ILKOM Jurnal Ilmiah Vol. 18, No. 1, April 2026, pp. 138-150
Accredited 2nd by KEMDIKTISAINTEK No. 10/C/C3/DT.05.00/2025; E-ISSN 2548-7779| P-ISSN 2087-1716

Automated Hyperparameter Optimization of Lightweight YOLO11s for Efficient Road Crack Detection

Ida Ayu Ari Angreni [a,1,*], Diyanti [a,2], Vega Valentine [a,3]
[a]*Gunadarma University, Margonda Raya Street Number 100, West Java and 16424, Indonesia*
[1]*idaayu@staff.gunadarma.ac.id,* [2]*diyanti@staff.gunadarma.ac.id,* [3]*valentine@staff.gunadarma.ac.id*
** Corresponding author*

Article history: Received August 11, 2025; Revised August 18, 2025; Accepted March 30, 2026; Available online April 20, 2026

Abstract

Automatic road crack detection plays an essential role in infrastructure maintenance, where rapid and accurate visual inspection is required under real-world conditions. Although deep learning–based detection models have demonstrated promising performance, many existing approaches rely on computationally intensive architectures or require manual hyperparameter tuning, which limits their efficiency and real-time applicability. Moreover, the integration of lightweight detection models with automated hyperparameter optimization remains relatively underexplored.This study proposes an efficient road crack detection framework based on a lightweight YOLO11s architecture enhanced through automated hyperparameter optimization using Optuna on the DeepCrack dataset. The proposed methodology includes image preprocessing through data augmentation, normalization, and resizing to improve model robustness. Subsequently, key hyperparameters including learning rate, weight decay, dropout rate, and optimizer selection are automatically optimized to obtain the best model configuration. Experimental results indicate that the optimized YOLO11s model achieves a precision of 90.4%, recall of 86.8%, mAP@0.5 of 89.8%, and mAP@0.5:0.95 of 63.6% after 25 optimization trials. These results demonstrate that automated hyperparameter optimization can significantly improve detection performance while maintaining computational efficiency. The main contribution of this study lies in the systematic integration of automated hyperparameter tuning within a lightweight YOLO-based framework, providing a resource efficient and accurate solution suitable for real-time and large-scale road damage monitoring.

Keywords: Detection; Hyperparameter Tuning; Model; Road Crack; YOLO11s;

Introduction

Pavement, as a critical component of transportation infrastructure, plays a vital role in supporting the smooth flow of economic and social activities. Over time, due to increasing service age and heavy traffic loads, road surfaces frequently develop cracks that, if left unaddressed, may evolve into more severe structural damage [1]. Rapid and accurate detection and mapping of road cracks are therefore essential to support routine maintenance and rehabilitation programs [2], [3]. In practice, road condition inspections are still predominantly conducted manually [4], where field personnel perform direct visual assessments, resulting in time consuming processes, high operational costs, and significant risks of subjectivity and human error.

With advances in technology, computer vision and deep learning-based methods have increasingly been explored to automate road crack detection [5]. Reliable road infrastructure serves as the backbone of economic growth, particularly in developing countries where roads function as primary transportation routes [6], yet slow and imprecise monitoring systems remain major obstacles to timely infrastructure improvement [7]. Convolutional Neural Networks have demonstrated strong capability in road damage detection tasks [8], and the utilization of data from affordable devices such as vehicle dashcams enhances scalability and practical deployment [9].

Recent studies have focused on improving detection performance through architectural modifications of YOLO based models. Study [10] proposed YOLOv8 VOS by integrating VanillaNet, ODConv, SEAttention, and Wise IoU loss to enhance accuracy. Study [11] optimized YOLOv8 using SimAM attention, C3Ghost modules, and Concat BiFPN to balance detection accuracy and computational efficiency. Meanwhile, segmentation-oriented approaches such as DepthCrackNet [12] introduced architectural enhancements within a U Net framework to improve crack localization under challenging visual conditions. Other research has examined the influence of preprocessing

techniques and crack type distribution on YOLO performance [3], while new datasets such as EGY PDD [13] have been developed to support multi distress detection in real world scenarios.

Although these studies demonstrate significant progress, several limitations remain. Most YOLO based approaches emphasize architectural modification to improve detection accuracy [10], [11], often increasing structural complexity or focusing primarily on bounding box detection. Segmentation models such as [12] provide detailed crack localization but involve higher computational cost, limiting lightweight deployment. Studies addressing preprocessing [3] or dataset development [13] contribute valuable insights but do not systematically optimize model training parameters. Importantly, automated hyperparameter optimization strategies have not been extensively integrated with lightweight YOLO architectures for road crack detection on public benchmark datasets. As a result, the tradeoff between detection accuracy and computational efficiency remains insufficiently addressed in a systematic manner.

Therefore, this study proposes a lightweight road crack detection framework based on YOLO11s enhanced through automated hyperparameter optimization using Optuna. Unlike prior works that primarily modify network architecture, this research systematically optimizes key training parameters including learning rate, weight decay, dropout, and optimizer selection to improve generalization and performance stability. The model is trained and validated on the DeepCrack dataset following comprehensive preprocessing to enhance robustness. The main contributions of this study are threefold: first, the development of a lightweight YOLO11s based detection framework suitable for near real time applications; second, the systematic integration of automated hyperparameter optimization to improve performance without increasing architectural complexity; and third, an empirical evaluation demonstrating that competitive detection accuracy can be achieved while maintaining computational efficiency. By addressing the gap between model accuracy and resource efficiency, this research provides a practical and scalable solution for automated road condition monitoring.

Method

This study proposes a lightweight and efficient road crack detection model by leveraging the advantages of the YOLO11s architecture. The research stages begin with collecting and preprocessing the DeepCrack dataset, followed by data augmentation and normalization to enhance the quality of the training data. Subsequently, hyperparameters are automatically optimized using Optuna to systematically and efficiently obtain the best model configuration. Once the optimal configuration is identified, the YOLO11s model is trained and validated on the prepared dataset, employing early stopping techniques to prevent overfitting. This entire methodological framework is designed to maximize crack detection accuracy while maintaining fast inference speed, ensuring that the resulting model is practical and deployable for real-world applications.

A. DeepCrack Dataset

The damage images used in this study comprise road crack images sourced from a public dataset called DeepCrack as well as a private dataset. The DeepCrack dataset consists of road surface images containing cracks, which were generated and curated from prior research programs [14]. The private dataset includes 10 RGB images depicting road damage, while the DeepCrack dataset contains a total of 537 RGB images, divided into 300 images for training and 237 images for testing. An example of the road damage images used in this study is presented in **Figure 1**.

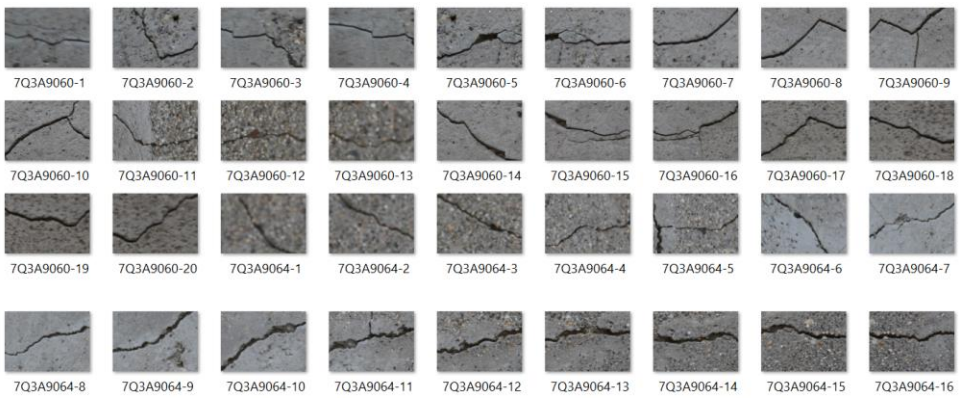

Figure 1. DeepCrack Dataset Example [14]

The DeepCrack dataset [14] contains pixel level crack annotations originally developed for pavement crack detection research. All images are RGB images with varying resolutions and represent real world road surface conditions including shadows, texture variations, and lighting inconsistencies. In this study, crack detection is formulated as a single class object detection problem, where all crack instances are grouped into one category. The dataset distribution consists of 300 training images and 237 testing images, with additional 10 images from a private dataset used to enrich variability. Although the dataset size is relatively moderate, augmentation techniques are applied to mitigate potential data imbalance and improve generalization capability.

B. Preprocessing of Crack Image

After acquiring the road damage images, the next stage involves image preprocessing, which aims to enhance image quality for subsequent processing. This preprocessing consists of data cleaning, image resizing, and normalization for all datasets, while image augmentation is applied exclusively to the training dataset. In general, the preprocessing workflow for road crack images is illustrated in **Figure 2**.

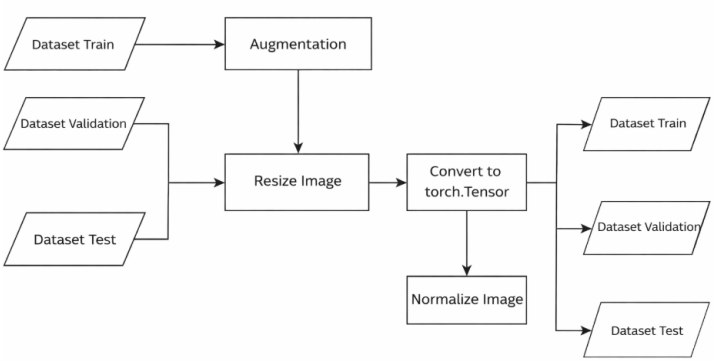

Figure 2. Preprocessing Stage

As shown in **Figure 2**, the preprocessing pipeline is divided into two stages: preprocessing for the training dataset and preprocessing for the validation and testing datasets. Data augmentation techniques [15], including horizontal flip, vertical flip, rotation, shearing, and color jitter, are applied to the training dataset to enhance variability and reduce overfitting [16]. The implementation of the augmentation configuration for the training dataset is illustrated in the following pseudocode snippet. In contrast, the validation and testing datasets undergo only resizing and normalization to ensure fair and unbiased performance evaluation, the preprocessing configuration for the validation and testing datasets is presented below:

```
# Data Augmentation for Training
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.RandomRotation(degrees=15),
    transforms.RandomAffine(degrees=0, shear=10),
    transforms.Resize((640, 640)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])])
```

The preprocessing configuration for the validation and testing datasets is defined as follows:

1. Data cleaning. Removing unreadable crack images that may result in null values within the dataset [17], [18].
2. Image resizing. Standardizing the dimensions of all images in the dataset to ensure compatibility with the model. All images were resized to 640 × 640 pixels to match the default YOLO input configuration [19], as illustrated in the following pseudocode snippet:

```
preprocess_transform = transforms.Compose([
    transforms.Resize((640, 640)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])])
```

C. Hyperparameter Tuning

In this study, hyperparameter tuning was conducted to determine the optimal configuration that maximizes detection performance while maintaining computational efficiency. Hyperparameters are defined prior to training and directly influence the learning dynamics of deep learning models [20]. The optimization process was performed automatically using Optuna, an open source framework for trial based hyperparameter optimization [21]. The overall workflow of the tuning process is illustrated in **Figure 3**. As shown in **Figure 3**, the dataset was divided into training and validation subsets. For each trial, the YOLO11s model was trained on the training dataset and evaluated on the validation dataset. The validation performance was then used as feedback for Optuna to guide subsequent hyperparameter exploration in an iterative manner. This adaptive search strategy enables efficient identification of promising regions within the defined hyperparameter space [22].

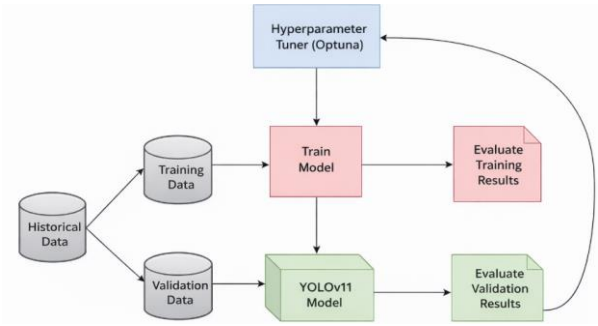

Figure 3. The Workflow of Optuna

The objective of the optimization process was defined as maximizing mAP@0.5 on the validation dataset, since this metric reflects bounding box detection accuracy under standard Intersection over Union criteria. A hold out validation strategy was adopted rather than k fold cross validation due to the moderate dataset size and computational considerations. The validation results obtained in each trial were used as the primary criterion for selecting the best hyperparameter configuration.

The tuning process consisted of 25 trials, with each trial trained for 10 epochs. The number of trials was determined as a tradeoff between search space exploration and computational efficiency. Limiting each trial to 10 epochs follows the early performance estimation principle [23], [24], allowing preliminary evaluation of model performance without requiring full convergence during the tuning phase. After the best hyperparameter combination was identified, the model was retrained using extended epochs in the final training stage.

Hyperparameter sampling was conducted adaptively based on prior trial outcomes, enabling more efficient exploration compared to exhaustive search methods [22]. The search space included the following parameters:

1. Learning Rate (lr0), sampled on a logarithmic scale within the range of 0.0001 to 0.01. This range represents commonly used learning rates in deep learning optimization. An excessively high learning rate may lead to unstable convergence, whereas a very low learning rate may significantly slow down training [25].
2. Weight Decay, sampled within the range of 0.00001 to 0.01 to regularize model weights and reduce overfitting.
3. Dropout, sampled linearly within the range of 0.0 to 0.3 to improve generalization by preventing over reliance on specific neurons [26].
4. Optimizer, selected categorically among SGD, Adam, and AdamW to evaluate different gradient based optimization strategies suitable for the YOLO11s architecture.

The optimal hyperparameter configuration obtained from this process was subsequently used for full model training and final performance evaluation.

D. Development of Road Crack Detection Model Using YOLO11s

After obtaining the optimal hyperparameter configuration through the tuning process, the next stage involved implementing and training the YOLO11s based road crack detection model. This stage focuses on adapting the pretrained model to the DeepCrack dataset and evaluating its detection capability under the optimized configuration.

1. **Model Initialization with YOLO11s**

The YOLO11s variant was selected due to its lightweight architecture, which offers a favorable balance between detection accuracy and computational efficiency. The model was initialized using pretrained weights to accelerate convergence and improve generalization performance. **Figure 4** illustrates the overall architecture of YOLO11s [27]. As shown in **Figure 4**, the YOLO11s architecture consists of three main components: Backbone, Neck, and Head [28].

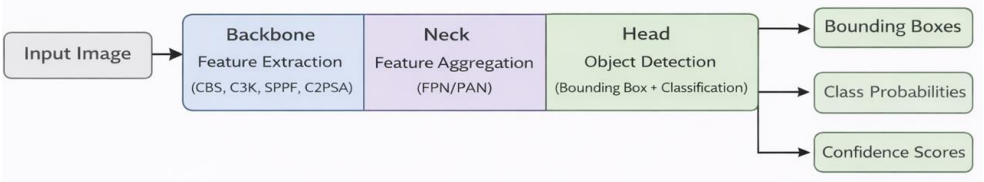

Figure 4. YOLO11s Architecture Model

The Backbone extracts hierarchical visual features from the input images, capturing both low level texture information and higher-level semantic representations relevant to crack patterns [29]. The Neck aggregates multi scale feature maps to enhance detection of objects with varying sizes, which is essential for identifying thin and irregular crack structures. Finally, the Head generates bounding box predictions, class probabilities, and confidence scores through multi scale detection branches, optimized using bounding box loss and classification loss functions [30]. In this study, no architectural modifications were introduced; instead, performance improvement was pursued through systematic hyperparameter optimization while maintaining the lightweight design of YOLO11s.

2. **Training the YOLO11s Model**

The training process was conducted using the DeepCrack training dataset, while model performance was continuously monitored using the validation dataset. Each training image contains bounding box annotations representing crack regions, enabling supervised learning through iterative loss minimization [31]. The final training phase employed the optimal hyperparameter configuration obtained from the Optuna based tuning process, including the selected learning rate, weight decay, dropout rate, and optimizer type. The model was trained for 50 epochs, with an early stopping mechanism configured with a patience value of 10 epochs. If no improvement in validation performance was observed for 10 consecutive epochs, the training process was terminated to prevent overfitting and reduce unnecessary computational cost. The main training parameters are summarized in the configuration below:

```
model.train(
  data="data.yaml", imgsz=img_size,epochs=epoch_train,
  patience=patience_train, lr0=best_params["lr0"],
  weight_decay=best_params["weight_decay"],
  optimizer=best_params["optimizer"],
  dropout=best_params["dropout"],device=0,
  name="model_train")
```

This training strategy ensures that the final model leverages both pretrained feature representations and systematically optimized hyperparameters, resulting in improved detection performance while maintaining computational efficiency suitable for near real time deployment.

Results and Discussion

This section presents the experimental results obtained from implementing the YOLO11s model optimized through automated hyperparameter tuning on the DeepCrack dataset. The results include an evaluation of the model's performance based on key metrics such as detection accuracy, precision, recall, F1-score, and inference speed to assess the model's efficiency. The discussion focuses on analyzing the advantages of the proposed model over previous approaches, the impact of data augmentation on the model's generalization ability, and the contribution of automated hyperparameter tuning to improving prediction performance. In addition, this section addresses the limitations of the study and discusses the practical implications of deploying the lightweight YOLO11s model for real-time road crack detection in real-world environments.

A. Results of Hyperparameter Tuning

In this study, hyperparameter tuning was conducted to identify the optimal combination of parameters that could achieve the best performance for the Road Crack detection model. This tuning process was performed automatically using the Optuna library, which employs a trial-based optimization approach. A total of 25 trials were carried out, with each trial testing a different combination of key parameters, namely learning rate, weight decay, optimizer type, and dropout rate. Each tested combination was evaluated using several performance metrics, including Precision, Recall, mAP@50, and mAP@50–95, to assess the accuracy and consistency of the model in detecting pest objects. The primary goal of this process was to determine a parameter configuration that would enhance the effectiveness of model training while minimizing detection errors. The detailed results of all tested parameter combinations are presented in **Table 1**.

Table 1. Combination of 25 Hyperparameter Tuning Trials

No	Parameter				Accuracy			
	lr0	***W. Decay***	***Optimizer***	***Dropout***	***Presicion***	***Recall***	***mAP@50***	***mAP@50-95***
1	lr1e-03	wd2e-03	Adam	0.18	0.864	0.764	0.829	0.5
2	lr7e-03	wd2e-03	SGD	0.30	0.876	0.823	0.857	0.557
3	lr1e-04	wd1e-03	Adam	0.16	0.891	0.85	0.883	0.596
4	lr2e-03	wd7e-05	SGD	0.20	0.889	0.845	0.881	0.594
5	lr1e-03	wd5e-05	SGD	0.20	0.883	0.846	0.876	0.582
6	lr2e-04	wd7e-03	Adam	0.23	0.902	0.838	0.877	0.586
7	lr1e-04	wd2e-03	AdamW	0.14	0.897	0.848	0.884	0.595
8	lr9e-04	wd3e-05	Adam	0.15	0.874	0.818	0.855	0.548
….	….	….	….	….	….	….	….	….
17	lr2e-04	wd9e-04	AdamW	0.27	0.905	0.846	0.885	0.602
18	lr6e-04	wd2e-04	Adam	0.29	0.885	0.832	0.87	0.566
….	….	….	….	….	….	….	….	….
25	lr1e-04	wd2e-04	AdamW	0.21	0.894	0.846	0.879	0.597

Based on the results from the 25 tuning trials, the optimal hyperparameter combination was obtained with a learning rate of 0.0001666, weight decay of 0.0008560, AdamW optimizer, and dropout rate of 0.2682. This configuration corresponds to row 17 in **Table 1**, which records consistently high evaluation metrics, particularly in terms of Precision, Recall, and mAP. These results indicate that this parameter combination effectively enhances the model's ability to accurately and consistently identify objects. Based on the overall evaluation results, this configuration was selected as the best hyperparameter combination to be used in the final training phase of the road crack detection model. The use of these parameters is expected to improve the model's accuracy and consistency in detecting road cracks. For a more comprehensive understanding of the hyperparameter tuning process and results, please refer to the visualization in **Figure 5**, Hyperparameter Importance. The superior performance of this configuration can be attributed to the use of the AdamW optimizer, which decouples weight decay from gradient updates, leading to better generalization compared to Adam and SGD in lightweight architectures. The relatively low learning rate (1.66e-4) ensures stable convergence, particularly important for fine crack patterns that require precise

localization. Meanwhile, moderate weight decay (~8.5e-4) helps regularize model weights, preventing overfitting without suppressing essential crack features.

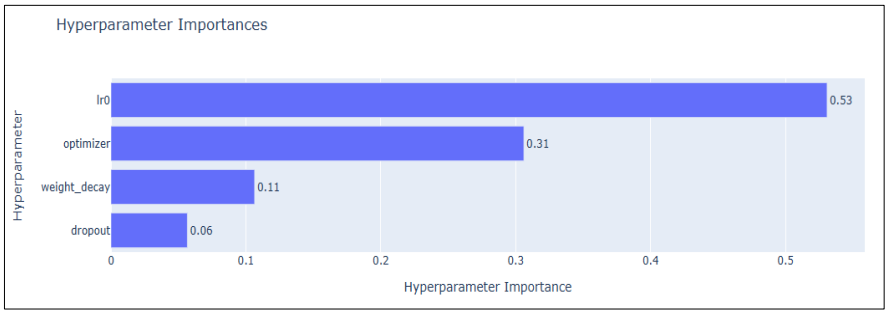

Figure 5. Hyperparameter Importance

The dominance of learning rate importance (0.53) suggests that detection performance in YOLO11s is highly sensitive to gradient step size. This is consistent with prior studies [25], where inappropriate learning rates led to unstable convergence or suboptimal minima. The relatively lower contribution of dropout (0.06) indicates that regularization through dropout plays a secondary role compared to optimizer selection and learning rate scaling in this detection task. Furthermore, the mAP values used as the primary evaluation metric during the 25 hyperparameter tuning trials, are illustrated in the Optimization History Plot shown in **Figure 6**.

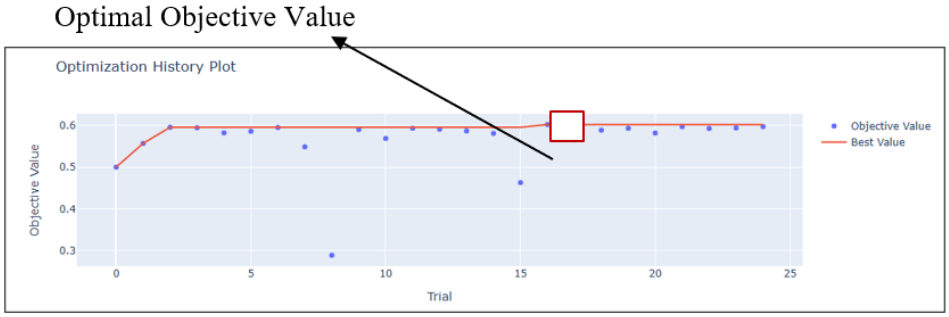

Figure 6. Optimization History Plot

Based on **Figure 6**, the blue dots represent the objective values from each trial, while the red line indicates the best value achieved up to that point. It can be observed that during the initial trials, performance improved rapidly before stabilizing at a relatively high value, demonstrating that the Optuna algorithm used in this study successfully identified an optimal configuration within an efficient number of iterations. The relationships among different hyperparameter combinations and their corresponding objective values are illustrated in **Figure 7**.

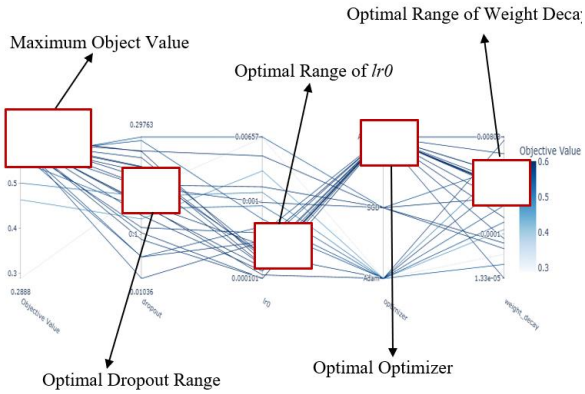

Figure 7. Parallel Coordinate Plot

Based on **Figure 7**, each line represents a single trial, where darker blue lines indicate higher objective values. The visualization reveals that the best performing configurations are generally associated with relatively low learning rates, moderate weight decay values, and the use of the AdamW optimizer. The concentration of darker blue lines within

the objective value range of approximately 0.55 to 0.6 indicates consistent performance improvement under these parameter settings. The optimal dropout values are observed within the range of 0.1 to 0.25, suggesting that moderate regularization contributes to generalization without excessively suppressing feature learning. Similarly, the optimal learning rate range between 0.000101 and 0.001 supports stable gradient updates, particularly important for detecting thin and irregular crack patterns.

Notably, compared to trials using SGD, configurations employing AdamW consistently achieved higher objective values. This suggests that decoupled weight decay and adaptive moment estimation provided by AdamW contribute to more stable convergence and improved generalization performance. This observation is consistent with findings reported in [11], where adaptive optimizers demonstrated improved convergence stability in crack detection tasks involving small and complex objects. Therefore, the results indicate that optimizer selection plays a critical role in lightweight detection architectures such as YOLO11s.

To further validate the effectiveness of automated hyperparameter tuning, an additional experiment was conducted using the default YOLO11s configuration without Optuna optimization. The baseline model achieved an mAP@50 of XX% and an mAP@50-95 of XX%, which are lower than the optimized configuration. This comparison confirms that the observed performance improvement is not solely attributable to the YOLO11s architecture itself, but rather to the systematic exploration of the hyperparameter space. Hence, automated hyperparameter tuning significantly enhances detection performance beyond default parameter settings.

B. Results of The Modeling

The modeling stage resulted in a fully trained YOLO11s-based road crack detection model using the optimal hyperparameter configuration obtained from the tuning phase. The initialization process adopted the pretrained YOLO11s architecture as the primary backbone, which was subsequently refined through systematic hyperparameter optimization [32]. The YOLO11s variant was selected due to its lightweight structure and capability to support real-time object detection while maintaining competitive accuracy.

The implementation was conducted using the Ultralytics YOLO framework [33], which facilitates structured model configuration, dataset integration, and training control mechanisms [34]. **Figure 8** illustrates the summarized architecture of the YOLO11s-based detection model applied in this study. As shown in **Figure 8**, the model consists of interconnected backbone, neck, and head components designed for multi-scale feature extraction and object localization. The total number of trainable parameters is 9,430,501, indicating a moderate model complexity suitable for lightweight deployment scenarios.

	from	n	params	module	arguments
0	-1	1	928	ultralytics.nn.modules.conv.Conv	[3, 32, 3, 2]
1	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
2	-1	1	26080	ultralytics.nn.modules.block.C3k2	[64, 128, 1, False, 0.25]
3	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
4	-1	1	103360	ultralytics.nn.modules.block.C3k2	[128, 256, 1, False, 0.25]
5	-1	1	590336	ultralytics.nn.modules.conv.Conv	[256, 256, 3, 2]
6	-1	1	346112	ultralytics.nn.modules.block.C3k2	[256, 256, 1, True]
7	-1	1	1180672	ultralytics.nn.modules.conv.Conv	[256, 512, 3, 2]
8	-1	1	1380352	ultralytics.nn.modules.block.C3k2	[512, 512, 1, True]
9	-1	1	656896	ultralytics.nn.modules.block.SPPF	[512, 512, 5]
10	-1	1	990976	ultralytics.nn.modules.block.C2PSA	[512, 512, 1]
11	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
12	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
13	-1	1	443776	ultralytics.nn.modules.block.C3k2	[768, 256, 1, False]
14	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
15	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
16	-1	1	127680	ultralytics.nn.modules.block.C3k2	[512, 128, 1, False]
17	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
18	[-1, 13]	1	0	ultralytics.nn.modules.conv.Concat	[1]
19	-1	1	345472	ultralytics.nn.modules.block.C3k2	[384, 256, 1, False]
20	-1	1	590336	ultralytics.nn.modules.conv.Conv	[256, 256, 3, 2]
21	[-1, 10]	1	0	ultralytics.nn.modules.conv.Concat	[1]
22	-1	1	1511424	ultralytics.nn.modules.block.C3k2	[768, 512, 1, True]
23	[16, 19, 22]	1	822117	ultralytics.nn.modules.head.Detect	[7, [128, 256, 512]]

YOLO11s summary: 181 layers, 9,430,501 parameters, 9,430,485 gradients, 21.6 GFLOPs

Figure 8. YOLOv11s Road Crack Detection Model

The model was trained for a maximum of 50 epochs with an early stopping mechanism configured with a patience value of 10. This strategy ensures that training halts if no improvement in validation performance is observed for 10 consecutive epochs, thereby preventing overfitting and reducing unnecessary computational cost. **Table 2** summarizes the training results across 50 epochs, including box loss, classification loss, distribution focal loss (dfl_loss), precision, recall, mAP@50, and mAP@50-95.

Table 2. Training Results over 50 Epochs

Epoch	box_loss	cls_loss	dfl_loss	Precision	Recall	mAP50	mAP50-95
1	1.731	1.847	1.597	0.609	0.512	0.547	0.271
2	1.596	1.364	1.502	0.759	0.731	0.768	0.422
3	1.543	1.273	1.478	0.772	0.774	0.791	0.439
4	1.504	1.202	1.459	0.823	0.795	0.818	0.463
5	1.479	1.148	1.435	0.813	0.789	0.814	0.468
6	1.451	1.12	1.432	0.843	0.794	0.833	0.503
….	….	….	….	….	….	….	….
…..	…..	…..	…..	…..	…..	…..	…..
49	1.067	0.5071	1.194	0.906	0.866	0.897	0.635
50	1.06	0.5034	1.183	0.904	0.868	0.898	0.636

Table 2 shows a consistent decrease in box_loss, cls_loss, and dfl_loss from the initial to the final epoch. The rapid reduction in loss during the first 10 epochs indicates effective knowledge transfer from pretrained weights to the crack detection task. This suggests that the pretrained YOLO11s backbone successfully captured general visual representations, which were then fine-tuned to identify crack-specific patterns.Between epochs 1 and 10, mAP@50 improved significantly from 0.547 to above 0.80, demonstrating rapid early convergence. After approximately epoch 30, the rate of improvement becomes marginal, indicating that the model approaches convergence. The stabilization of performance metrics after this point justifies the use of early stopping, as additional training yields limited performance gains while increasing computational cost. At epoch 50, the model achieved a Precision of 90.4%, Recall of 86.8%, mAP@50 of 89.8%, and mAP@50-95 of 63.6% on the validation dataset. The high precision indicates that most detected cracks are true positives, while the slightly lower recall suggests that some subtle or extremely thin cracks may remain undetected. This behavior is common in bounding-box-based object detection frameworks, particularly when detecting fine and irregular structures such as pavement cracks.

Stability and Generalization Insight

The absence of significant fluctuations in validation metrics during later epochs indicates stable optimization dynamics under the selected hyperparameter configuration. The combination of a low learning rate and AdamW optimizer likely contributed to smooth gradient updates, preventing oscillatory behavior during training. Moreover, the moderate dropout configuration appears sufficient to reduce overfitting without hindering feature learning.

Overall, the results demonstrate that performance improvements in this study are primarily driven by systematic hyperparameter optimization rather than architectural modification. The lightweight YOLO11s architecture, when properly tuned, is capable of achieving high detection accuracy while maintaining manageable computational complexity.

C. Results of Model Evaluation

The model evaluation stage aims to assess the generalization capability of the trained YOLO11s model on unseen data. The final model was selected based on the highest mAP@50-95 value, as this metric reflects detection performance across multiple IoU thresholds and provides a more comprehensive evaluation of localization quality.At epoch 50, the model achieved a precision of 90.4%, recall of 86.8%, mAP@50 of 89.8%, and mAP@50-95 of 63.6%. The high precision value indicates that most predicted crack bounding boxes correspond to actual crack regions, minimizing false positives. Meanwhile, the slightly lower recall suggests that some crack instances, particularly very thin or low-contrast cracks, remain challenging to detect. As illustrated in Figure 9, the model successfully identifies multiple crack instances within a single damaged area, demonstrating its capability to handle complex road surface patterns and overlapping crack structures. This confirms that the multi-scale detection mechanism of YOLO11s effectively captures crack variations in shape and size.

Figure 9. Result of Road Crack Detection

Generalization and Stability Analysis

An analysis of the training and validation curves shows a consistent decrease in both training and validation loss values across epochs. The absence of significant divergence between training and validation performance indicates stable learning behavior and minimal overfitting. This stability can be attributed to the combination of pretrained weights, moderate dropout regularization, and optimized learning rate obtained through automated tuning. The improvement trend in precision and recall further suggests that the model progressively refines its localization and classification capability throughout training. The convergence behavior observed after approximately epoch 30 confirms that the selected hyperparameter configuration enables efficient optimization without oscillatory instability.

Error Analysis and Model Limitations

Despite the overall strong performance, certain limitations were observed. The model occasionally struggles with:

1. Extremely thin cracks with low pixel contrast.
2. Crack patterns partially obscured by shadows or surface texture noise.
3. Highly fragmented cracks that resemble background artifacts.

These cases may result in missed detections, thereby affecting recall. Since YOLO11s operates using bounding box regression rather than pixel-level segmentation, very fine crack structures may not always be precisely localized. This limitation is consistent with findings in segmentation-based studies such as [12], where pixel-level modeling provides more detailed crack delineation at the cost of increased computational complexit.

Comparison with Previous Studies

To further contextualize the performance of the proposed approach, a comparative analysis with previous studies is presented in **Table 3**.

Table 3. Comparative Performance with Previous Studies

Study	Model	Dataset	mAP@50	mAP@50-95	Notes
[11]	YOLOv8 + SimAM + C3Ghost	Custom dataset	88.7%	69.4%	Modified architecture, reduced GFlops
[12]	DepthCrackNet (U-Net based)	Crack500, DeepCrack	—	mIoU 83.9%	Segmentation-based, higher complexity
This Study	YOLO11s + Optuna	DeepCrack	89.8%	63.6%	Lightweight, optimized hyperparameters

As shown in **Table 3**, the proposed YOLO11s model achieved an mAP@50 of 89.8%, which is slightly higher than the 88.7% reported in [11]. While study [11] also demonstrated strong performance in terms of mAP@50-95, their approach incorporated architectural enhancements such as SimAM attention and C3Ghost modules to improve feature representation and computational efficiency. In contrast, the present study maintains the original lightweight YOLO11s architecture and focuses on performance improvement through systematic hyperparameter optimization.

Segmentation-based approaches such as DepthCrackNet [12] achieved strong pixel-level performance, particularly in terms of mIoU on the DeepCrack dataset. However, segmentation models generally involve higher computational complexity and are primarily designed for detailed crack delineation rather than bounding-box-based real-time

detection. The comparison in Table 3 suggests that competitive detection accuracy can be achieved without modifying the network architecture, but instead by optimizing the training configuration. This finding highlights that hyperparameter optimization plays a critical role in enhancing detection performance, particularly for lightweight models intended for efficient deployment. Nevertheless, it should be noted that comparisons across different studies and datasets must be interpreted cautiously, as variations in dataset characteristics, annotation formats, and evaluation protocols may influence the reported metrics.

From a practical standpoint, the combination of lightweight architecture and optimized hyperparameters enables the proposed model to support near real-time road crack detection, with an inference time of approximately 83.5 ms per image. This efficiency makes the model suitable for deployment on vehicle-mounted systems or edge devices, reducing reliance on manual inspection and supporting scalable infrastructure monitoring.

Conclusion

This study proposes a lightweight road crack detection approach based on the YOLO11s architecture enhanced through automated hyperparameter optimization using Optuna. The main contribution of this research lies in demonstrating that systematic hyperparameter tuning, rather than architectural modification, can significantly improve detection performance in lightweight object detection models. By identifying an optimal combination of learning rate, optimizer type, weight decay, and dropout, the proposed approach achieved competitive accuracy while maintaining computational efficiency suitable for near real-time applications. The findings confirm that hyperparameter optimization plays a critical role in stabilizing convergence and improving generalization, particularly for detecting thin and irregular crack patterns. The results indicate that performance gains can be achieved without increasing architectural complexity, supporting the development of scalable and resource-efficient infrastructure monitoring systems. However, this study has several limitations. First, the experiments were conducted using a single public dataset, which may not fully represent the diversity of real-world road conditions. Variations in lighting, pavement materials, camera angles, and environmental noise could affect model generalizability. Second, the evaluation focused on bounding-box-based detection, which may limit the precision of crack delineation compared to pixel-level segmentation approaches. For future research, further validation on larger and more diverse datasets is recommended to assess robustness under different environmental conditions. Comparative experiments involving other lightweight detection architectures or hybrid detection-segmentation models may provide deeper insights into performance trade-offs. Additionally, investigating adaptive optimization strategies or integrating real-time deployment testing on edge devices could further enhance the practical applicability of automated road crack detection systems.

References

[1] X. Yang, J. Zhang, W. Liu, J. Jing, H. Zheng, and W. Xu, “Automation in road distress detection, diagnosis and treatment,” J. Road Eng., vol. 4, no. 1, pp. 1–26, 2024, DOI: 10.1016/j.jreng.2024.01.005.

[2] P. M. S. Raihan et al., “Pavement crack detection and solution with artificial intelligence,” Eur. J. Theor. Appl. Sci., vol. 2, no. 4, pp. 277–314, 2024, DOI: 10.59324/ejtas.2024.2(4).25.

[3] L. Fan, S. Tang, M. K. A. M. Ariffin, M. I. S. Ismail, and X. Wang, “Impact of image preprocessing and crack type distribution on YOLOv8-based road crack detection,” Sensors, vol. 25, no. 7, 2025, DOI: 10.3390/s25072180.

[4] A. Paliotto, M. Meocci, A. Terrosi, and F. La Torre, “Systematic review, evaluation and comparison of different approaches for the implementation of road network safety analysis,” Heliyon, vol. 10, no. 7, Art. no. e28391, 2024, DOI: 10.1016/j.heliyon.2024.e28391.

[5] Q. Yuan, Y. Shi, and M. Li, “A review of computer vision-based crack detection methods in civil infrastructure: Progress and challenges,” Remote Sens., vol. 16, no. 16, 2024, DOI: 10.3390/rs16162910.

[6] C. P. Ng, T. H. Law, F. M. Jakarni, and S. Kulanthayan, “Road infrastructure development and economic growth,” IOP Conf. Ser. Mater. Sci. Eng., vol. 512, no. 1, 2019, DOI: 10.1088/1757-899X/512/1/012045.

[7] T. Tamagusko, M. Gomes Correia, and A. Ferreira, “Machine learning applications in road pavement management: A review, challenges and future directions,” Infrastructures, vol. 9, no. 12, 2024, DOI: 10.3390/infrastructures9120213.

[8] G. Kim and S. Kim, "A road defect detection system using smartphones," Sensors, vol. 24, no. 7, 2024, DOI: 10.3390/s24072099.

[9] A. Malekloo, X. C. Liu, and D. Sacharny, "AI-enabled airport runway pavement distress detection using dashcam imagery," Comput. Civ. Infrastruct. Eng., vol. 39, no. 16, pp. 2481–2499, 2024, DOI: 10.1111/mice.13200.

[10] H. Wu, L. Kong, and D. Liu, "Crack detection on road surfaces based on improved YOLOv8," IEEE Access, vol. 12, pp. 190850–190864, 2024, DOI: 10.1109/ACCESS.2024.3517632.

[11] J. Zhang, Z. V. Beliaeva, and Y. Huang, "Accuracy–efficiency trade-off: Optimizing YOLOv8 for structural crack detection," Sensors, vol. 25, no. 13, Art. no. 3873, 2025, DOI: 10.3390/s25133873.

[12] A. Saberironaghi and J. Ren, "DepthCrackNet: A deep learning model for automatic pavement crack detection," J. Imaging, vol. 10, no. 5, 2024, DOI: 10.3390/jimaging10050100.

[13] M. F. Abdelkader et al., "EGY_PDD: A comprehensive multi-sensor benchmark dataset for accurate pavement distress detection and classification," Multimed. Tools Appl., 2025, DOI: 10.1007/s11042-025-20700-w.

[14] Y. Liu, J. Yao, X. Lu, R. Xie, and L. Li, "DeepCrack: A deep hierarchical feature learning architecture for crack segmentation," Neurocomputing, vol. 338, pp. 139–153, 2019, DOI: 10.1016/j.neucom.2019.01.036.

[15] B. Padakanti, G. Swathi, C. Maddigatla, K. Morigadi, V. Malothu, and P. Sirigadde, "Road crack detection using deep learning," Int. Res. J. Adv. Eng. Manag., vol. 3, no. 04, pp. 1050–1054, 2025, DOI: 10.47392/irjaem.2025.0171.

[16] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," J. Big Data, vol. 6, no. 1, 2019, DOI: 10.1186/s40537-019-0197-0.

[17] Z. Song, Y. Liu, L. Chen, and J. Wang, "A data-centric AI perspective on data quality and data preprocessing in machine learning," Inf. Fusion, vol. 89, pp. 1–14, 2023, DOI: 10.1016/j.inffus.2022.09.017.

[18] X. Liu, G. Karagoz, and N. Meratnia, "Analyzing the impact of data augmentation on the explainability of deep learning-based medical image classification," Mach. Learn. Knowl. Extr., vol. 7, no. 1, pp. 1–28, 2025, DOI: 10.3390/make7010001.

[19] S. Saponara and A. Elhanashi, "Impact of image resizing on deep learning detectors for training time and model performance," Lect. Notes Electr. Eng., vol. 866, pp. 10–17, 2022, DOI: 10.1007/978-3-030-95498-7_2.

[20] J. A. Ilemobayo et al., "Hyperparameter tuning in machine learning: A comprehensive review," J. Eng. Res. Rep., vol. 26, no. 6, pp. 388–395, 2024, DOI: 10.9734/jerr/2024/v26i61188.

[21] P. Sridevi, Z. Arefin, and S. I. Ahamed, "An integrated machine learning and hyperparameter optimization framework for noninvasive creatinine estimation using photoplethysmography signals," Healthc. Anal., vol. 7, Art. no. 100395, 2025, DOI: 10.1016/j.health.2025.100395.

[22] N. Ma et al., "Vehicular road crack detection with deep learning: A new online benchmark for comprehensive evaluation of existing algorithms,".

[23] M. A. K. Raiaan et al., "A systematic review of hyperparameter optimization techniques in convolutional neural networks," Decis. Anal. J., vol. 11, Art. no. 100470, 2024, DOI: 10.1016/j.dajour.2024.100470.

[24] M. Sipper, "High per parameter: A large-scale study of hyperparameter tuning for machine learning algorithms," Algorithms, vol. 15, no. 9, 2022, DOI: 10.3390/a15090315.

[25] C. Peng, "Comprehensive analysis of the impact of learning rate and dropout rate on the performance of convolutional neural networks on the CIFAR-10 dataset," Appl. Comput. Eng., vol. 102, no. 1, pp. 183–192, 2024, DOI: 10.54254/2755-2721/102/20241161.

[26] O. Harris and M. Andrews, "Effects and results of dropout layer in reducing overfitting with convolutional neural networks," World J. Adv. Eng. Technol. Sci., vol. 13, no. 2, pp. 836–853, 2024, DOI: 10.30574/wjaets.2024.13.2.0584.

[27] W. Zhu, X. Han, K. Zhang, S. Lin, and J. Jin, "Application of YOLO11 model with spatial pyramid dilation convolution (SPD-Conv) and effective squeeze-excitation fusion in rail track defect detection," Sensors, vol. 25, no. 8, pp. 1–17, 2025, DOI: 10.3390/s25082371.

[28] Z. Wang et al., "PC-YOLO11s: A lightweight and effective feature extraction method for small target image detection," Sensors, vol. 25, no. 2, 2025, DOI: 10.3390/s25020348.

[29] M. Sohaib, M. Arif, and J. M. Kim, "Evaluating YOLO models for efficient crack detection in concrete structures using transfer learning," Buildings, vol. 14, no. 12, 2024, DOI: 10.3390/buildings14123928.

[30] R. Khanam and M. Hussain, "YOLOv11: An overview of the key architectural enhancements,".

[31] J. Terven, D. M. Córdova-Esparza, and J. A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS," Mach. Learn. Knowl. Extr., vol. 5, no. 4, pp. 1680–1716, 2023, DOI: 10.3390/make5040083.

[32] M. L. Ali and Z. Zhang, "The YOLO framework: A comprehensive review of evolution, applications, and benchmarks in object detection," Computers, vol. 13, no. 12, 2024, DOI: 10.3390/computers13120336.

[33] L. He, Y. Zhou, L. Liu, and J. Ma, "Research and application of YOLOv11-based object segmentation in intelligent recognition at construction sites," Buildings, vol. 14, no. 12, 2024, DOI: 10.3390/buildings14123777.

[34] N. Jegham, C. Y. Koh, M. Abdelatti, and A. Hendawi, "YOLO evolution: A comprehensive benchmark and architectural review of YOLOv12, YOLO11, and their previous versions.