Optimized Deep Learning Model for Concrete Crack Classification Monitoring on Resource Constrained Devices

Vanusha. D [1], Karthikeyan. M [1], Naga Malleswari. TYJ [2,*], and Ushasukhanya. S [2]

[1] Department of Computing Technologies, School of Computing,
SRM Institute of Science and Technology, Kattankulathur, Chennai, India
[2] Department of Networking and Communications, School of Computing,
SRM Institute of Science and Technology, Kattankulathur, Chennai, India
Email: vanushad@srmist.edu.in (V.D.); karthikm1@srmist.edu.in (K.M.); nagamalt@srmist.edu.in (N.M.T.Y.J.); ushasuks@srmist.edu.in (U.S.)
*Corresponding author

***Abstract*—Structural cracks are commonly caused by factors such as high loading, fatigue, degradation, thermal expansion, and humidity. Detection of cracks in early stages is important. If neglected, these cracks can spread and weaken the building as well as other civil infrastructure. To enhance safety and prevent catastrophic structural failures, accurately detecting cracks is essential in Structural Health Monitoring (SHM) systems, thus enabling early maintenance. The latest developments in Deep Learning (DL) have facilitated the automatic detection of structural cracks through images taken with the help of cameras, drones, or mobile devices. Such methods minimize the manual inspection and enhance monitoring efficiency. In this paper, a systematic optimization of deep learning models through hyperparameter tuning with random search was implemented to achieve a balance between classification performance and computational efficiency for resource-constrained environments. Custom architecture, (Convolutional Neural Network (CNN) and CNN-Long Short-Term Memory (LSTM)), as well as transfer learning models (MobileNetV2, ResNet101, and DenseNet201), are evaluated on the Middle East Technical University (METU) concrete crack dataset (40,000 labelled images). The performance measures, such as accuracy, precision, recall, F1-Score, inference time, and computational complexity in Million Floating-Point Operations (MFLOPs), are evaluated and compared on the models. From the results, MobileNetV2 shows a classification accuracy of 99.8% with a reduction of 50% computational complexity, thus making it suitable for deployment on constrained devices in real-world applications.**

***Keywords*—building crack detection, deep learning optimization, hyperparameter tuning, MobileNetV2, resource-constrained devices**

I. Introduction

Cracks in buildings can result in economic loss and a negative impact on the lives of the occupants. The extent of the loss depends on the crack type, the speed of detection of the cracking, and how fast the corrective actions are taken to control the cracking. Therefore, early signs of structural degradation and building cracks should be detected and classified to prevent potential failures [1]. The manual approaches for identifying the cracks are tedious, time-consuming, require inspector expertise, and are error prone. As buildings age and infrastructure become more complex, effective and scalable structural health monitoring becomes increasingly important. As a result of natural disasters such as earthquakes, manual inspection becomes even more difficult when numerous structures must be assessed within a short period of time. To overcome these limitations, vision-based civil infrastructure evaluation and monitoring methods have emerged as effective alternatives [2–4]. Vision-based crack detection is a non-destructive evaluation technique that utilizes computer vision and image analysis methods to identify cracks on structural surfaces. Historical or other monuments where physical inspection methods cannot be applied can be protected by using these techniques. Artificial Intelligence (AI), especially Deep Learning (DL), has developed to a point that the crack detection systems detect cracks automatically with high accuracy and efficiency. DL models automatically learn hierarchical image features from large volumes of structural images, resulting in improved detection accuracy and generalization capabilities [5]. With the help of Convolutional Neural Networks (CNNs), image classification tasks are used to a significant extent for identifying cracks in civil infrastructure and showing strong performance due to the capability of extracting spatial features by fully connected layers or other classifiers. Due to the increase in urbanization, infrastructure networks expand, and scalable automated monitoring systems are required to maintain structural safety and reliability [6–8].

Manuscript received January 4, 2026; revised February 26, 2026; accepted April 16, 2026; published June 26, 2026.

Yamaguchi and Mizutani [9] shows that the DL models are efficient in automatically detecting cracks in civil infrastructure. The discriminative features are directly learned from images of structures by the CNN model, thus improving the accuracy of crack detection. Deep neural networks, supported by complex datasets and transfer learning, enhance generalization under varying structural conditions [9]. Additionally, advanced frameworks improve robustness against environmental variations and noise [3]. To resolve this challenge, the focused research is on designing lightweight deep learning models, which are neural network architectures to achieve reduced computational complexity, model size, and memory consumption while maintaining acceptable prediction accuracy. To get the reduced number of Floating-Point Operations (FLOPs) and model parameters, lightweight models use architectural strategies such as depth-wise separable convolutions, parameter sharing, and reduced network depth. For resource-constrained edge devices, the MobileNet model and similar models perform well due to their efficient network design. The models such as YOLOv4, YOLOv8, and YOLOv11 are good in real-time detection, they are primarily designed for object localization and bounding box prediction in complex scenes. Whereas, the objective of the study in this paper is the binary classification of building cracks. The model determines whether an input image contains a crack or not. Therefore, classification-based deep learning architectures are considered more suitable for this problem. The above-mentioned object detection models typically require more computational resources due to the simultaneous tasks of localization and classification. Hence, this study focuses on optimizing classification architectures with reduced computational complexity for deployment in resource-constrained structural monitoring devices.

To achieve this objective, hyperparameter optimization plays a critical role in improving model's performance while maintaining computational efficiency. In this paper, we implemented grid search, Bayesian optimization, evolutionary algorithms (e.g., genetic algorithms), and random search hyperparameter-tuning approaches. Random search, with fewer iterations, high-dimensional hyperparameter spaces, and near-optimal configurations, is explored and identified efficiently when analyzed with exhaustive search strategies. To balance classification accuracy and computational efficiency, in this paper, random search is adopted for systematic hyperparameter optimization. These hyperparameters are adjusted in the implementation phase to reduce the overall computation and to achieve high classification precision. Random search is more efficient to explore high-dimensional hyperparameters and to identify optimal configurations in fewer iterations when compared to other techniques such as Bayesian optimization and evolutionary algorithms like genetic algorithms. The proposed work optimizes deep learning models using the benchmark Middle East Technical University (METU) concrete crack dataset, which contains images collected from different campus buildings with positive and negative crack samples.

The objectives of the proposed work are as follows:

- Optimizing hyperparameters to improve the performance of custom-built deep learning architectures, including CNN and CNN-LSTM hybrids.
- Fine-tuning pre-trained transfer learning architectures such as ResNet101 and DenseNet201 to improve performance.
- Improving the performance of a lightweight network, such as MobileNetV2, to ensure efficient crack classification.
- Comparing the performance of optimized architectures in terms of efficiency and computational load.

This paper mainly presents binary classification of cracks as preliminary work for deploying the model on edge devices and to ensure computational efficiency and suitability. Severity stage detection needs a large, annotated dataset, complex models, and a higher computational cost. These are planned as future work.

II. Literature Review

This section gives a detailed review of related studies on concrete crack classification using deep learning. Yamaguchi and Mizutani [9] developed an automated crack detector using deep learning to resolve the critical problem of aging structures and reduce maintenance costs. A dataset was created with various images of cracks. The data set was increased ten times by horizontal inversion and cutting operations. The dataset was finally prepared with images of 64×64 size. With the available RGB images of cracked concrete and others, the Convolution Neural Network model gave an accuracy of 79.9% with Rectified Linear Unit (ReLU) as an activation function. 73.3% accuracy was obtained with cracked concrete, chalk letters, and other images after training 30 times on AWS GPU instances, with a processing time of 4 h. The limitation in this work is that it achieved an average accuracy of 79.9% and when trained on AWS Graphics Processing Unit (GPU), the accuracy dropped to 73.3%. Though the drop is accuracy can be considered, the work did not optimize computation efficiency and accuracy. Also, different models were not compared. In Ref. [10], a total of 800 images of cracked and non-cracked pavement images were collected using drones. In the preprocessing step, a normalization function is applied to average the intensity, followed by converting the images to grayscale. In addition, edge detection was done by a Sobel filter with noise removal. Feature extraction was done using a bilateral filter, Canny algorithm, Fuzzy k-means clustering, and morphological filter with dilation operation using an erosion filter, which gave the best results. A feedforward neural network with 8 neurons in the hidden layer was used to process the extracted features and achieved an accuracy of 82.5%. The drawback in this work is that the small size of dataset affected accuracy. However, the work did not focus on reducing computation efficiency and the work failed to perform comparative analysis. Maintenance operations are crucial for concrete structures [11]. A deep CNN model was used to detect the cracks on the concrete structures. A dataset with 1250 authentic images of crack and no crack were used to train

and validate the model. Images with 224×224×3-pixel resolution were given as input to the CNN model and trained on a GPU workstation. 60,000 images were built with preprocessing operations. Parameter weights and Bias were preset using the methods "xavier" and "constant", respectively. The model was trained for 15000 iterations with a momentum of 0.9 and a learning rate of 0.01 values. The model achieved the highest accuracy of 99.39% at the 50th training iteration. The work achieved high accuracy with an augmented dataset as the real dataset had 1250 images. The model used in this work was custom-made CNN and no comparison with other deep learning models. The paper not focused on optimizing computational efficiency for deployment in resource constrained device for automation of crack classification in real-time.

In Ref. [12], road and bridge crack detection using a deep learning framework was discussed. Three datasets with 2068 bridge crack images, 40,000 cracked and uncracked concrete images and 400 images of Asphalt cracks and non-cracks images were considered to train the modified LeNet-5 model. The images were resized to 1200×1200 pixels. Principal Component Analysis (PCA) was used for dimensionality reduction to train fast. Stochastic Batch Gradient Descent has been used for training all the datasets with a batch size of 32 and Min delta value 0.01, Patience value 4, No. of epochs 10. For Asphalt data set, No. of epochs used is 30. Finally, the model with and without PCA gave 68% and 82% of accuracy respectively for the Asphalt dataset. For the bridge dataset, the models with and without PCA gave an accuracy of 96% and 98%, respectively. The work however has employed the transfer learning model Lenet with modification which achieved higher accuracy for the bridge dataset and average accuracy with asphalt dataset. The limitation in this work is that the dataset size for various categories is less, and computational optimization was not discussed. Liu *et al.* [13] used a deep hierarchical Convolutional Neural Network (CNN). DeepCrack method included extended Fully Convolutional Networks (FCN) and Deeply Supervised Nets (DSN) which gave better results with an F1-Score of 86.5%. 537 images of manually annotated building crack and non-crack images were used to train the DeepCrack. 13 convolution layers were used, and the fifth pooling layer was discarded for refined results. Predictions were refined using guided filtering, and the computational efficiency was good. During training, the images were resized to 256×256 with batch size of 1, learning rate of 0.0001, and momentum 0.9 to get better results. The study didn't present the comparative analysis of performance with the other deep learning models and did not focus on optimization of computational efficiency. In Ref. [14], a multi-class classification technique using the pre-trained CNN models was proposed. They employed a transfer learning approach for feature extraction. In addition, the authors used K-Nearest Neighbor (KNN) to classify the features. ResNet 50 with KNN integration achieved an exceptional outcome compared to VGG16-KNN combination. They employed AlexNet, VGG16, VGG19, GoogleNet, and ResNet variants to detect the cracks using images. The experimental outcomes suggested that VGG16 and GoogleNet were superior in detecting cracks with limited computational resources. This work lacks the actual image dataset from a genuine source for concrete crack classification. The work involved scraping the concrete crack image, resulting in a total of 2516, which is a serious concern about the quality and reliability of images. In Ref. [15], a lightweight CNN-based vision model for concrete crack detection was built on low-cost embedded controller. The model achieved an accuracy of 90.03% for pavement, 83.24% for walls, and 84.62% for bridge decks with limited memory and parameters. The limited, accessible, diversified, and large-scale datasets on building crack categorization is a significant knowledge gap. The current models frequently employ limited datasets that fail to incorporate many potential crack sizes, forms, and environmental factors. The existing models may produce excellent results in a controlled environment. Many researchers work on AI-based automated inspection systems [16–19], However, this work focuses on systematic hyperparameter optimization and reducing computational operation while improving accuracy, especially for classification-based crack detection models, and a comparative analysis to identify the model well-suited for AI-based IoT based crack monitoring systems.

However, the work doesn't focus on real-time settings. The current challenge is the generalization of crack detection models, which can be applied to any type of building and environment. Various models are trained on various categories of images, but the model trained on a concrete dataset may not give good results when applied to wood or steel images. There is a lack of information and understanding of the required features for crack categorization precisely. CNNs and vision transformers can extract the essential features, but they didn't provide features that influence classification. Recent models require more computing resources for training and inference, which causes problems in real-time deployment. Recent studies focus on pruning, quantization, and Neural Architecture Search (NAS) to deploy models in edge devices. Models on edge devices or real-time monitoring systems should be optimized for accuracy and computing efficiency. Conversely, this paper addresses these gaps by implementing systematic hyperparameter optimization on light weight architectures for real-time deployments on edge devices.

III. Materials and Methods

Most of the research work was on customized CNN models or transfer learning architectures that are mainly focusing on improving accuracy. Small data set size was the major limitation in many of the papers. The trade-off analysis between model performance and computational complexity was not focused in most of the papers, which is a very critical concern for real-time deployment of light weight models in edge devices. This is very much needed for automated AI-driven crack classification in real-time environment. This study mainly focused on optimization

of customized deep learning models and transfer learning on pre-trained architectures with hyperparameter tuning to identify the best configurations that balance both model performance and computational efficiency. This enables the deployment of optimized models with reduced computational cost and high accuracy for IoT-based automated crack classification. Most importantly, the data set selected for our work is well suited for training, validation and testing of the models. Both custom models (CNN, CNN-LSTM) and pre-trained models (MobileNetV2, DenseNet201, ResNet101) were optimized using Random Search with 50 iterations to determine the optimal hyperparameters. Random search was chosen for optimizing hyperparameters due to its efficiency in searching for high dimensional search spaces and to identify the best configuration parameters in a smaller number of iterations and less computational cost when compared to optimization algorithms like Bayesian and genetic algorithms. To balance computational efficiency for edge devices, 50 was set as random search iterations. Important hyperparameters, such as learning rate, dropout, filters, and dense units, were collectively optimized. The convergence of the performance measures was also early, and no significant growth was observed beyond mid-iterations. Therefore, 50 iterations were not only beneficial to experiment with, but were also intuitive to calculate, and the process was, therefore, suitable when there was need to apply IoT in real-time A consistent search space (learning rate: 0.002–0.008, dropout: 0.2–0.4) was used for all the models to ensure a fair and unbiased comparison across all models. to give a good comparison of all the models to ensure. The hyperparameters such as learning rate, dropout, optimizer (Adam) and dense layers were optimized using random search. Fig. 1 illustrates random search hyperparameter tuning, in which learning rate and dropout rate are adjusted through sampling random points in the search space. At the same time, architecture-specific parameters (e.g., filters in CNN/CNN-LSTM and dense units in transfer learning models) were also varied. In this study, the computational efficiency of the models was determined using Floating Point Operations (FLOPs). The FLOPs are calculated by counting operations per filter over feature maps in the case of convolutional layers and by counting input-output neuron connections in the case of fully connected layers. To ensure fair comparison, the same methodology was used in all models (CNN, CNN-LSTM, MobileNetV2, ResNet101, DenseNet201). MFLOPs were measured for evaluation across the models. Each of the models was trained and tested in the same experimental conditions. Experiments were performed in Kaggle with NVIDIA T4x2 GPUs (64 GB memory each). All models used the same dataset, preprocessing procedures, and settings of the batch size. This ensures that differences in training, testing and inference times are because of variations in model architecture and the level of computation rather than hardware or configuration differences. Therefore, comparisons of training, testing, inference times and computational cost are consistent and reliable across the models.

Hyperparameter tuning was done using random search for CNN and CNN-LSTM [20] optimizing parameters such as learning rate, dropout, filters/kernels. To balance the model capacity and computational efficiency, a range of filters from 32 to 256 was used. Along with varied dropout rates, the model's sensitivity to hyperparameters can be controlled, improving the generalization and robustness of the models. In addition, for validating model performance, accuracy, precision, recall, and F1-Score are computed for training, validation, and testing. MobileNetV2, ResNet101, and DenseNet201 models were pretrained using ImageNet weights, and the convolutional base was frozen, and the fine-tuning of the classifier head was done.

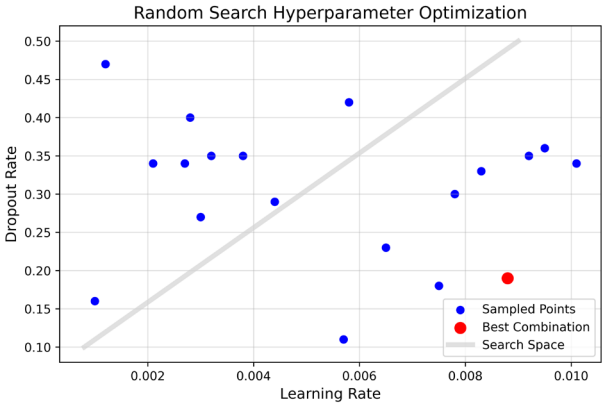

Fig. 1. Random search hyperparameter optimization.

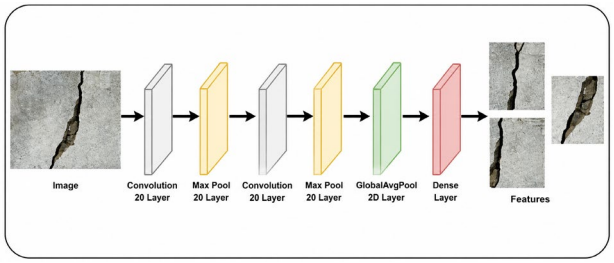

Fig. 2. Architecture of CNN model.

A. CNN Model

Convolutional Neural Networks (CNNs) [21] as shown in Fig. 2 are used for image-based tasks. They are made up of several layers, each of which is responsible for extracting sophisticated information from the input data. Convolutional layers which are tiny filters are moved across the input to capture local patterns. The network may automatically learn hierarchical features starting from edges, textures and progressing to more abstract forms and concepts. Down sampling the collected features using the pooling layer lowers the computing burden and improves translational invariance. Convolutional and pooling layers are frequently stacked with fully connected layers for classification in CNN. Their ability to automatically learn relevant features from raw data and their translation invariance property make CNNs highly effective for image recognition, object detection, and various other tasks in computer vision. Thus, the authors have optimized the CNN model using Random search methodology and accordingly best hyperparameter settings are tabulated

which are used for model training and validation. Table I outlines the optimized hyperparameter setting of the CNN models.

TABLE I. CNN MODEL OPTIMIZED HYPER PARAMETERS

Hypermeter Setting	Dropout Rate	Learning Rate	Filter 1	Filter 2	Optimizer
1	0.30	0.008	64	32	ADAM

B. CNN-LSTM

Convolutional Neural Networks-Long Short-Term Memory (CNN-LSTM) [22] is another hybrid architecture which combines the strengths of CNNs with LSTMs to process sequential data that contains spatial hierarchies. Convolutional Neural Network (CNN) layers are used to extract features of the input data and LSTMs to process the sequences over time in CNN-LSTM architecture. This paradigm is useful when use of data like videos or time series are needed. The CNN component captures each frame or time step's spatial information, which then extracts regional patterns. The LSTM is a recurrent neural network created to capture temporal dependencies and receive output from CNN. The LSTM analyzes the CNN output to model the context and distant dependencies, allowing the model to understand intricate temporal correlations in the data. CNN-LSTM architecture finds applications in action recognition, video analysis, and natural language processing, where spatial and temporal features are crucial for accurately understanding and predicting sequential information. Fig. 3 highlights the architecture of the CNN-LSTM model.

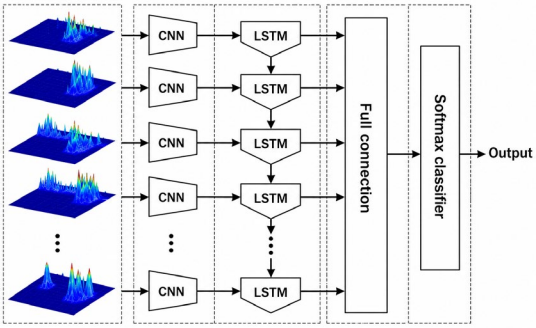

Fig. 3. Architecture of CNN-LSTM model.

In this study, the authors fine-tuned the CNN-LSTM model using Random search hyperparameter tuning and accordingly best hyperparameter settings are tabulated which are used for model training and validation.

Despite sequential data of LSTM network, CNN-LSTM is considered to learn dependencies in extracted feature representations. Nevertheless, findings have shown that CNN is more effective when used in the classification of a static image. Table II presents the key parameters of the CNN-LSTM model.

TABLE II. CNN-LSTM PARAMETERS

Hypermeter Setting	Dropout Rate	Learning Rate	Filter 1	Filter 2	No. of layers	Optimizer
1	0.2	0.00033	48	32	8	ADAM

C. MOBILENETV2

MobileNetV2 [23] has 3.4 million parameters and 0.32 GFLOPs to fit in mobile gadgets, focusing on efficiency and not on precision. It expands the original MobileNet model, with such features as inverted residual structure, depth-wise convolutions, and width and depth multipliers. These trade-offs enable MobileNetV2 to balance the performance and resource requirements; thus, it is appropriate to be used in real-world scenarios in devices with low computational power. Fig. 4 shows the architecture of the MobileNet model. The models of MobileNetV2, where pretrained models are usually trained on large datasets such as ImageNet, are a helpful starting point in a variety of computer vision tasks. Using such models on datasets pertinent to any given application (object detection or semantic segmentation), developers can use learned features in MobileNetV2 and customize them to their application. This method of transfer learning helps to solve a significant amount of training from scratch, which makes MobileNetV2 an effective alternative in real-life situations. Thus, MobileNetV2 has been optimized with the help of random search hyperparameter tuning, and accordingly, the optimal hyperparameter settings have been tabulated according to which the model is trained and validated. The crucial parameters of the MobileNet model are outlined in Table III.

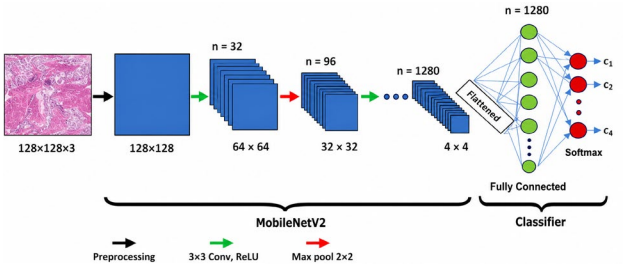

Fig. 4. MobileNet architecture.

TABLE III. MOBILENET PARAMETERS

Hyperparameter Setting	Dropout Rate	Learning Rate	Dense_Units_1	Dense_Units_2	No of Layers	Optimizer
1	0.3	0.002	64	224	6	ADAM

D. ResNet101

The deep convolutional neural network design known as ResNet101, or Residual Network-101 [24], is renowned for its creative application of residual blocks to solve the vanishing gradient problem in intense networks. ResNet-101, a 101-layer neural network consists of 42.513 million parameters and requires about 7.6×10^9 floating

point operations (7.6 GFLOPS). ResNet introduced by Microsoft Research, excels at picture categorization and other computer vision applications. The residual block, which includes skip connections that skip one or more layers, is the main component of ResNet-101. The network can learn residual mappings with these connections, effectively capturing only the modifications required to improve the input attributes. As a result, the degradation issue in deeper networks is lessened, and incredibly deep architectures can be trained. Fig. 5 shows the ResNet-101 architecture.

So, in this work, the ResNet101 model was optimized using Random search hyperparameter tuning and accordingly, the best hyperparameter settings are tabulated in Table IV which are used for model training and validation.

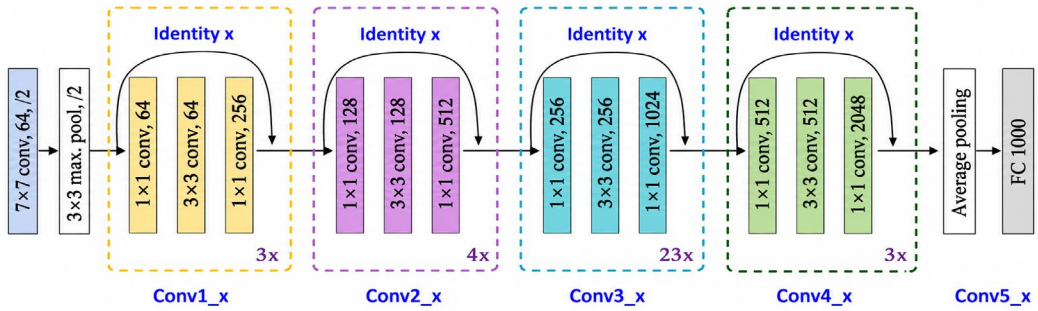

Fig. 5. ResNet architecture.

TABLE IV. RESNET PARAMETERS

Hyperparameter Setting	Dropout Rate	Learning Rate	Dense_Units_1	Dense_Units_2	No of Layers	Optimizer
1	0.30	0.00896	32	224	7	ADAM

E. *DenseNet201*

One of the convolutional neural network designs that is widely recognized to be used to extract image features is the DenseNet201 [25] design that is known to have a dense connection and be better designed to extract image features. The dense connections of 201 layers with 20.24M parameters and 8.4 GFLOPs are expected to address the issue of vanishing gradients and encourage feature reuse. These connections allow the connection of all the previous layers directly to every layer, which improves gradient flow and propagation of features during training, and the number of parameters is reduced as one-layer shares features with another thus achieving computational efficiency. DenseNet201 is an excellent trade-off between cost and complexity of modeling through bottleneck layers and transition layers. DenseNet201 outperforms among the other models well on numerous image identification tasks, delivering state-of-the-art results with a smaller number of parameters. Fig. 6 shows DenseNet architecture. DenseNet 201 model was optimized using Random search hyper parameter tuning in this work. Accordingly, best hyper parameter settings are tabulated in Table V which are used for model training and validation.

To train deep models on low-resource devices, Adam was selected over Stochastic Gradient Descent (SGD) and Root Mean Square Propagation (RMSProp) due to its adaptive learning rates, usage of momentum, and quick converging ability. It requires minimal tuning by hand, treats gradients, whether noise or sparse, correctly, and achieves training stability at minimal cost of computation overhead.

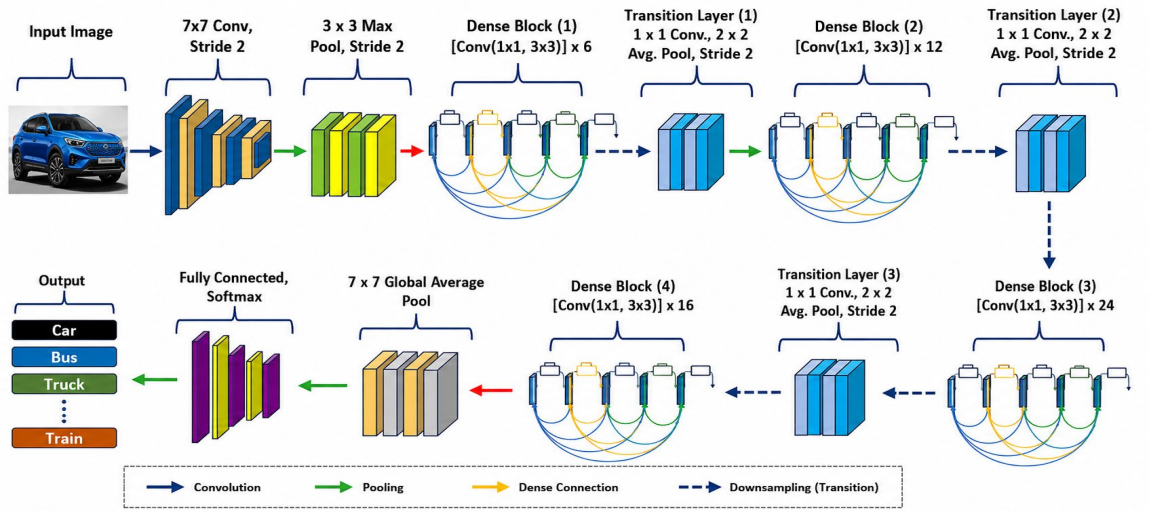

Fig. 6. DenseNet architecture.

TABLE V. DENSENET PARAMETERS

Hypermeter Setting	Dropout Rate	Learning Rate	Dense Unit 1	Dense_Unit_2	No. of Layers	Optimizer
1	0.3	0.0017	224	224	6	ADAM
2	0.4	0.00018	192	160	7	ADAM
3	0.2	0.0021	96	224	7	ADAM

F. Dataset

For the implementation of optimized custom CNN, CNN-LSTM and pre-trained models like MobileNetV2, ResNet101, and DenseNet201 models for crack classification, the publicly available dataset [26] was used. The models were tuned using Random Search for hyperparameter optimization with 50 iterations. Model implementation was performed on Kaggle using NVIDIA T4x2 GPUs, each with 2580 CUDA cores and 16 GB GPU memory. The dataset was collected from various buildings on the METU campus and is divided into two categories: negative (no cracks) and positive (cracks present) images, each containing 20,000 images, for a total of 40,000 images. The images have a resolution of 227×227 pixels with RGB channels. These images were generated from 458 high-resolution originals (4032×3024 pixels) and exhibit variability in surface finish, crack type, crack size, and environmental conditions such as illumination and shadows. No data augmentation (e.g., rotation, flipping) was applied, and all images were resized to 120×120 pixels for faster training. The METU dataset consists of various crack types (e.g., fine, wide, longitudinal, transverse), their sizes, and the environmental conditions during capture, providing better context for model evaluation and generalization. The METU dataset, which is employed in the present study, is balanced between the classes of crack and non-crack, and the issue of the imbalance in classes does not affect the performance of the model as much. Nonetheless, in real life situations, there might be unequal distributions of data. Therefore, the future work be evaluated in imbalanced conditions using techniques such as class weighting, resampling, or focal loss to improve robustness and generalization. To ensure reproducibility and for fair comparison across models, the data set was split as 70% training data, 15% validation data and 15% testing data.

G. Proposed Methodology

1) CNN

Convolutional Neural Network (CNN) model was customized for this task to extract the features and dimensionality reduction by various convolutional layers, followed by max-pooling layers. These layers are designed to capture spatial feature hierarchies automatically. The spatial data was then compressed using a global average pooling layer applied to the retrieved features. The last step was to add a fully connected dense layer with a sigmoid activation function to create the final classification result. Further parameter tuning has been done using random search for different layers where different numbers of kernels and dropout layers are used. To fix the step size of parameters, learning is essential in neural networks training. The techniques such as learning rate schedules or adaptive algorithms such as Adam or RMSProp aid are used to adjust the rates dynamically. Conversely, dropout rates are used for regularization by randomly nullifying input units that demand careful calibration. Higher rates augment regularization but can induce underfitting, while excessively low ones may fail to curb overfitting. The selection and configuration of filters that define the convolutional feature extractors require a wise trade-off between model capacity and computational efficiency. And the placement of dropout layers, commonly after convolutional and fully connected layers, wisely enhances generalization, but it must be balanced against the risk of hindering learning, significantly in shallower networks or when data availability is limited.

2) CNN-LSTM

CNN-LSTM architecture is appropriate for sequence classification tasks. It combines LSTMs to record temporal relationships and CNNs to extract spatial characteristics from individual frames. Like CNN, model hyperparameter tuning using random search is employed, such as dropout, learning rates, and using varied filters to train the CNN-LSTM model. These were proven better than CNN, which is tabulated in the results and analysis section.

3) Transfer learning models

The foundation model for the transfer learning technique was based on pre-trained model such as MobileNetV2, ResNet, or DenseNet. Convolutional layers in the base model, which have previously undergone extensive training on sizable datasets, are excellent feature extractors. These layers were kept frozen to prevent overwriting the learned features. On top of the primary model, a few additional layers—possibly fully connected layers—were added to customize for the specific goal.

All experiments were conducted on Kaggle Notebooks using NVIDIA T4x2 GPUs (16 GB memory each, 2580 CUDA cores) with a fixed random seed of 42 across all models to ensure reproducibility and consistency in results across several runs. The dataset split into 70%, 15%, 15% for training, validation and testing respectively for all the models. The hyperparameter search spaces for learning rate were 0.002–0.008 and dropout was 0.2–0.4 and were not changed across various architectures of this paper for fair comparison. Results were obtained from a single run of experiments as there are constraints on computation which is a limitation of our work. This single-run design limits our ability to assess model stability and variance; reported metrics may not fully capture the risk of overfitting or performance fluctuations across different random initializations. All figures reporting accuracy, FLOPs, and training/testing time reflect results from this single experimental run. This single-run design limits our ability to assess model stability and variance; reported metrics may not fully capture the risk of overfitting or performance fluctuations across different random initializations. All figures reporting accuracy, FLOPs, and

training/testing time reflect results from this single experimental run. This single-run design limits our ability to assess model stability and variance; reported metrics may not fully capture the risk of overfitting or performance fluctuations across different random initializations. All figures reporting accuracy, FLOPs, and training/testing time reflect results from this single experimental run. Future work is planned to conduct experiments in five independent runs per model and measuring performance in terms of mean ± standard deviation, along with various statistical significance tests.

IV. Results and Discussion

A. CNN

CNN model was implemented as the model architecture shown in Fig. 1 for crack classification. CNN model performed well on the training set. Its testing accuracy was 97.99% after using optimized hyperparameter Setting. Model efficiency and computation operation must be traded off while training a CNN model.

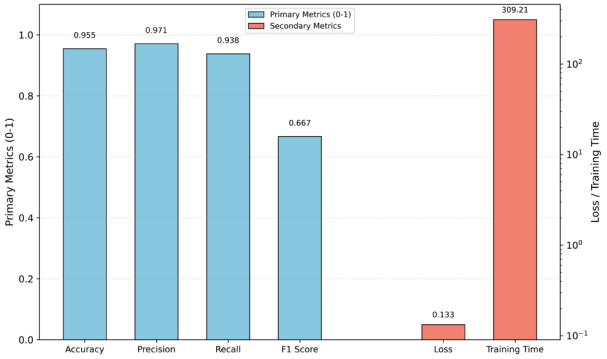

Fig. 7. Training results of CNN.

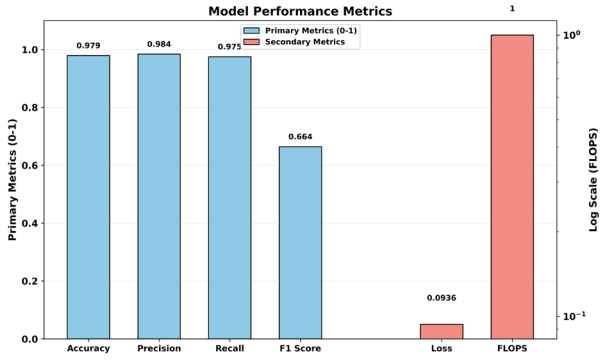

Fig. 8. Validation results of CNN.

The best hyperparameter setting is optimized for model training using Random Search method. The optimized hyperparameter setting showed the best overall performance with the highest training and validation accuracy of 95.5% and 97.9%. In addition, the training and testing time is 309.21 s and 10.94 s. Loss during training and validation have also been with in an acceptable range for optimized settings to a performance degradation defined as an accuracy decrease of ≥1%, F1-Score decrease of ≥0.05, or an increase in loss value of ≥0.01 g. Throughout this section, "considerable loss" refer relative to the best-performing hyperparameter setting. Although an explicit confusion matrix is not presented, the model performance has been evaluated using precision, recall, and F1-Score, which are directly derived from True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). True positives indicate crack images that are correctly detected and false negatives indicate that the non-crack images have been correctly identified. False positives are non-crack images that are misclassified as cracks and false negatives are missing cracks that are imperative in safety-sensitive situations. The low false negative rates indicate that the analysis will have high crack detection reliability. The Precision, recall and F1-Score for training, validation, and testing results also performed remains low (below 3% of total predictions). This shows the number of False positives and False negatives in prediction has been relatively less which is marginal. This clearly shows the CNN model with optimized setting being the best for Classification with reduced Flops, higher accuracy and loss within the acceptable threshold defined above. These have been represented in Figs. 7–9.

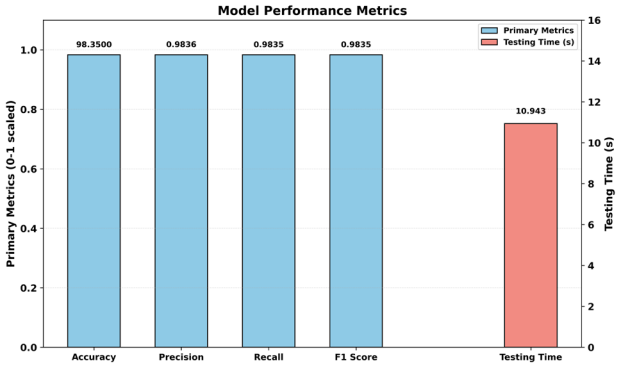

Fig. 9. Testing results of CNN.

B. CNN-LSTM

CNN-LSTM model performed well where CNN was used for extracting the features of images and LSTM used for Classification. Its testing accuracy was 99.23% after using hyperparameter tuning of Setting 2. The implementation of model architecture is shown in Fig. 2 While training a convolutional neural network (CNN-LSTM) model, accuracy and computation operation must be traded off. The three best hyperparameter configurations are selected for model training using Random Search optimization. Setting 2 showed the best overall performance with the highest training and validation accuracy of 98.12% and 98.8%. with reduced FLOPS, which is 1Gflops. Also training and testing time for setting 2 is 350.13 s and 13.32 s. Loss during validation was higher for Setting 2 (validation loss: 0.058) compared to Settings 1 and 3 (0.041 and 0.049), though the difference was within the acceptable threshold. Precision, recall, and F1-Scores all exceeded 0.98 across training, validation, and testing, with false positives and false negatives each below 1% of total predictions. Setting 2 achieved the best balance of accuracy and computational cost, outperforming Settings 1 and 3 while maintaining the same

1 GFLOPs footprint. These results confirm CNN-LSTM Setting 2 as the best-performing configuration among the CNN-LSTM variants. These have been represented as Figs. 10–12.

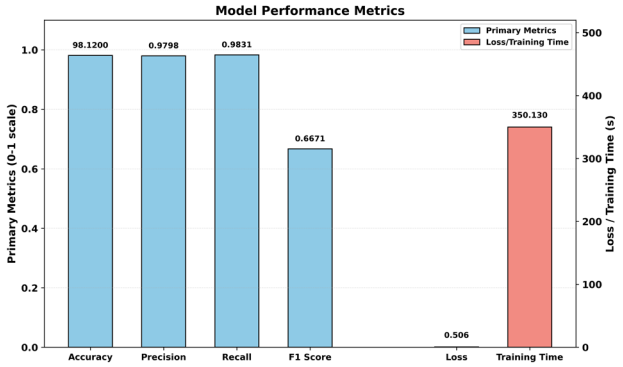

Fig. 10. Training results of CNN-LSTM.

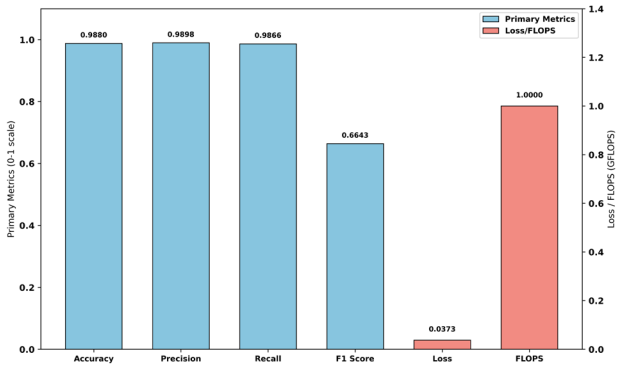

Fig. 11. Validation results of CNN-LSTM.

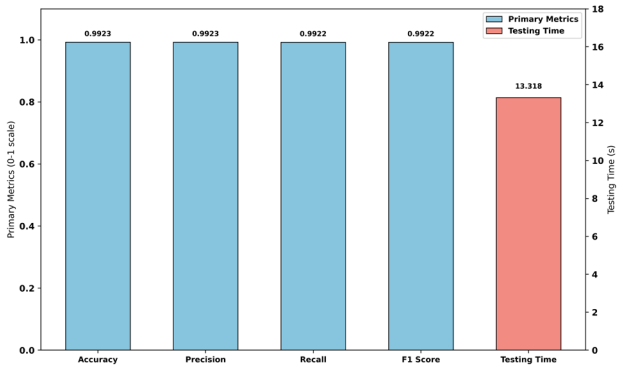

Fig. 12. CNN-LSTM testing results.

C. *MobileNet V2*

MobileNet V2 is a pre-trained model used for Classification of concrete cracks and resource-constrained devices like mobile phones. The implementation model adopted for MobileNet v2 which is transfer learning, is shown in Fig. 3. Its testing accuracy was 99.85% after using optimized hyperparameter setting. The optimized hyperparameter setting was selected using Random search that showed the best overall performance with the highest training and validation accuracy of 99.91% and 94.91%. with reduced FLOPS, which are 1Mflops (1 Mega flops). Also training and testing time for optimized setting is 320.51 s and 10.91 s. Validation loss remained within the acceptable threshold (increase <0.01 over the best setting). Precision, recall, and F1-Scores all exceeded 0.99, with false positives and false negatives each accounting for fewer than 0.2% of test predictions. MobileNetV2 achieved the best trade-off between accuracy and computational efficiency among all evaluated models, confirming its suitability for resource-constrained deployment. These have been represented in Figs. 13–15.

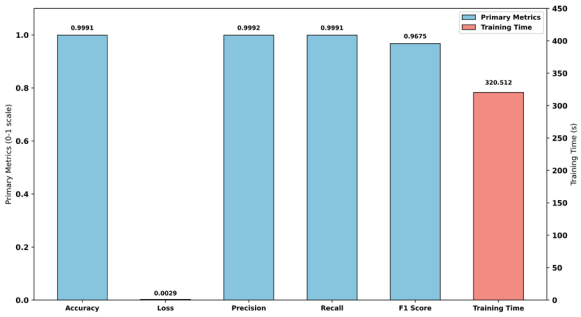

Fig. 13. Training results of MobileNetV2.

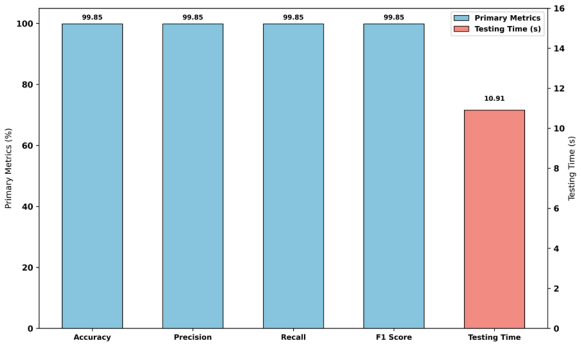

Fig. 14. Validation results of MobileNetV2.

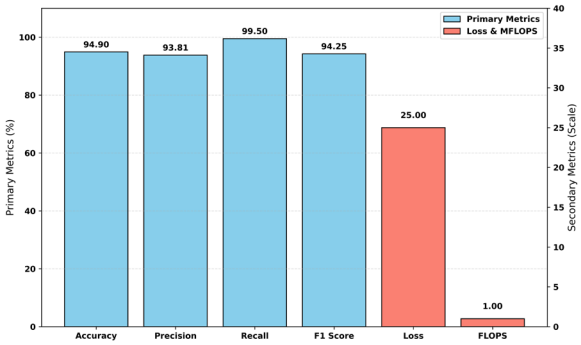

Fig. 15. Testing results of MobileNetV2.

D. *ResNet101*

ResNet-101 is another pre-trained model used for concrete crack classification but unsuited for resource constrained devices. The implementation model adopted for ResNet 101, which is transfer learning, is shown in Table IV.

The model has been optimized using random search optimization to suit resource-constrained devices. The model resulted in a testing accuracy of 99.85% with reduced Flops of 1.4 Mega Flops based on optimized hyper parameter setting. With respect to training and testing time, optimized setting resulted in 913.99 s and 15.04 s.

The optimized setting 1 has resulted in Flops which is 2 M Flops. Loss during training and validation have also been within the acceptable threshold for optimized setting. The Precision, recall and F1-Score of training, validation, and testing results all exceeded 0.99, with false positives and false negatives each below 0.2% of test predictions. In summary, ResNet101 achieved strong classification metrics with reduced FLOPs compared to the custom CNN and CNN-LSTM. These have been represented in Figs. 16–18.

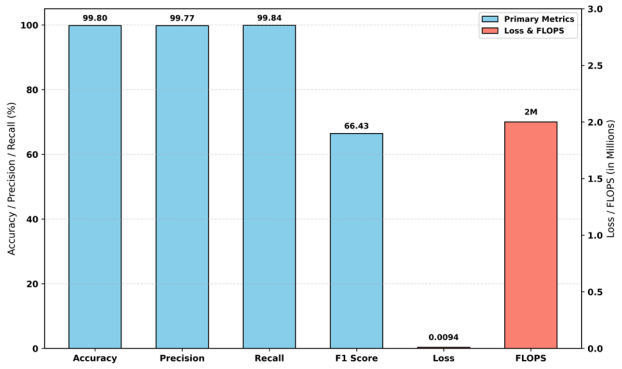

Fig. 16. Training results of Resnet 101.

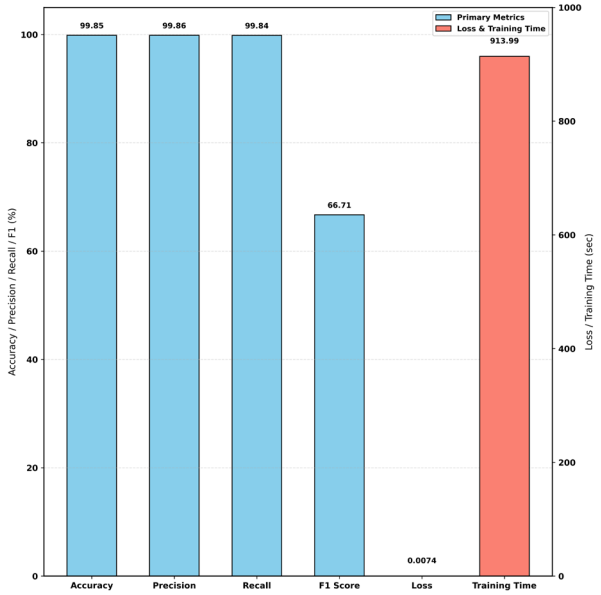

Fig. 17. Validation results of Resnet 101.

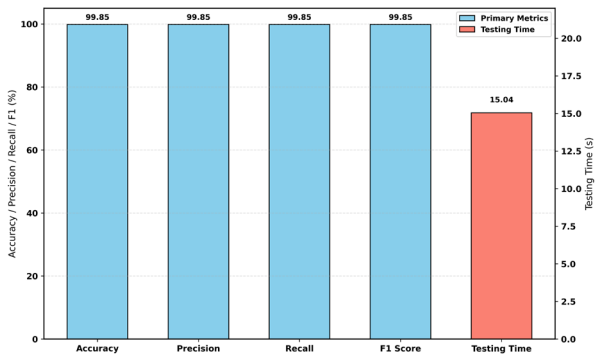

Fig. 18. Testing results of Resnet 101.

E. *DenseNet201*

DenseNet 201 is another pre-trained model used to classify concrete cracks and optimized resource-constrained devices. The implementation model used for DenseNet 201 is shown in Table V. Its testing accuracy was 99.89% after using hyperparameter Setting optimized using Random search. The best hyperparameter configuration or setting optimized is selected for model training using Random Search optimization. The optimized hyperparameter setting showed the best overall performance with the highest training and validation accuracy of 99.83% and 99.89%. with reduced FLOPS which is 3Mflops. In addition, the training and testing time for setting is 742.42 s and 12.28 s. Loss during training and validation were within the acceptable threshold for optimized setting. The precision, recall, and F1-Scores of trainings, validation, and testing results all exceeded 0.99, with false positives and false negatives each below 0.12% of total test predictions. DenseNet201 achieved the highest classification accuracy among all models, at the cost of higher FLOPs (3 MFLOPs) and longer training time (742.42 s) compared to MobileNetV2. These have been represented in Figs. 19–21.

From the analysis of results, it has been found that the trained models, which are CNN and CNN-LSTM, have used a more significant number of Flops as compared to Pretrained model. The reduced number of Flops have been used by custom model CNN with reduced training and testing time for optimized hyperparameter setting as compared to CNN-LSTM.

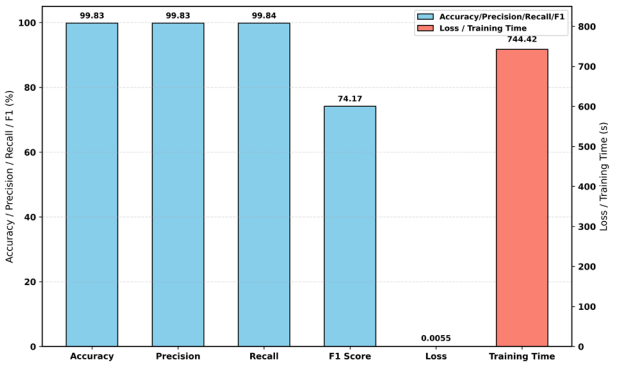

Fig. 19. Training results of DesNet 201.

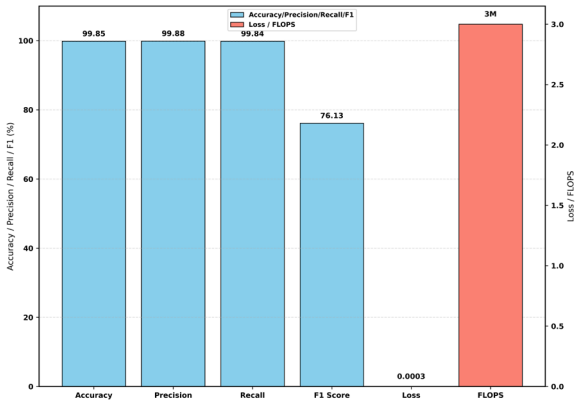

Fig. 20. Validation results of Desnet 201.

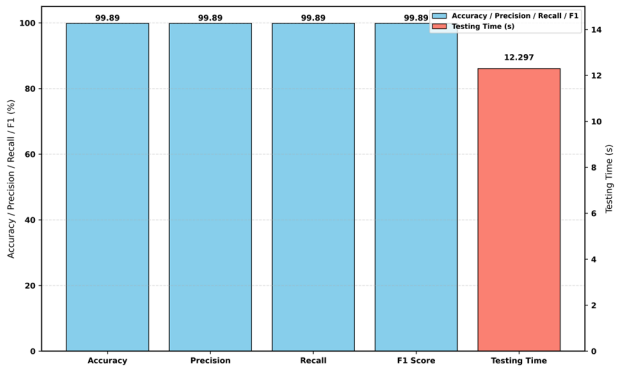

Fig. 21. Testing results of Desnet 201.

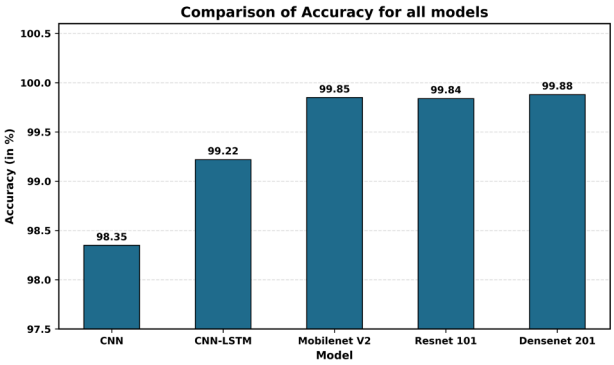

Fig. 22. Comparison of accuracy of all models.

CNN-LSTM also utilized the same number of Flops with optimized hyperparameter setting as CNN, resulting in increased training and testing time. The overall accuracy, precision, recall, F1-Score have performed well across both the custom model during training, testing and validation. Loss has also been considerably less in both models. But overall, the CNN model is the best-suited custom model compared to CNN-LSTM with reduced Flops of 1 Gflops with training and testing time of 309.21 s and 10.943 s. CNN-LSTM consumed the same 1 Gflops for optimized hyperparameter setting resulted in increased training and testing time of 350.13 s and 13.32 s with lesser accuracy of 99.23% compared to other settings. So, CNN outperformed CNN-LSTM with reduced Flops, training and testing time compared to CNN-LSTM. Accuracy, Loss, Precision, recall, F1-Score have performed considerably well across both models. In terms of pretrained models, three models used are MobileNet V2, ResNet 101 and DenseNet 201. All these models have been trained on larger datasets, making it convenient to freeze certain layers and run the model without having to train from scratch by using the transfer learning approach. MobileNet V2, lightweight deep learning model being specifically developed for resource constrained device with reduced number of convolution layers. On validating the performance of all three models including lightweight model, MobileNet V2 resulted in reduced Flops of 1 Mflops as compared to DenseNet 201 and ResNet 101. In addition, training and testing time also been considerably less for optimized hyperparameter setting. Resnet-101 and Densenet 201 also have performed well in terms of Flops against custom model. The overall performance of precision, recall, F1-Score and Loss was also considerably good across all pretrained models with slight variation. In summary, the transfer learning models have resulted in reduced Flops as compared to other custom models which are CNN and CNN-LSTM. On whole, the MobileNet V2 resulted in reduced Flops with lesser training and testing time though accuracy is considerably good across all models as compared to other transfer learning models which are ResNet-101 and DenseNet-201. The comparative analysis of all models in terms of accuracy, training time, testing time, and Flops are shown below in Figs. 22–25.

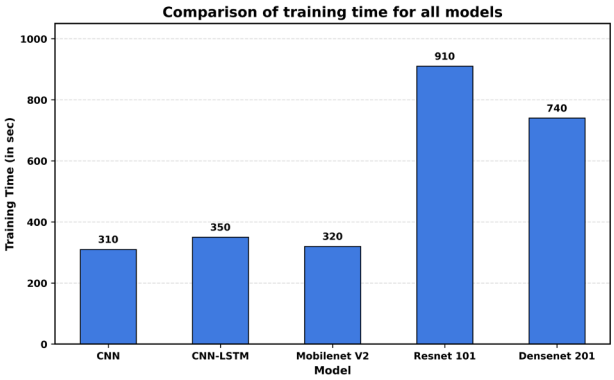

Fig. 23. Comparison of training times.

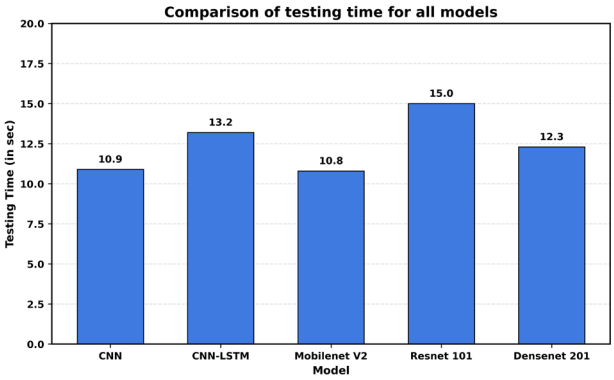

Fig. 24. Comparison of testing time of all models.

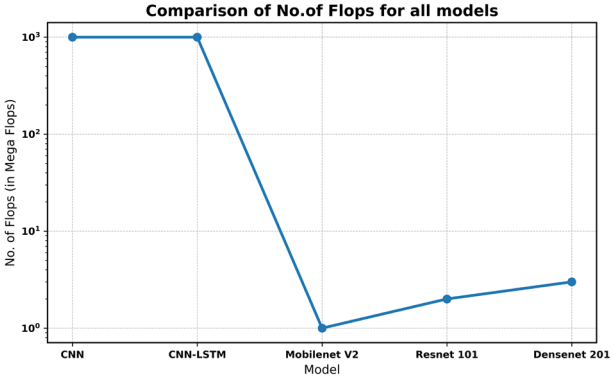

Fig. 25. Comparison of flops.

TABLE VI. BENCHMARKING OF RESULTS

Study	Dataset	Model Used	Performance Metrics	Hyperparameter / Notes
Development of an automatic detector of cracks in concrete using machine learning [1]	Cracked concrete dataset (2000 images).	CNN	79.9%, 2 class (cracked concrete & other)	Images resized to 64×64 pixel
	Cracked part—5335, chalked part—2761, joint part—4223, surface part—1533, other parts—4020		-	Activation function LReLU and drop out convolution layer and fully connected layer to prevent overtraining of the network
	-		73.3%, 3 class (Cracked concrete, chalk letter, others)	-
Automatic pavement crack detection using image processing techniques and neural network [2]	800 drone images of cracked pavement & crack-free pavement	ANN (Feedforward Neural Network)	82.50%	8 neurons in hidden layer
CNN-based crack detection for real concrete surface [3]	1250 real world crack images. cropped into 60,000 small images of 256×256 resolution with equal proportion of crack and no cracks.	CNN	99.39%	Weights-Xavier Method, Bias constant method
				momentum = 0.9; learning rate = 0.01; epochs = 15000
				Images resized to 224×224 pixels
Automatic classification of pavement crack using DCNN [14]	800 image sets of cracked pavement & crack free pavement	Feedforward Neural Network	Accuracy 82.5%;	8 neurons in hidden layer
			Precision 87%	
Crack detection of structures using deep learning framework [4]	2068 bridge crack images; 40,000 cracked and non-cracked concrete images	Modified LeNet-5	Asphalt Dataset-without PCA	Stochastic Batch
	400 images of asphalt cracks		Accuracy—82%, Precision—82%, Recall—82%;	Gradient Descent used for training all the datasets with the batch size of 32
	and non-crack images		Asphalt Dataset with PCA	Min delta value—0.01, Patience value—4
	-		Accuracy—68%, Precision—84%, Recall—48%;	No. of epochs—10, For Asphalt dataset
	-		Bridge Dataset without PCA	No. of epochs—30, Images resized to 1200×1200 pixels
	-		Accuracy—98%, Precision—98%, Recall—96%;	-
	-		Bridge Dataset-with PCA	-
	-		Accuracy—96%, Precision—96%, Recall—94%	-
DeepCrack: hierarchical feature learning [5]	537 images	Deep hierarchical Convolutional Neural Network	Mean IU 85.9%; F-Score—86.5%; 0.1 s per image	Images resized to 256×256, Batch size = 1; learning rate—0.0001; momentum—0.9; No. of epochs-200000
Wall Crack Multiclass classification [6]	2516 images of 224×224 dimensions	ResNet50 + KNN	Average AUROC—0.92; Average Precision—0.84; Average Recall—0.83; Average F-Score—0.83	High-quality images were used. Images were classified into light, medium, and severe. 7 convolutional layers with average pooling were used
Performance comparison of pretrained CNN [15]	40k image patches with 224×224 dimensions	AlexNet, VGG16, VGG19, GoogleNet, ResNet50, ResNet101, ResNet152	ResNet50 Model outperformed other models with accuracy of 96% and F-Score of 95%	19 convolutional layers with parameters of 144 M.
A Lightweight CNN based vision system for concrete crack detection [7]	SDNET2018 contains images of 256×256 pixels	CNN V1 (12 layers with 7 weight layers)	CNN V2 achieved an accuracy of 90.03% for pavement, 80.12% for walls, 82.86% for bridge decks with 0.222 memory and 12570 parameters	Dropout range 0.2 and 0.5. A total of 4 convolutional layers with ReLU activation function was integrated with CNN V2 model
		CNN V2 (13 layers with 6 weight layers)		
Surface crack detection with transfer learning [16]	808 crack images and 86 images with moss	GoogleNet Inception V3	Average accuracy—98.5%	Learning rate = 0.001 with Stochastic Gradient Descent was used to fine tune the model
Current Research	Concrete crack images for classification (40,000 images)	CNN	Test Accuracy—98.35%, Flops—1 G FLOPs, Training Time—309.21 s	Images resized to 120×120 pixels for faster training
		CNN-LSTM	Testing Time—99.23%, TFlops—1 G Flops, Training Time—350.13 s, Testing Time—13.32 s	
		MobileNetV2	Testing Accuracy—99.85%, Flops—1 M FLOPs, Training Time—320.51 s, Testing Time—10.91 s	

Though the proposed research has been validated against all DL models, including custom and transfer learning, our work was compared with other works in terms of efficiency and computation operation. These are shown below in Table VI. Building crack monitoring is very important for assessing the health of the building. Many technologies have been deployed to monitor buildings' health using crack detectors and other technologies. Even with the use of technologies for crack detection using crack detectors, a site inspector needs to assess the crack based on digital output, which depends on inspector's competence.

Vision-based crack detection is an ideal solution, which is a non-destructive evaluation method. With the advent of deep learning, a good amount of work has been done to analyze the cracks in the building and accordingly classify them, resulting in good performance using custom CNN and pre-trained models. The challenge in all the work is that they have employed a smaller dataset except for a few works where a good number of images were used which resulted in performance. In the training of deep models on low-resource devices, Adam was selected over SGD and RMSProp because it has adaptive learning rates, use of momentum, and fast converging ability. It requires minimal tuning by hand, manages gradients, whether noised or sparse, in the correct manner, and achieves training stability at low expense of computation overhead. Since no data augmentation (e.g., rotation, flipping, scaling) was utilized for the METU dataset, there is a likelihood of overfitting. Models might learn dataset-specific features, e.g., illumination, texture, or surface patterns, rather than generalizable features of cracks.

Such constraints may be applied while used in real-life conditions wherein the cracks occur in different orientations, sizes, and exposure conditions. Augmentation or cross-dataset validation may be included in future work and help reduce this risk and enhance generalizability. According to the observed performance, "considerable" loss corresponds to an accuracy decrease of ≥1%, F1-Score decrease of ≥0.05, or increase in loss of ≥0.01 relative to the best-achieving hyperparameter setting. Such thresholds give a precise, objective quantity to interpret performance deterioration and inform model selection and optimization choices. The METU dataset is a large dataset that contains diverse crack patterns and is used for the evaluation of the models. Cross-dataset generalization is the limitation of the present study, as the images under different geographical conditions and surface characteristics were not captured and used for evaluation. In the future, validation of the mentioned models and assessing robustness and generalization capability on various benchmark datasets can be focused on. The dataset itself is large with various image variants in crack patterns and surface textures, and hence, the data augmentation techniques are not applied on the dataset. Augmentation strategies can be applied to the dataset to improve the model robustness and reduce bias. k-fold cross-validation will also be implemented to assess the stability and variance of the models. The learning rate greatly influenced the convergence speed, stability, and accuracy of the models with the moderate rates of (0.002–0.008) providing the best balance. The dropout was used to prevent overfitting where the best values between (0.2 and 0.3) which enhanced the generalization and high values caused underfitting. Also, filters and dense units contributed to the trade-off between model capacity and computation cost (FLOPs).

Mostly learning rate and dropout were the important parameters, that indicates the necessity to carefully tune the model with the help of Random Search. Even though high accuracy (>99) is obtained, the overfitting is alleviated by dropout regularization and validation-based model selection. The number of images (40,000) is large enough to be generalized. Nonetheless, there is no data augmentation, which could create a possible dataset bias. Additional augmentation techniques that will be added to work in the future include rotation, scaling and the change in illumination to enhance the strength even more. In this study, the optimized models were not directly deployed on physical edge or IoT devices. Nonetheless, the feasibility of deployment is measured based on the computational metrics of MFLOPs, model size, and inference time. The decreased complexity and latency of lightweight models, especially MobileNetV2, suggests that it is applicable to real-time applications in resource-limited settings. The next step in work is implementations and validation on an embedded edge. The single run results are reported because of computational limitations. The next step in the work will involve several repetitions of mean plus standard deviation and statistical analysis to determine the difference in performance as valid.

All experiments are conducted and the FLOPs values after post optimization were calculated Kaggle with NVIDIA T4x2 GPUs under similar experimental setup, thus ensuring transparency and reproducibility. The measured FLOPs were 1 MFLOPs for MobileNetV2, 1.4 MFLOPs for ResNet101, and 3 MFLOPs for DenseNet201, reflecting only the fine-tuned classifier head, as the convolutional base was frozen during transfer learning. Full-network inference FLOPs are 300 MFLOPs for MobileNetV2, 7600 MFLOPs for ResNet101, and 8,400 MFLOPs for DenseNet201 on their architectures. In this study, MFLOPs, inference time, and model size were computed on a GPU-accelerated cloud environment with NVIDIA T4x2 on Kaggle notebooks. These values are proxy estimates for edge deployment feasibility not the direct benchmarks on physical edge hardware. The inference time of MobileNetV2 was around 10.91 s for the test set and approximately 0.27 ms per image. This value reduced in MFLOPs further which indicates that this model is suitable for the deployment on resource-constrained devices. But the actual performance on edge devices may be different for different devices such as Raspberry Pi 4 or NVIDIA Jetson Nano because of their memory, bandwidth constraints, thermal throttling, and the absence of GPU-level parallelism. In the future, the optimized MobileNetV2 model will be deployed on NVIDIA Jetson Nano and Raspberry Pi 4. Benchmark metrics such as latency, memory usage, CPU/GPU

utilization, power consumption, and FPS for real-time inference will be measured. The deployment claims in this paper should be considered GPU-based feasibility assessments rather than validated edge performance results.

V. Conclusion and Future Work

Building crack monitoring is very important for assessing the health of the building. Even with the use of technologies for crack detection using crack detectors, a site inspector needs to assess the crack based on digital output, which depends on inspector's competence. Vision-based crack detection is an ideal solution, which is a non-destructive evaluation method. So, this study focused on optimizing deep learning models such as CNN, CNN-LSTM and pre-trained models like MobileNetV2, ResNet101, and DenseNet201 for different optimized hyperparameter settings using the random search method. Experimental results demonstrate that MobileNetV2 performs well among all the models in terms of accuracy, precision, recall and F1-Score with reduced computational cost including training time, inference time and FLOPs. The results show that the models are highly suitable for deployment on edge devices such as AI-based IoT structural monitoring systems. The model can be expanded in future to classify crack severity stages at high levels of segmentation and to ensure robustness, data augmentation and cross-dataset validation. This model will be tested under the conditions of imbalanced data, with strategies such as class weighting and focal loss. Furthermore, it will be installed on edge machines (e.g., Raspberry Pi, Jetson Nano) to test its real-time performance and statistically checked during a series of consecutive steps. Also, the statistical significance tests and confidence interval analysis will be performed to ensure strict measures toward testing the reliability of performance differences between models. More advanced optimization methods including pruning, quantization, and neural architecture search will be considered to reduce the computational complexity and to improve efficiency in resource-constrained settings.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

VD Contributed to the literature survey, analysis of related work, and drafting of the survey section. Assisted in result interpretation and manuscript review. KM is responsible for the materials and methods, including methodology design, experimental setup, and implementation. Also contributed to data collection and validation of results. NMTYJ supervised the research study; contributed to the introduction, overall research design, and coordination of the work. Led the results and discussion, manuscript structuring, critical revisions, and final approval of the version to be published. US contributed to the results analysis, preparation of figures and tables, and drafting of the conclusion and future work section. Assisted in manuscript editing and proofreading. All authors approved the final version.

Acknowledgment

The authors would like to thank the Intel Unnati IoT Solutions Lab, Department of Networking and Communications, School of Computing, SRM Institute of Science and Technology (SRMIST), Kattankulathur, Tamil Nadu, India, for providing the necessary infrastructure and research facilities to carry out this work.

References

[1] L. Ali, F. Alnajjar, H. A. Jassmi *et al.,* "Performance evaluation of deep CNN-based crack detection and localization techniques for concrete structures," *Sensors*, vol. 21, no. 5, 1688, 2021.

[2] H. S. Munawar, A. W. A. Hammad, A. Haddad *et al.,* "Image-based crack detection methods: A review," *Infrastructures*, vol. 6, no. 8, 115, 2021

[3] H. Kaveh and R. Alhajj, "Recent advances in crack detection technologies for structures: A survey of 2022–2023 literature," *Frontiers in Built Environment*, vol. 10, 1321634, 2024.

[4] J. Yang, F. Lin, Y. Xiang *et al.,* "Fast crack detection using convolutional neural network," arXiv preprint, arXiv:2105.10892, 2021.

[5] M. M. M. Islam and J.-M. Kim, "Vision-based autonomous crack detection of concrete structures using a fully convolutional encoder–decoder network," *Sensors*, vol. 19, no. 19, 4251, 2019.

[6] Z. Li, D. Jia, Z. He *et al.,* "Multi-frame joint detection approach for foreign object detection in large volume parenterals," *Mathematics*, vol. 13, no. 5, 1333, 2025.

[7] M. Prunella, R. M. Scardigno, D. Buongiorno *et al.*, "Deep learning for automatic vision-based recognition of industrial surface defects: A survey," *IEEE Access*, vol. 11, pp. 43370–43423, 2023.

[8] M. M. Hosen, M. M. U. Sabbir, M. I. Hossain *et al.*, "Leveraging AI and sensor technologies for real-time structural health monitoring of in-service bridges," *Frontiers in Applied Engineering and Technology*, vol. 2, no. 1, pp. 135–163, 2025.

[9] S. Yamaguchi and T. Mizutani, "Development of an automatic detector of cracks in concrete using machine learning," *Procedia Engineering*, vol. 171, pp. 1250–1255, 2017.

[10] N. Shatnawi, "Automatic pavement cracks detection using image processing techniques and neural network," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 9, pp. 399–402, 2018.

[11] S. Li and X. Zhang, "Convolutional neural networks-based crack detection for real concrete surface," in *Proc. Sensors and Smart Structures Technologies Conf.*, 2018, pp. 1–7.

[12] A. Kumar, A. Kumar, A. K. Jha *et al.*, "Crack detection of structures using deep learning framework," in *Proc. 3rd Int. Conf. Intelligent Sustainable Systems (ICISS)*, Thoothukudi, India, 2020, pp. 526–533.

[13] Y. Liu, J. Yao, X. Lu *et al.*, "DeepCrack: A deep hierarchical feature learning architecture for crack segmentation," *Neurocomputing*, vol. 338, pp. 139–153, 2019.

[14] A. P. Wibowo, A. Adha, I. F. Kurniawan *et al.*, "Wall crack multiclass classification: Expertise-based dataset construction and learning algorithms performance comparison," *Buildings*, vol. 12, no. 12, 2135, 2022.

[15] L. Falaschetti, M. Beccerica, G. Biagetti *et al.*, "A lightweight CNN-based vision system for concrete crack detection on a low-power embedded microcontroller platform," *Procedia Computer Science*, vol. 207, pp. 3948–3956, 2022.

[16] R. Raushan, V. Singhal, and R. K. Jha, "Damage detection in concrete structures with multi-feature backgrounds using the YOLO network family," A*utomation in Construction*, vol. 170, 105887, 2025.

[17] V. Pandey and S. S. Mishra, "Enhanced concrete crack detection using YOLOv8: A multi-background approach," *Australian Journal of Structural Engineering*, pp. 1–16, 2025.

[18] V. Pandey and S. S. Mishra, "Lightweight frameworks for real-time crack monitoring in civil infrastructure," *Ultrasonics*, vol. 162, 107970, 2026.

[19] A. J. Fetterman, E. Kitanidis, J. Albrecht *et al.*, "Tune as you scale: Hyperparameter optimization for compute efficient training," arXiv preprint, arXiv:2301, 2023.
[20] A. U. R. Durrani, N. Minallah, N. Aziz *et al.*, "Effect of hyper-parameters on the performance of ConvLSTM based deep neural network in crop classification," *PLOS ONE*, vol. 18, no. 2, e0275653, 2023.
[21] M. Mayank. (Aug. 2020). Convolution neural networks explained. [Online]. Available: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
[22] J. Brownlee. (Aug. 2019) CNN long short-term memory networks. [Online]. Available: https://machinelearningmastery.com/cnn-long-short-term-memory-networks/
[23] S.-H. Tsang. (May 2019). Review: MobileNetV2—Lightweight model. [Online]. Available: https://towardsdatascience.com
[24] R. Modi. (Dec 2021). ResNet—Understand and implement from scratch. [Online]. Available: https://analyticsvidhya.com
[25] S.-H. Tsang. (Nov 2018). Review: DenseNet—Dense convolutional network. [Online]. Available: https://towardsdatascience.com
[26] Ç. F. Özgenel. (2019). Concrete crack images for classification. *Mendeley Data*. [Online]. Available: https://data.mendeley.com