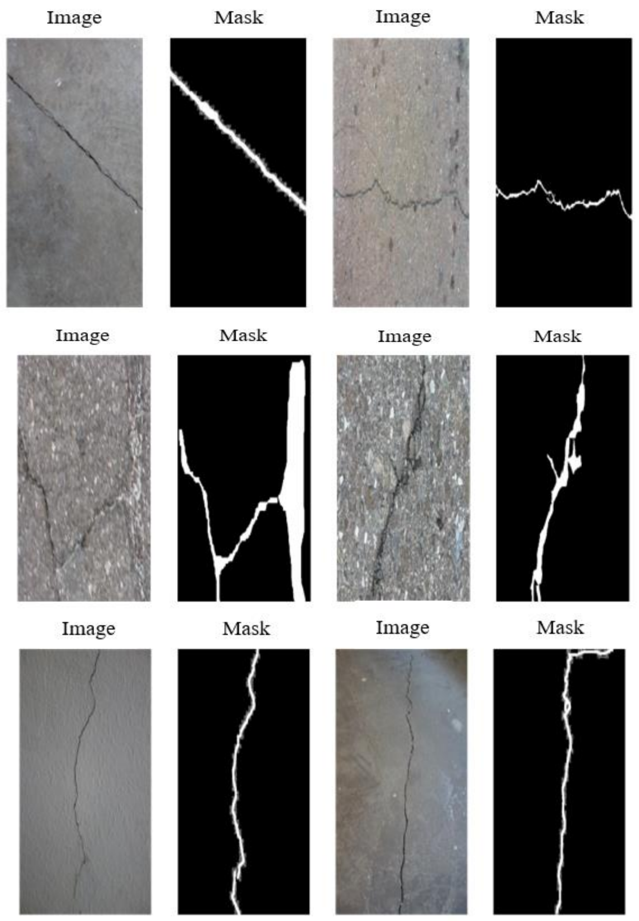

Figure 3. Images and their corresponding masks

5.3 Preprocessing and feature extraction

Marked in the previous phase, the arranged dataset is then loaded into two objects. One object serves the purpose of storing all the input images in a proper format which may be an array or tensor. The other object would store the mask or ground truth value utilized in the model optimization. The loaded datasets are then normalized to bring the values of the data points within the range of 0-1. Each loaded dataset is divided by 255 to achieve the normalized dataset.

The encoder in an encoder-decoder architecture used in this research study performs hierarchical feature extraction using convolutional layers. However, its role is specifically to create a rich, compressed representation of the input image that the decoder can then use to generate the segmentation map. The details of feature extraction are as follows:

Downsampling and Feature Extraction: The encoder used in the proposed study uses a series of convolutional layers followed by pooling (or strided convolutions) to progressively downsample the input image. This downsampling serves two key purposes:

Increases Receptive Field: Going deeper into the encoder, the receptive field of the neurons expands. This means that neurons in later layers "see" a larger portion of the input image, enabling them to capture broader contextual information about the cracks. This is crucial for understanding the overall crack structure and distinguishing it from noise or other image features.

Creates a Compressed Representation: The downsampling effectively compresses the image's spatial dimensions, creating a lower-resolution representation that encodes the essential features of the crack. This compression helps the network learn more abstract and generalizable features, making it less sensitive to minor variations in crack appearance.

Feature Hierarchy Within the Encoder: The hierarchical feature learning has used in the encoder decoder architecture that has been deployed here. The details of the layers used in feature extraction are as follows:

Early Layers: This layer focus on low-level features like edges, gradients, and corners, providing the initial building blocks for crack detection.

Middle Layers: These layers capture more complex patterns, linear structures, and crack morphology. The increased receptive field allows these layers to integrate information from a larger area of the image.

Later Layers (Bottleneck): Extract the most abstract, high-level features representing the overall presence, location, and potentially the type of crack. These features form the compressed representation that is passed to the decoder.

The key difference in the encoder-decoder is how these extracted features are used. Unlike in a standard CNN where the final layer might be a classifier, here, the encoded features serve as input to the decoder.

Decoder Upsampling and Segmentation: The decoder takes the compressed feature representation from the encoder's bottleneck and progressively upsamples it, recovering the spatial information needed for precise segmentation.

Skip Connections: Crucially, encoder-decoders like U-Net utilize skip connections. These connections directly pass feature maps from corresponding encoder layers to the decoder. This allows the decoder to combine the high-level contextual information from the deeper layers with the fine-grained details from the earlier layers, leading to more accurate and sharp segmentation boundaries around the cracks.

In essence, the encoder's role is to learn a rich, multi-scale representation of the crack. The decoder then leverages this representation and the skip connections to reconstruct a detailed segmentation map that precisely outlines the crack's location and extent in the original image.

5.4 Experimentation setup

The experimentation is performed on Google Colab virtual environment with the memory of 13.6 Gigabytes and graphical memory of 12 Gigabytes. The virtual environment also has a storage space of 70 Gigabytes which is utilized to store the temporary data and intermediate results of the experimentations. Colab is based on Debian-based Ubuntu Linux 18.04 LTS and hence has most of the dependencies, for experimentation, like python, pip, and git are pre-installed. All the required libraries as OpenCV, Numpy, etc are also preinstalled on the environment. Few other libraries, like imutils, from third-party sources, are required to be installed according to the need of the experimentation. This virtual environment is accessed throug web-browser client Google Chrome installed on Ubuntu 22.04 LTS with memory of 12 Gigabytes and storage space of 500 Gigabytes. The host operating system runs on Intel i5 processor from 8th generation.

5.5 Model construction

After loading the datasets in the respective objects according to the type of the data, the next step would consist of the model construction. As the model follows the encoder-decoder structure, so the model is constructed accordingly. And in this case of encoder-decoder structure, the model follows the sequential order so the model is constructed as the Sequential() so various layers can be stacked on top of each other. As the model follows the encoder-decoder structure the model is mainly constructed with the input layer, encoder block, decoder block, and the output layer in the end. The input layer accepts the image in the size of 448×448×1.

Encoder: The encoder block follows the input layer and consists of 4 sub-blocks which are made up of repetitive conv2D and Batch Normalization layers, repeated 2 times in the first sub-block of the encoder. The connection between each sub-block of the encoder is established by deploying the Activation layer. The second sub-block has a MaxPooling2D layer for down sampling and then, as in the first sub-block of the encoder, the Conv2D layer is stacked, followed by the Batch Normalization layer. This pattern of Conv2D followed by Batch Normalization is repeated twice in the given order in each remaining sub-block of the encoder. After stacking all the blocks and connecting them by the Activation layer, the encoder is connected by the decoder by connecting both through a block called the bridge. The bridge consists of a MaxPooling layer followed by a repeating layer structure as each sub-block of the encoder, followed by another Activation layer, and at the end, the bridge has the Conv2DTranspose layer.

Decoder: The decoder, as encoder, has a fixed number of blocks i.e. 4. The bridge is connected to the first sub-block of the decoder. The first decoder block is connected to the bridge by concatenating layer. The concatenate layer connects the previous sub-block to the next one in the case of the decoder and the same layer also connects the current sub-block of the decoder to the corresponding sub-block of the encoder. The correspondence of the sub-blocks of encoder and decoder is determined based on the parameters they have or based on the order of the insertion in the network. Each sub-block of the decoder has the repeating structure of layers as in the case of sub-blocks of the encoder with a Conv2DTransopose layer added at the end of each sub-block of the decoder except the last sub-block of the decoder. In the last sub-block, the Conv2DTranspose layer is replaced by the Conv2D layer which produces the output.

The input accepts the input of the size 448×448×1 and each Conv2D layer in each sub-block of the encoder has filters number of filters as 16,32,64,128 in-order, in each sub-block of the encoder. The kernel size for all the Conv2D layers is set as 2×2 and stride of 2 and padding is not changed. MaxPool has a pool size of (2,2). The Activation function is RELU in the entire model. The decoder block has Conv2DTranspose layers which have several filters as 128, and 64,32,16 in order. The kernel size for this layer is kept as 2×2. The stride of 2 is initialized and padding is, as the encoder part, kept the same. The output layer is the Conv2D layer which has kernel size 1×1 and activation function as Sigmoid.

5.6 Callbacks and parameters initialization and model compilation

For training the model efficiently and getting the best results during the training phase 2 callbacks are initialized named EarlyStopping and tensorboard logs. Early Stopping is a callback method that ensures that the model does not learn noise from the supplied data and hence saves the model from overfitting. It does by stopping the model from training when the model does not show any improvement in the accuracy over the epochs. The EarlyStoppping is the method from Keras API. It has a parameter named min_delta which determines the minimum amount of change in the validation accuracy to keep training continues. Patience defines the number of epochs for which the training won't be stopped even if there is no change in the model's validation accuracy over the epochs. Another parameter named restore_best_weights ensures that the model is saved with the best weight even if the training does not show any improvements over the epochs. The initialization of the tensorboard would assist in the visualization of the training in the form of interactive graphs as it would save the training logs which can be used to plot the graphs like accuracy and loss graphs.

Then in this phase, the initialization of the hyperparameters takes place. The first hyperparameter is adam optimizer is used in training. The learning rate is kept as 0.0001. The loss function is assigned as binary_crossentropy and the metric function is the accuracy that indicates the performance of the model would be judged based on the accuracy achieved during the training phase. After the initialization of hyperparameters and callback functions to bolster training, the model is compiled with the compile() method.

5.7 Model training and testing

The model training is started by using the method fit(). This method takes arguments such as training and validation data, batch size, number of epochs, and callbacks. The training and validation dataset is provided as Xtrain, YTrain, and Xtest, Ytest, respectively. Batch size is kept as 4 which suggests that each batch in the dataset would contain 4 image samples.

The significance of the batch size uring training is that it determines the memory usage during the training phase of the model. More value of batch size would use more memory and would be computationally expensive, and the pace of training would be slower in comparison to the case when the batch size is smaller. The model is projected to be trained for 5, 10, 15, and 20 epochs respectively as the experimentation trial maximize the chances to achieve the best accuracy. Verbosity is set to 1 to check the training logs on the screen in real-time. The shuffling is disabled in this case, but it also can be enabled with no noticeable impact on the training or metrics like accuracy and errors. Callbacks are assigned as tensorboard_callback and EarlyStopping.

After assigning all the essential variables, finally, the training is started using the model.fit().

Running trials with the predefined number of epochs, the model achieved the best accuracy on the validation set in the case of 5 epochs. The model suffers from overfitting if trained for more than 10 epochs. The final epoch in each trial reported the accuracy over validation split as 94.5%, 95.0%, 94.5%, and 94.02% respectively with epochs as 5, 10, 15, and 20 epochs. The average time taken by the model to process a single epoch is 183 seconds in each trial. The parameter tuning process was conducted to optimize model performance using a systematic grid search method. Key hyperparameters, including the learning rate (0.0001-0.001), batch size (4-32), and the number of convolutional filters (16, 32, 64, 128), were varied to evaluate their impact on model accuracy and mean IoU score. We evaluated combinations of these parameters on the validation set and observed that a learning rate of 0.0001, batch size of 4, and filter sizes of 16, 32, 64, and 128 achieved the best trade-off between convergence stability and computational efficiency.

The accuracy achieved on the training set and validation set for each epoch is shown in Figure 4. By following the orange curve in the graph, the distinct gradual increment in the accuracy of the model on the training set is visible. The model achieves an accuracy of 95.0%, on the training set till the end epoch which is epoch 10.

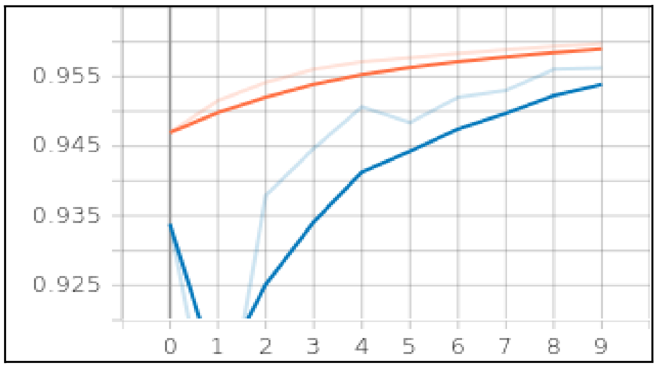

Figure 4. Epoch accuracy

The outcomes from the loss functions are depicted in Figure 5. The small difference between the training and validation curve proves that the model does not learn from the noise, so it is not the case of overfitting. Figure 5 also shows that the losses for the validation split are gradually decreasing which substantiates the optimized learning by the model.

Figure 6 denotes the model's accuracy on the validation split over the progressing numbers of iterations. There is a sharp decrement in the accuracy on iteration 1700 and then there is a constant sharp growth in the same. Till the end iteration, the model achieved an accuracy of 95.0%. Thus, the number of iterations focused during experimentation is 1700.

Justified by Figure 7 which denotes the loss over the validation set. The graph of accuracy and loss are closely related as clear from Figures 6 and 7. As there is a sharp decrement in the accuracy in Figure 6, in Figure 7 there is a sharp increase in the loss function around iteration 1700. After iteration 1700, the model shows improvements sharply which is clear from the accuracy and loss graph of the validation set.

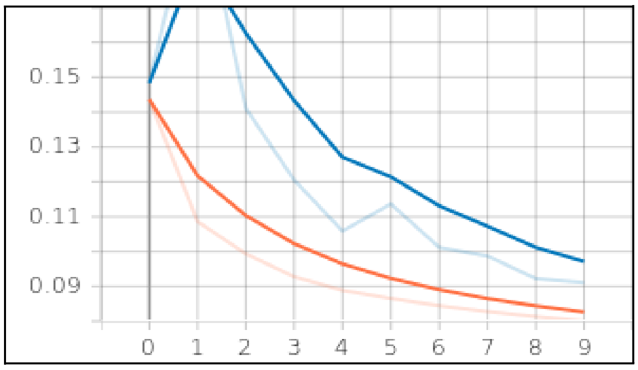

Figure 5. Epoch loss

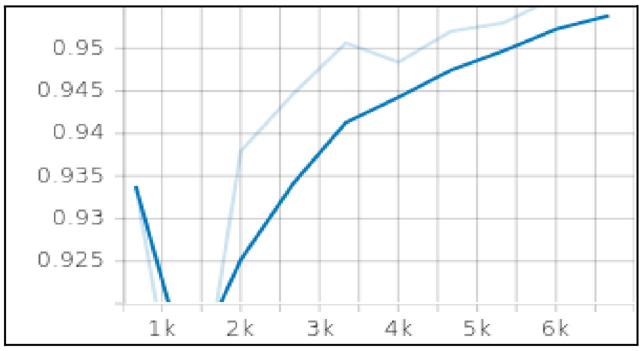

Figure 6. Evaluation accuracy

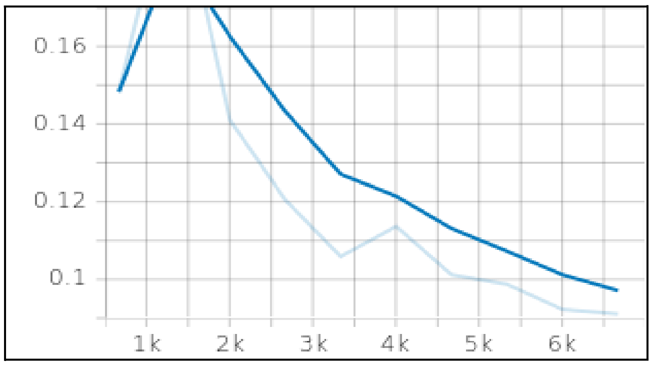

Figure 7. Evaluation loss

After completion of the training phase, the model is tested against the split that was marked as testing split, comprised of 1000 images, during the data preparation phase. The entire testing split is sent to be predicted for the segmented results from the input dataset. The result from this phase of prediction of the testing split is then utilized to determine the prediction threshold which would be used to determine the model's accuracy over the testing split by employing mean IoU or mean of intersection over union score. This performance metric would be the ratio of the intersection of the predicted image and ground truth of the same input image over the union of the predicted image and ground truth. A higher mean IoU score indicates the prediction accuracy of the model.

Prediction on the test split by the trained model shows the mean score of IoU as 83%, shown in Figure 8, which means that the predicted images by the model have been Extending the insights received by mean IoU score on the testing split, the trained model was then exposed to the unseen images. This testing with the unseen would mimic the real-world deployment of the model. The model produces the outputs corresponding to the input images. The few sample predictions made by the model are shown in Figure 9. The model predicts new segmented images which show the structure deformities, i.e. cracks, with a pixel intensity of 255, and the background region is shown by absolute black color or pixel intensity of 0.

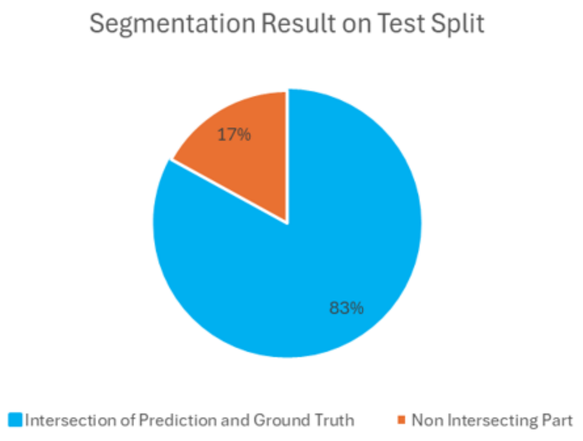

Figure 8. Segmentation score (Mean IoU) on test split

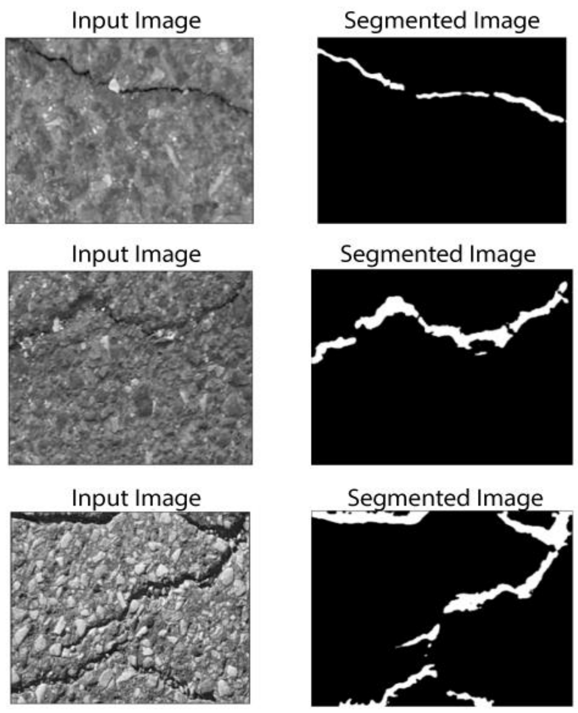

Figure 9. Prediction results by model

6. RESULT ANALYSIS

The achieved mean IoU score of 83% on a dataset size as small as containing only 2500 images, promises enhanced accuracy on the larger dataset. The memory limitation on the environment acts as a constraint to improve accuracy as alteration of the hyperparameters crashes the session and for that reason, the maximum achieved accuracy on various trials limits itself to only 83% of the mean IoU score shown in Figure 8. On the other hand, the visual interpretation, as shown in Figure 9, confirms the superiority of the model's prediction over any static algorithm or method. In the pseudo deployment phase, the model predicts the segmented image with such an accuracy that it can detect the fillings between the trace of cracks on the surface as depicted in the first image sample of Figure 9. From the last sample in Figure 9, it can be inferred by the visual interpretation that the model struggles to predict an ideal segmented image if the input image has a complex crack structure on the surface. This shortcoming can be overcome by training the model over the larger dataset. The basic architecture of the proposed and applied model is depicted in Figure 10 where the convoluted encoder and decoder are connected via a bridge.

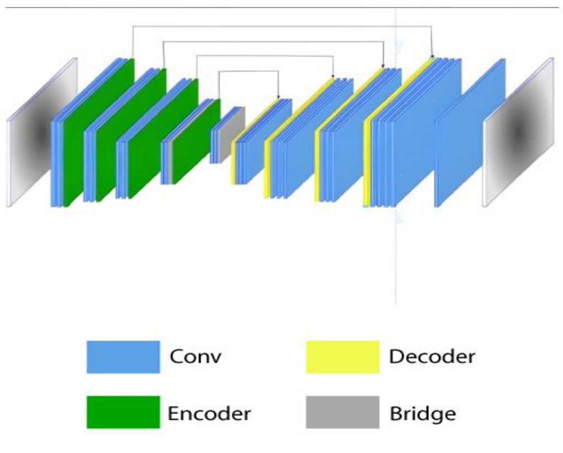

Figure 10. Encoder-decoder structure

7. CONCLUSION AND FUTURE SCOPE

This study targets to automate the process of inspection and detection of the deformities, and cracks in this case, so they can be repaired, and the structure can sustain for a long time. The proposed methodology not only ensures the structural safety of the system but also promises to save the lives of workers who would otherwise require manual inspections. The accuracy achieved on the testing dataset further reinforces the objectives of this study. Currently, the model addresses only one type of deformity, namely cracks. However, augmenting the dataset by incorporating sample images of other deformities, such as rust and peeling, could enhance the model's generalizability as a predictor. Furthermore, by integrating the model with additional algorithms for area calculation and geotagging applications, the entire processes of detection and cost estimation can be automated.

While the model achieved promising results with a mean IoU score of 83%, certain failure cases were observed. For instance, as depicted in Figure 9, the model struggled with detecting complex crack structures on surfaces with high texture or noise. This may be attributed to the limited representation of such scenarios in the training dataset, resulting in insufficient generalization. Similarly, low-contrast cracks in poorly lit environments occasionally led to false negatives. The observed failure cases, particularly with complex or low-contrast cracks, highlight the need for larger and more diverse datasets encompassing challenging real-world conditions. Additionally, exploring hybrid architectures or attention-based mechanisms could enhance the model's robustness to such scenarios. Incorporating multimodal data, such as thermal imaging, may further reduce the likelihood of misclassification.

To address limitations in detecting complex crack patterns, future work could explore the use of 3D imaging modalities such as structured light scanning or photogrammetry to provide a more comprehensive representation of crack morphology and depth. This would necessitate adapting the current 2D encoder-decoder architecture to handle 3D data. Additionally, integrating multimodal data (e.g., infrared thermography for thermal anomalies, acoustic emission sensing for crack propagation detection, or Ground Penetrating Radar for subsurface crack detection) offers significant potential for enhancing the accuracy and reliability of the crack detection system. Advanced data fusion techniques would be crucial to effectively combine these disparate data sources.

REFERENCES

[1] McGlinchy, J., Johnson, B., Muller, B., Joseph, M., Diaz, J. (2019). Application of UNet fully convolutional neural network to impervious surface segmentation in urban environment from high resolution satellite imagery. In IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium, Yokohama, Japan, pp. 3915-3918. https://doi.org/10.1109/IGARSS.2019.8900453

[2] Munawar, H.S., Hammad, A.W., Haddad, A., Soares, C.A.P., Waller, S.T. (2021). Image-based crack detection methods: A review. Infrastructures, 6(8): 115. https://doi.org/10.3390/infrastructures6080115

[3] Mohan, A., Poobal, S. (2018). Crack detection using image processing: A critical review and analysis. Alexandria Engineering Journal, 57(2): 787-798. https://doi.org/10.1016/j.aej.2017.01.020

[4] Kang, D., Benipal, S.S., Gopal, D.L., Cha, Y.J. (2020). Hybrid pixel-level concrete crack segmentation and quantification across complex backgrounds using deep learning. Automation in Construction, 118: 103291. https://doi.org/10.1016/j.autcon.2020.103291

[5] Chen, K., Reichard, G., Xu, X., Akanmu, A. (2021). Automated crack segmentation in close-range building façade inspection images using deep learning techniques. Journal of Building Engineering, 43: 102913. https://doi.org/10.1016/j.jobe.2021.102913

[6] Pan, Y., Khodaei, Z.S., Aliabadi, F. (2025). Online fatigue crack detection and growth modelling through higher harmonic analysis: A baseline-free approach. Mechanical Systems and Signal Processing, 224: 112167. https://doi.org/10.1016/j.ymssp.2024.112167

[7] Hsieh, Y.A., Tsai, Y.J. (2020). Machine learning for crack detection: Review and model performance comparison. Journal of Computing in Civil Engineering, 34(5): 04020038. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000918

[8] Zhang, X., Tang, H., Yu, C., Zhai, D., Li, Y. (2025). SS-CCDN: A semi-supervised pixel-wise concrete crack detection network using multi-task learning and memory information. Measurement, 239: 115478. https://doi.org/10.1016/j.measurement.2024.115478

[9] Hu, H., Li, Z., He, Z., Wang, L., Cao, S., Du, W. (2024). Road surface crack detection method based on improved YOLOv5 and vehicle-mounted images. Measurement, 229: 114443. https://doi.org/10.1016/j.measurement.2024.114443

[10] Kirthiga, R., Elavenil, S. (2024). A survey on crack detection in concrete surface using image processing and machine learning. Journal of Building Pathology and Rehabilitation, 9(1): 15. https://doi.org/10.1007/s41024-023-00371-6

[11] Zadeh, S.S., Khorshidi, M., Kooban, F. (2024). Concrete surface crack detection with convolutional-based deep learning models. arXiv preprint arXiv:2401.07124. https://doi.org/10.48550/arXiv.2401.07124

[12] Hu, F., Gou, H.Y., Yang, H.Z., Yan, H., Ni, Y.Q., Wang, Y.W. (2025). Automatic PAUT crack detection and depth identification framework based on inspection robot and deep learning method. Journal of Infrastructure Intelligence and Resilience, 4(1): 100113. https://doi.org/10.1016/j.iintel.2024.100113

[13] Kaveh, H., Alhajj, R. (2024). Recent advances in crack detection technologies for structures: A survey of 2022-2023 literature. Frontiers in Built Environment, 10: 1321634. https://doi.org/10.3389/fbuil.2024.1321634

[14] Sakshi, Kukreja, V. (2024). Machine learning and non-machine learning methods in mathematical recognition systems: Two decades' systematic literature review. Multimedia Tools and Applications, 83(9): 27831-27900. https://doi.org/10.1007/s11042-023-16356-z

[15] Sakshi, Kukreja, V. (2023). Image segmentation techniques: Statistical, comprehensive, semi-automated analysis and an application perspective analysis of mathematical expressions. Archives of Computational Methods in Engineering, 30(1): 457-495. https://doi.org/10.1007/s11831-022-09805-9

[16] Helvig, K., Trouvé-Peloux, P., Gavérina, L., Roche, J.M., Abeloos, B. (2024). Database for transfer learning in crack detection and localization on metallic materials using flying spot thermography and deep learning. Journal of Electronic Imaging, 33(3): 031202-031202. https://doi.org/10.1117/1.JEI.33.3.031202

[17] Huang, L., Fan, G., Li, J., Hao, H. (2024). Deep learning for automated multiclass surface damage detection in bridge inspections. Automation in Construction, 166: 105601. https://doi.org/10.1016/j.autcon.2024.105601

[18] Shen, J., Ma, T., Song, D., Xu, F. (2025). An embedded physical information network for blade crack detection considering dynamic multi-level credibility. Mechanical Systems and Signal Processing, 224: 111948. https://doi.org/10.1016/j.ymssp.2024.111948

[19] Amini, L., Karimi, H. (2024). CNN-based labelled crack detection for image annotation. Industrial and Manufacturing Engineering, 17(2): 1-9. https://doi.org/10.20944/preprints202405.1702.v1

[20] Pal, S., Das, S., Manna, S. (2024). Deep learning-based detection of defects from images. In Green Industrial Applications of Artificial Intelligence and Internet of Things, Netherlands, pp. 176-182. https://doi.org/10.2174/9789815223255124010016

[21] Nuanmeesri, S. (2024). Utilization of multi-channel hybrid deep neural networks for avocado ripeness classification. Engineering, Technology & Applied Science Research, 14(4): 14862-14867. https://doi.org/10.48084/etasr.7651

[22] Fan, C.L. (2025). Evaluation model for crack detection with deep learning: Improved confusion matrix based on linear features. Journal of Construction Engineering and Management, 151(3): 04024210.

https://doi.org/10.1016/j.ymssp.2024.111948
[23] Arun, R.K. (2024). Concrete surface sample images for surface crack detection. Kaggle. https://www.kaggle.com/datasets/arunrk7/surface-crack-detection.
[24] Atkin, G. (2024). Concrete crack image detection. Kaggle. https://www.kaggle.com/code/nadagamal3/concrete-crack-image-detection/input.
[25] Atkin, G. (2024). Surface-defect-detection-dataset PCB inspection, solar panels, fabric defect, magnetic tile. Kylberg Texture. Kaggle. https://www.kaggle.com/datasets/yidazhang07/bridge-cracks-image.
[26] Atkin, G. (2024). Crack segmentation dataset. Kaggle. https://www.kaggle.com/datasets/lakshaymiddha/crack-segmentation-dataset.