IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 27 March 2023, accepted 9 April 2023, date of publication 13 April 2023, date of current version 18 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3266737

APPLIED RESEARCH

An Aggressively Pruned CNN Model With Visual Attention for Near Real-Time Wood Defects Detection on Embedded Processors

WEI-HAN LIM, MOHAMMAD BABRDEL BONAB, (Member, IEEE), AND KEIN HUAT CHUA, (Senior Member, IEEE)

Lee Kong Chian Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, Kajang, Selangor 43000, Malaysia

Corresponding author: Mohammad Babrdel Bonab (babrdel@utar.edu.my)

This work was supported by the Research University Grant from Universiti Tunku Abdul Rahman (UTAR) under Grant IPSR/RMC/UTARRF/2019-C2/M01.

ABSTRACT The use of Convolutional Neural Networks (CNN) for the application of wood defects detection has gained significant attention in recent years. In industrial settings, these tasks are typically performed in a strict and consistent environment, making the use of large and complex CNN models unnecessary. Despite this, recent research has continued to focus on the use of such models to achieve increasingly accurate detections. These models require costly machines for inference, making adoption less likely especially for manufacturers in the developing nations. In view of this limitation, this paper proposes a set of strategies to achieve a highly efficient CNN model for fast and accurate wood defects detection based on the YOLOv4-Tiny architecture. The model has been improved with Efficient Channel Attention (ECA) to better select multi-scale features from the backbone network and has been drastically reduced in size through an iterative pruning and recovery process. This results in an 88% reduction in model parameters while retaining accuracy comparable to most state-of-the-art (SOTA) methods. Consequently, the model can perform near real-time inference directly on a general-purpose embedded processor without external hardware accelerators. This research hopes to motivate the development of efficient defect detectors that can run on low-cost embedded devices.

INDEX TERMS Attention, channel pruning, CNN, defect detection, embedded, lightweight, wood.

I. INTRODUCTION

Wood is a scarce but essential resource. The wood products industry contributes significantly to the economies of many developing nations. A common nuisance faced by the wood industry is the presence of wood defects. This is because defects negatively impact the appearance and structural strength of wood, thus reducing the commercial value of its derived products [1]. In fact, the average yield reduction due to wood manufacturing defects are about 10% [2]. As quality wood is often in high demand, wood manufacturers have employed considerable manpower to identify and eliminate defective woods during the manufacturing process to avoid compromising the quality of their products.

The associate editor coordinating the review of this manuscript and approving it for publication was Alba Amato.

Traditional approaches to wood defects detection utilize human labor to perform manual eye inspection. The downside of this approach is that humans often suffer from eye fatigue after long hours of work which can result in high human error. Besides that, humans typically make subjective judgements during decision making which can cause inspection results to become inconsistent. However, with the advent of machine vision technology, automated solutions now exist to improve the efficiency of wood defects detection. By utilizing automated visual inspection systems, it can not only improve the inspection accuracy and consistency, but also increase its efficiency and throughput due to the capability for high-speed and non-stop operation of automated systems.

Conventional methods for automated wood defects detection are based on manually curated features. These features are typically derived from the color [3] or texture [4] of wood.

Prasitmeeboon and Yau [5] have performed wood defect detection on wood particleboards based on the hue and saturation channels of the input images. Riana et al. [6] performed color conversion before segmenting defective regions from wood images while utilizing the Gray Level Co-occurrence matrix (GLCM) to extract textural features for the classification of wood defects of Swietenia Mahagoni wood. Qayyum et al. [1] have also used the GLCM method as textural features while using a Particle Swarm Optimization (PSO) trained neural network for the classification of wood defects. Besides GLCM, other feature extraction algorithms have also been proposed for wood defects detection. This include the use of Local Binary Pattern (LBP) [7], Law's Texture Energy Measures (LTEM) [8] as well as the Dual-Tree Complex Wavelet Transform (DTCWT) [9] algorithms. Some researchers have also combined several textural features for the recognition of wood defects. Li et al. [10] have combined LBP with their newly proposed local binary differential excitation pattern (LB_DEP) model to create a two-dimensional (2D) histogram as the features for the classification of birch wood defects. Hittawe et al. [11] have also proposed to combine both LBP and the Speeded Up Robust Features (SURF) for the classification of knot and crack defects. Besides having different feature extraction algorithms, conventional vision inspection methods also utilize various algorithms for the detection and recognition of wood defects. These include the use of Artificial Neural Networks (ANN) [1], [7], [8], k-Nearest Neighbours (kNN) algorithm [12], Support Vector Machines (SVM) [5], [11], gaussian distance [10] and compressed sensing theory [9]. The problem with conventional methods is that a high level of human expertise is required for its development. This is a result of having no clear rules for the determination of optimal features for wood defects detection [13]. Additionally, separate designs are required for the feature extractor and classifier which makes the development of wood detectors tedious and sub-optimal [14].

With the rapid increase in Graphics Processing Unit (GPU) computational power and the advancement of CNNs, it is now possible to train deep CNN networks to perform automatic feature extraction from image data. These networks learn the optimal set of features for a particular problem based on its training dataset. CNN-based networks have proven effective in various fields of engineering such as infrared target detection [15] and electroencephalography [16]. In manufacturing, CNN have been extensively used for the purpose of visual inspection including applications for civil works [17], [18], railroad tracks [19], [20], electronic components [21], [22] and steel surfaces [23], [24], [25], [26], [27], [28], [29]. In recent years, the use of CNN for wood defects detection has quickly become established. The models used for wood defects detection can be generally split into two categories, namely two-shot detectors and single-shot detectors. A two-shot detector performs defect detection in two stages. The first stage is known as the region proposal stage where suggestions are provided for regions of interests (RoI) that contain defects. These regions are then sent to the second stage which performs classification and localization refinement for the RoIs simultaneously. For instance, Urbonas et al. [30] have utilized a two-stage detector known as the Faster R-CNN network with an AlexNet backbone to achieve an 80% detection accuracy for wood veneer defects. In 2021, Pan et al. [31] proposed a modification to the Faster R-CNN network based on the Gaussian Proposal Network (GPN) to allow for elliptic RoI to better suit the detection of knot defects. Mask R-CNN is also a variant of the Faster R-CNN network that allows fine-grained segmentation in addition to normal detection. Both Li et al. [32] and Shi et al. [33] have used Mask R-CNN networks in the development of their wood defects detection models. In contrast to two-shot detectors, single-shot detectors perform defect detection in a single pass, which makes them faster than their counterpart. Two common single-shot detectors are the Single-shot Multibox Detector (SSD) and the You Only Look Once (YOLO) family of networks. For instance, Ding et al. [34] have utilized the SSD network with a DenseNet backbone to perform detection of live knot, dead knot and checking defects. Tu et al. [14] have also improved the YOLOv3 network by incorporating Complete Intersection over Union (CIoU) metric and Gaussian modelling for localization uncertainty to achieve accurate and real-time wood defects detection. Other YOLO variants such as YOLOv5 [35] and YOLOv5s-BiFPN [36] have also been experimented to achieve highly accurate detection of wood surface defects.

Despite the many research done on wood defects detection using CNN-based networks, most of them focus on building models with ever increasing complexity to achieve highly accurate detections. These computationally intensive models typically require a well-invested machine with a GPU to perform its task. However, the deployment of such machines for the visual inspection of wood defects is very costly which can create a high barrier to entry for manufacturers especially in developing nations. Besides that, more computations implies higher energy consumption which may result in higher running cost for the manufacturers. Improving the efficiency of CNN detectors and supporting their use on embedded devices can also equalize the access and prevent their exclusive use by high-capital businesses [37]. This circumstance motivates us to explore the possibility of highly efficient models for wood defects detection that is fast, accurate and lightweight. During the model's development, several optimizations and modifications were performed to counteract the accuracy trade-off of lightweight models. After obtaining the optimized model, aggressive compression is performed to drastically reduce the model's number of parameters while maintaining its detection capability. This results in a highly compact wood defects detection model that has minimal computational footprint. The compressed model is then tested on an embedded device without external hardware accelerators to achieve an accurate and near real-time performance for wood defects detection.

To summarize, the main contributions of this paper are as follows:

1) Proposed a lightweight and optimized object detection model for wood defects detection based on the YOLOv4-Tiny architecture. The model is greatly enhanced through hyperparameter search, data augmentation and architectural modifications. Efficient Channel Attention (ECA) modules were added to the model to enhance the detection performance with minimal impact on the model size.
2) Performed aggressive channel pruning iteratively based on the L1 norm strategy to drastically shrink the model size while maintaining high accuracy for high speed inference on embedded devices without external hardware accelerators.

This article is structured as follows: Section II reviews the related literature and compares them with the current research. Section III describes the methods employed whereas Section IV explains the dataset, experiments and metrics used. Lastly, Section V and VI give the discussions and conclusions respectively.

II. RELATED WORK

Despite its importance, the subject of improving model efficiency for detecting wood defects is still relatively underexplored. The general method used by most researchers is to replace part of their CNN model with lightweight alternatives. This will result in a decrease in model complexity and an increase in inference speed. For instance, Wang et al. [38] have replaced the residual block in the YOLOv3 algorithm with a Ghost module structure to reduce model size by 38% while improving inference speed by 11 FPS. Zhao et al. [39] also utilized the Ghost module while replacing regular convolution with depthwise convolution to reduce model size of YOLOv5s by 64%. Architecture modifications can be used to reduce model size significantly, but the reduction is only fixed based on which components are being replaced.

Another method to improve wood inspection efficiency is proposed by Shi et al. [33] in 2020. They introduced an auxiliary component known as a glance network to filter out non-defective wood from their detector network. This increases the overall inspection speed for a batch of wood samples but did not introduce any improvements to the inference speed of the detector.

This research proposes a configurable approach to model size reduction for wood defects detection. Rather than replacing existing architectural components with lighter ones, the entire model is trimmed uniformly based on a reduction ratio. This process can then be repeated until a desired level of lightening has been achieved. With this method, the model is compressed so that it can perform inference directly on an embedded processor, whereas the aforementioned methods were all executed on a GPU.

III. PROPOSED DETECTION METHOD

The wood defects detection model proposed in this paper is based on the YOLOv4-Tiny architecture. The training hyperparameters are tuned so that optimal training conditions can be achieved. Furthermore, several data augmentation strategies are employed to allow the model to generalize better and achieve higher accuracy. Subsequently, attention modules are added to the features of the model so that the relevant features for detection gets enhanced while insignificant features are diminished. Lastly, the model is compressed by cutting the model size via iterative structured pruning. The model is retrained after pruning to recover its lost accuracy. This allows us to develop a model that can perform inference on embedded processors without any external hardware accelerators like GPUs, Tensor Processing Units (TPUs), or Vision Processing Units (VPUs).

A. YOLOv4-TINY ARCHITECTURE AND PRINCIPLES

The YOLOv4-Tiny model is a simplified and compressed version of the YOLOv4 object detection model proposed by Bochkovskiy et al. [40] in 2020. It drastically reduces the total number of convolutional layers in YOLOv4 from 110 layers to a total of only 21 layers. YOLOv4-Tiny model is suitable for object detection applications where devices do not have the luxury of large computational power, but still require accurate and high-speed detection. The YOLOv4-Tiny architecture, despite being a heavily reduced version of YOLOv4, still retains several features of the former, such as having a backbone network made of residual [41] and cross-stage partial (CSP) [42] modules; and a multi-scale feature fusion neck [43] before each of its prediction head. The architecture of YOLOv4-Tiny is shown in Fig. 1.

YOLOv4-Tiny performs detection at two different scales simultaneously. The detection is performed by two 3×3 convolutional layers right before the model's output. YOLOv4-Tiny uses the anchor box mechanism [44] for detection, where each bounding box is predicted relative to a predefined set of width-height dimensions. In YOLOv4-Tiny, each prediction head is assigned A number of unique anchor box dimensions as targets for regression. To perform prediction, YOLOv4-Tiny divides the image into a grid of $M \times M$ cells, and each cell will output A number of anchor predictions. Each prediction is represented by 4 coordinate values (x_c, y_c, w, h), 1 object-ness score (o) and N number of probability scores corresponding to each object class in the dataset. The number of cells per grid dimension (M) depends on the scale of the prediction head, and given that the input image size is $W \times W$, the value of M for the detection head i can be calculated using the formula $M_i = W/(8 \times 2^i)$, where $i \in \{1, 2\}$. With two detection heads, YOLOv4-Tiny effectively outputs $(M_1 \times M_1 + M_2 \times M_2) \times A \times (4 + 1 + N)$ number of predicted values for each input image.

After obtaining the predictions, a confidence threshold value is set to remove predictions with confidence score

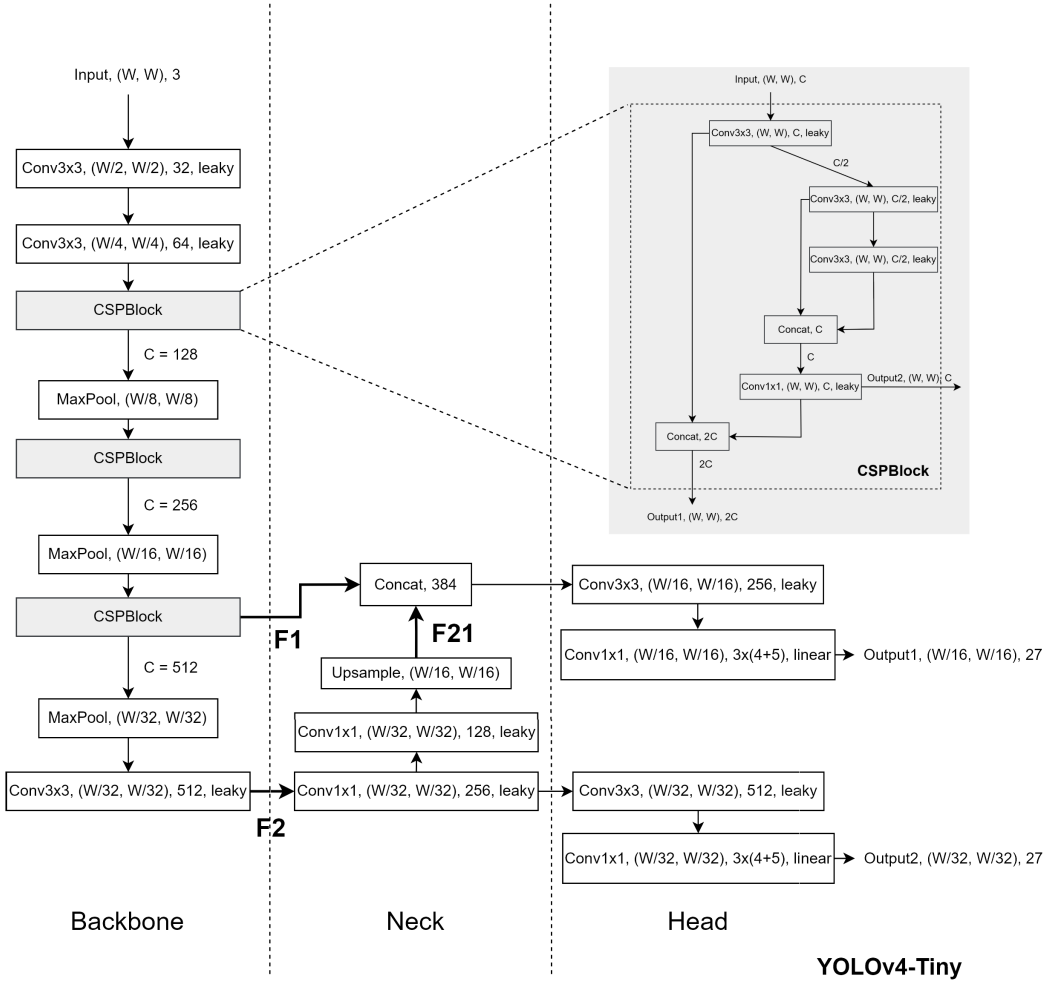

FIGURE 1. YOLOv4-Tiny Architecture Diagram. Attention modules will be added at locations *F1*, *F2*, and *F21* as discussed in Section III-D.

lower than the threshold value. Following that, a standard non-maximum suppression (NMS) algorithm is then performed to remove overlapping predictions and the remaining bounding boxes are treated as the valid outputs of the network. Fig. 2 shows an illustration of the detection process for YOLOv4-Tiny.

YOLOv4-Tiny optimizes its weights through an end-to-end learning process using a single multi-part loss function. The loss function consists of three parts, namely the localization loss (L_{loc}), the object-ness loss (L_{obj}), and the classification loss (L_{cls}). The CIoU [45] metric as shown in (1) is used by YOLOv4-Tiny for the localization loss as it considers both bounding box distance as well as the bounding box aspect ratio in addition to the degree of overlap between the bounding boxes. The equations for v and α are shown in (2) and (3) respectively. The localization loss is then defined as $L_{\text{loc}} = 1 - \text{CIoU}$.

$$\text{CIoU} = \text{IoU} - \frac{\|\boldsymbol{b}_{\text{gt}} - \boldsymbol{b}_{\text{pred}}\|_2^2}{c^2} - \alpha v \tag{1}$$

$$v = \frac{4}{\pi^2}\left(\arctan\frac{w_{\text{gt}}}{h_{\text{gt}}} - \arctan\frac{w_{\text{pred}}}{h_{\text{pred}}}\right)^2 \tag{2}$$

$$\alpha = \frac{v}{1 - \text{IoU} + v} \tag{3}$$

L_{obj} of YOLOv4-Tiny is used to optimize the object-ness score for each prediction, where the score resembles how likely the prediction is to contain an object. Assuming that $\hat{p}$ represents the predicted value while p represents the ground truth label, both the L_{obj} and L_{cls} of YOLOv4-Tiny uses the binary cross-entropy loss function for optimization as shown in (4). The final multi-part loss function for YOLOv4-Tiny is then defined as a linear combination of the three losses as shown in (5), where λ_{loc}, λ_{obj} and λ_{cls} are scalar values used to balance the contributions of each loss to the whole.

$$\text{BCELoss} = \begin{cases} -\log\hat{p} & \text{if } p = 1 \\ -\log 1 - \hat{p} & \text{if } p = 0 \end{cases} \tag{4}$$

$$\text{Loss} = \lambda_{\text{loc}}L_{\text{loc}} + \lambda_{\text{obj}}L_{\text{obj}} + \lambda_{\text{cls}}L_{\text{cls}} \tag{5}$$

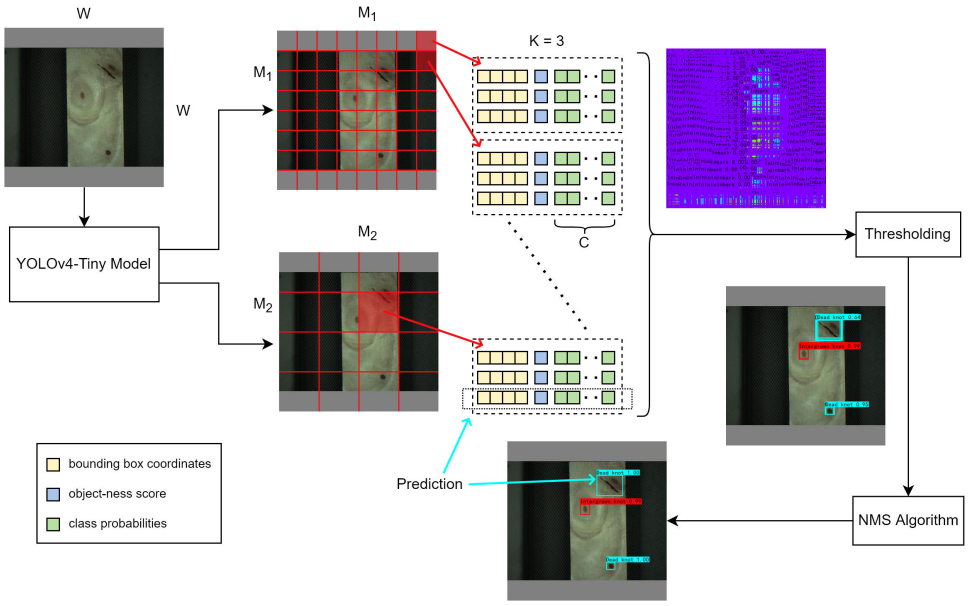

FIGURE 2. YOLOv4-Tiny Detection Process.

B. HYPERPARAMETER OPTIMIZATION

Other than model selection, there exists other variables that can affect the performance of deep neural networks. Primarily, the training criteria and the quality of the training data both play critical roles in determining the performance of the trained model. Currently, the most widely used optimization methods for deep architectures still depend on gradient descent via back-propagation. This method involves several hyperparameters during training and researchers typically rely on best practices [46] that are validated through numerous experimentation. Despite being a decent guide, best practices do not guarantee optimal training as different models and datasets require different sets of training hyperparameters to achieve its fullest potential. Thus, performing a thorough search through the training hyperparameter space is crucial before further improvements are made.

Our hyperparameter search space include the initial learning rate, the training batch size, the optimizer type and the learning rate scheduler type. The initial learning rate is one of the most important training hyperparameters to tune. An initial learning rate that is too large will cause model training to diverge. Inversely, an initial learning rate that is too small will take a long time for training to converge. Given a reasonable number of training epochs, a suitable initial learning rate should thus be selected to train the model optimally. In addition, the training batch size can also have a considerable effect on training performance. A smaller training batch size can allow more updates per epoch and can also act as a form of regularization for the model. However, a training batch size that is too small might slow down computation while also reducing the effectiveness of Batch Normalization (BN) layers [47] for models that utilize them such as YOLOv4-Tiny.

During model training, an optimizer is selected to perform gradient descent and update the model weights. The typical optimization algorithm is Stochastic Gradient Descent (SGD) which uses a single learning rate throughout the training procedure to update all model parameters. SGD with momentum (SGDm) [48] is later proposed to reduce oscillations in each SGD step and speed up convergence. Recently, adaptive optimizers were proposed as alternatives to SGD and its variants. In adaptive optimization, the learning rate for each parameter is different and is changed each iteration based on previous updates. The most well-known adaptive optimizer currently is the Adam [49] optimizer which combines SGDm and the RMSProp [50] algorithm to reduce noisy gradient updates and speed up convergence. Nonetheless, there is still debate among researchers over the effectiveness of adaptive optimizers such as Adam over traditional non-adaptive optimizers like SGDm [51], [52], [53]. Thus, to guarantee optimal convergence of our model, both the SGDm and Adam optimizers are included into the hyperparameter search space.

Lastly, the learning rate scheduler is also a key element when training deep architectures. Many well-known deep neural networks [54], [55], [56] include learning rate schedulers during their training process to achieve SOTA performances. A learning rate scheduler works by reducing the learning rate as training progresses, either continuously or at specific intervals. This has the benefit of having fast convergence during early stages of training due to a large initial learning rate and stabilized convergence due to a gradually decaying learning rate [57]. Two common learning rate schedulers are selected for our hyperparameter search.

The first scheduler is the step decay scheduler as shown in (6) and (7) where t represents the current epoch, $\eta_{\max}$ represents the initial learning rate and T represents the total number of training epochs. The step decay scheduler repeatedly decreases the learning rate (η) by a factor of α after each interval (E) until the minimum learning rate ($\eta_{\min}$) is achieved. The second scheduler chosen is the cosine annealing scheduler as proposed in [58]. Cosine annealing as shown in (8) diminishes the learning rate after each epoch unlike step decay and thus provide a more gradual decay for the learning rate.

$$\eta_t = \alpha^{\lfloor t/E \rfloor} \eta_{\max} \tag{6}$$

$$\alpha = \left(\frac{\eta_{\min}}{\eta_{\max}}\right)^{1/\left(\frac{T}{E}-1\right)} \tag{7}$$

$$\eta_t = \eta_{\min} + \frac{1}{2}\left(\eta_{\max} - \eta_{\min}\right)\left(1 + \cos\left(\frac{t}{T}\pi\right)\right) \tag{8}$$

C. DATA AUGMENTATIONS

Besides the training hyperparameters, the quality of the training dataset is also important to train better deep learning models. Deep neural networks usually tend to have a large number of trainable parameters. This makes them highly susceptible to overfitting, especially when training data is scarce or limited. To tackle this issue, various regularization techniques such as Dropout [59] and Weight Decay [60] have been proposed which have been shown to improve the generalization capability of the model. Another type of regularization method that directly acts on the training dataset is known as data augmentation. To perform data augmentation, various transformations are applied on the input image to artificially inflate the training dataset. The selection of transformations suitable for a specific problem is paramount as these transformations should preserve the semantic meaning of the labels.

Four data augmentation techniques have been considered in this research. This include random flipping, scale jittering, color jittering and mosaic augmentation [40]. Random flipping flips the input image in the horizontal and vertical direction with each action having a 50% chance of occurring. Scale jittering applies a random scale transformation on the image and later modifies the aspect ratio of the image. The scaling factor for the image ranges from $[0.25, 2]$ while the aspect ratio of the image is modified by a factor in the range of $[0.7/1.3, 1.3/0.7]$. Color jittering is performed based on the Hue-Saturation-Value (HSV) color space of the image. The degree of jittering for the hue, saturation and value channels of the image are within $\pm 10\%$, $\pm 70\%$ and $\pm 40\%$ respectively.

As opposed to the former transformative augmentations, Mosaic is a synthetic data augmentation method that improves the model generalization capability by synthesizing images for training. The mosaic algorithm works by splicing 4 different images into a single training image and combines the labels for each of these images during training. The 4 images are combined into a 2×2 grid with unequal cell areas that eventually fit into a square image. Assuming that (x_s, y_s) represents the bottom-left corner of the top-left grid cell and the size of the square image is $W \times W$, the range of values for x_s and y_s is set to fall within $[0.3W, 0.7W]$. With mosaic data augmentation, each object can be presented in a different context [40] during training and thus help improve generalization. An illustration of mosaic data augmentation is shown in Fig. 3.

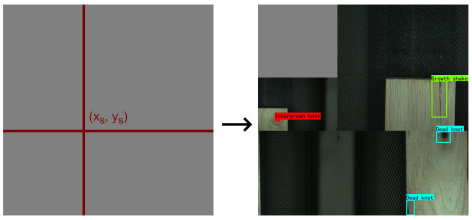

FIGURE 3. An Illustration of Mosaic Data Augmentation.

D. ATTENTION MODULES

Besides obtaining highly discriminative features, it is also important for the model to focus on the salient features while suppressing the unimportant ones. This can be achieved via an attention mechanism. An attention mechanism performs a feature selection process by applying a learn-able weight to each feature of the model based on their importance [61]. Attention can be applied to a convolutional neural network either via channel-wise attention or spatial attention. Channel-wise attention allows the model to discriminate between semantic features while spatial attention allows the model to pay attention to regions that matter.

In our model, attention modules are added at three different locations denoted by the symbols **F1**, **F2**, and **F21** in Fig. 1. Locations **F1** and **F2** are the two sets of features extracted from the backbone network of the model for the detection of objects at two different scales. Applying attention at the two locations can help the model emphasize on important features extracted from the backbone network while reducing unwanted noise. In addition to **F1** and **F2**, an attention module is also added at location **F21**. The features at **F21** are features with high semantic value that are being accommodated to a smaller scale to assist in the detection of small objects. Attention module is added here to pick the relevant features before feature fusion for small scale detection.

Three attention modules have been experimented and the best attention module is selected for the final model. The first attention module considered is the Squeeze-and-Excitation (SE) attention module proposed by Hu et al. [62] in 2018. The SE module is a channel-wise attention module that utilizes a squeeze operation followed by an excitation operation to perform attention. In the squeeze operation, global average pooling (GAP) is applied to each channel output of a convolutional layer to condense the channel's global information into a single descriptor. Later on, this one-dimensional vector

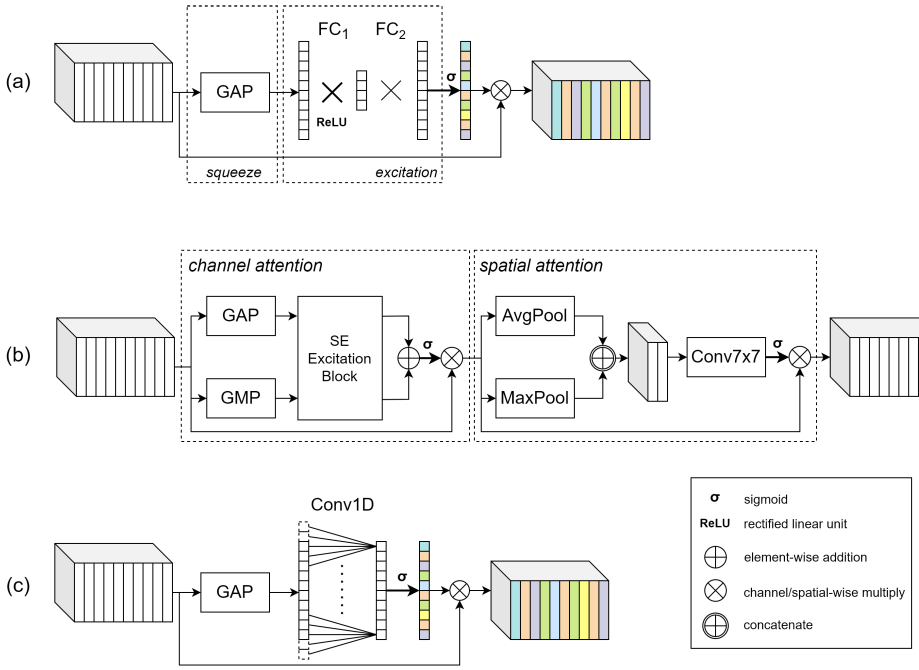

FIGURE 4. The various attention modules tested in our research. (a) SE module (b) CBAM module (c) ECA module.

is "excited" by passing through two fully connected (FC) layers with non-linearity that learns the weights required to scale the channels. This scaling effectively performs feature selection for attention. In the excitation phase, the first FC layer performs a dimensionality reduction with a reduction ratio of (r) while the second FC layer recovers the dimension of the output for scaling the channels. This reduction is performed to limit the number of learn-able parameters while aiding generalization. The architecture for SE is shown in Fig. 4(a).

The second attention module considered is the Convolutional Block Attention Module (CBAM) [63]. It is a hybrid attention module that combines both channel and spatial attentions. The attentions are performed in a sequential manner where channel attention is performed before spatial attention. CBAM uses a mechanism similar to SE for channel attention. The difference is that CBAM performs two parallel SE channel attention with one using GAP for the squeeze operation while the other uses a global maximum pooling (GMP) operation. Both branches share the same weights for the FC layers. The final weights for channel attention is then obtained by adding the output of the two branches followed by a sigmoid activation function. For spatial attention, CBAM passes the channel-wise attended features through two separate branches. The first branch performs average-pooling in the channel dimension to obtain a descriptor for each spatial point on the feature map. The second branch performs a similar operation but with max-pooling instead of average-pooling. The weights for spatial attention is then obtained by concatenating the output of the two branches and passing it into a 7×7 convolutional layer followed by a sigmoid activation function. The architecture for CBAM is shown in Fig. 4(b).

The third attention module considered is the Efficient Channel Attention (ECA) module [64]. The ECA module is an improvement over the SE module without the dimensionality reduction. ECA retains the GAP used in the squeeze phase of SE while replacing the two FC layers in the excitation phase wtih a single 1D convolutional layer. The 1D convolutional layer is still able to capture cross-channel interactions between channels like the two FC layers but without the indirect correspondence introduced by dimensionality reduction. Even though 1D convolution trade offs full cross-channel interactions with local interactions to lower the number of computations, the authors of ECA showed that both can achieve similar results. In ECA, the kernel size of the 1D convolutional layer is determined adaptively based on the number of channels. Let C be the number of channels and $|t|_{\text{odd}}$ represents the nearest odd number to t, the kernel size (k) for ECA can be determined using (9). The architecture for ECA is shown in Fig. 4(c).

$$k = \left| \frac{\log_2 C + 1}{2} \right|_{\text{odd}} \tag{9}$$

E. AGGRESSIVE STRUCTURED MODEL PRUNING

Despite the complexity of vision-based problems, deep learning models are still typically over-parameterized. Therefore, there exists sufficient capacity for the model to be compressed so as to save memory and computation time. One such

method is known as unstructured pruning whereby individual connections from the network are removed based on a scoring criteria. Han et al. [65] performed unstructured pruning based on the magnitude of the weights and was able to compress the model size of AlexNet by 9× without incurring any accuracy loss. However in practice, despite the high sparsity of the resulting model, this method does not accelerate inference significantly as it requires specialized hardware and software to do so. On the other hand, structured pruning removes model parameters in groups such that model inference can be sped-up using off-the-shelf libraries. Channel pruning [66] is an instance of structured pruning where entire channels are removed from a convolutional layer, which results in a thinner and more efficient model. This is also the strategy employed for pruning our model as it is relatively simple to implement and can result in significant gains in model compression.

In this research, model pruning is performed in an iterative fashion. Each iteration is split into a model pruning stage and a retraining stage. The first iteration of model pruning starts after standard training of the model. Channel pruning is then performed on all convolutional layers of the model including layers involving residual connections and channel splitting. Channel pruning is done by removing the filters of the convolutional layer corresponding to the pruned channels at the output. Whenever a filter is truncated, the filters in the following layers will also be pruned to accommodate the change in input channel size. In terms of pruning criteria, the L1 norm is used to determine the importance of the channel's filter to the specific convolutional layer. Assuming that the channel filter of size $K \times K \times C$ is denoted by $\boldsymbol{W}$ where w_{ijk} represents its individual weights, the L1 norm criteria for $\boldsymbol{W}$ is then defined in (10). For each layer, the L1 norm of its channel filters are ranked and the lowest 30% will be removed. After pruning, the model is retrained from its remaining weights to recover the lost accuracy due to pruning. Model pruning and retraining is performed for a total of 7 times and the best model is selected based on the trade-off between model size and accuracy.

$$\|\boldsymbol{W}\|_{\mathrm{L1}} = \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \sum_{k=0}^{C-1} |w_{ijk}| \tag{10}$$

IV. EXPERIMENTS

A. WOOD DEFECT DATASET

The dataset used in this research is the rubber wood defects dataset provided by Tu et al. [14] in their GitHub repository. This dataset contains 1545 images of sawn rubber wood. Among them, 1112 images are allocated for training while the remaining 278 and 155 images are set aside for validation and test. The dataset contains four types of wood defects, namely Intergrown Knot (IK), Dead Knot (DK), Growth Shake (GS) and Inbark (IB) defects. Knot defects are circular grain structures found on the surface of lumber. IK defects are knot structures closely integrated with its surrounding wood while DK defects are loose knots with a darker tone. The GS defect presents itself as cracks or splits in the wood while IB defects are embedded bark in the xylem which appear as dark patches on the wood surface. An example for each of the wood defects is shown in Fig. 5.

Within the dataset, the most common defects are the IK and DK defects while the least occurring defect is the IB defect. The distribution of the defects for the training, validation and test dataset is shown in Fig. 6. From Fig. 6, it can be seen that the training, validation and test dataset share a similar distribution. Thus, the validation and test dataset can provide a good benchmark to verify the performance of the trained models.

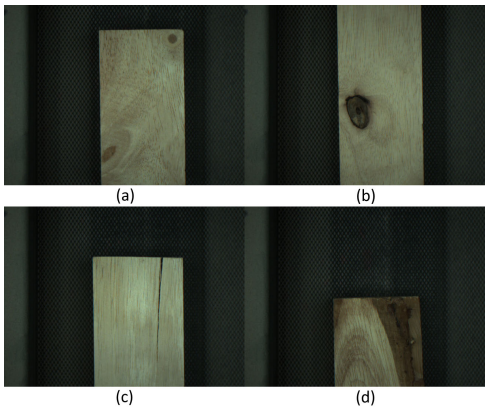

FIGURE 5. The four types of wood defects in the dataset. (a) Intergrown Knot (b) Dead Knot (c) Growth Shake (d) Inbark.

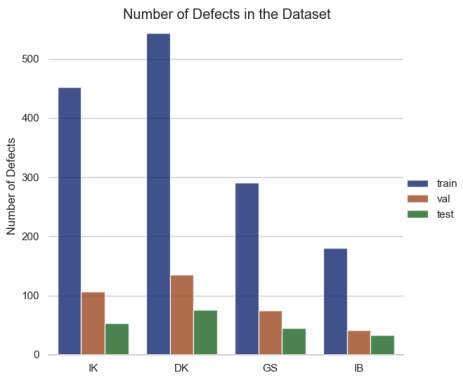

FIGURE 6. The number of wood defect occurrences in the dataset.

B. EXPERIMENTAL SETUP

The experiments were conducted in the following order. Firstly, the optimal training hyperparameters were determined via a grid search of the initial learning rate, batch size, optimizer type and scheduler type as discussed in Section III-B. After obtaining the optimal training hyperparameters, all subsequent model training used the same set of hyperparameters. The second step involves experimenting with the various data augmentation strategies outlined in

Section III-C. To find the best augmentations for our specific problem, each strategy has been evaluated separately by excluding it and examining its effect on the overall accuracy. Later on, different attention modules (see Section III-D) were added to the model to further improve its detection accuracy. The attention module that gives the best performance was selected and the final model is compared with other SOTA models. Lastly, the model was aggressively pruned for the purpose of achieving real-time inference on embedded processors.

During training, the images were pre-processed using the letterbox method to ensure that the input image size is square. Given a target image size (W), the letterbox method resizes the training image so that its greater side equals to W. The RGB channels of the remaining space is then filled with the value of 128 to mimic the color of gray. For images that are subjected to scale jittering as described in Section III-C, letterbox is applied after the augmentation to ensure that the final input image is square. For each experiment, the model was trained for a total of 500 epochs and the model with the lowest validation loss was selected. During evaluation, the IoU threshold for the NMS algorithm is set to 0.5. All other hyperparameters are determined by the hyperparameter search.

To evaluate the performance of the model on embedded processors, a bare Raspberry Pi 4 was used as the test bench. The model was tested on the ARM Cortex-A72 Central Processing Unit (CPU) of the Raspberry Pi 4 using Pytorch's CPU inference mode. All other experiments were conducted on a machine running Ubuntu 20.04.3 LTS with an AMD Ryzen 5 3600 CPU processor and a 12 GB NVIDIA GeForce RTX 3060 GPU. The deep learning framework used in our research is Pytorch 1.8.1 with CUDA 11.1 and the programming language used is Python 3.7.10.

C. EVALUATION METRICS

The performance of the models were evaluated based on their accuracy, speed and size. For accuracy, several universally accepted metrics for object detection were used including precision, recall, F1 score and the average precision (AP) score. Precision and recall of the model are calculated from the total number of true positives (TP), false positives (FP) and false negatives (FN) of the model. TP represents the total number of correct detections while FP represents the total number of false predictions generated by the model. FN is the total number of ground truth objects labelled in the dataset but not detected by the model. It also represents the total number of missed detections of the model. Hence, the precision and recall of the model can be defined as shown in (11) and (12). Subsequently, the F1 score can then be calculated from the precision and recall using (13). The precision, recall and F1 score shown will be calculated at a confidence threshold (θ_{conf}) of 0.5.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{11}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{12}$$

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{13}$$

Even though precision, recall and F1 score are good metrics used for performance comparison, the most well known metric to evaluate object detection models is AP. This is because the precision, recall and F1 score all depend on the value of θ_{conf} set during inference. To better evaluate the performance of object detection models, the metric should represent the accuracy of the model regardless of this threshold. AP evaluates the accuracy across confidence thresholds by integrating the area under the curve (AUC) for the precision-recall (PR) curve of the model. The PR curve is a parametric function of the θ_{conf} where the output is the precision and recall values of the model. The AP metric is independently calculated for each class in the dataset. The mean average precision (mAP) score as in (14) is then defined as the arithmetic mean of the AP metric across all classes and is typically used as the de facto performance metric for all object detection models.

$$\text{mAP} = \frac{1}{N} \sum_{i \in \text{classes}}^{N} \text{AP}_i \tag{14}$$

For inference speed, the frames per second (FPS) metric is used. FPS is calculated by averaging the inference time of the model over 100 passes. The size of the model is measured using the number of parameters as well as the number of floating point operations counted in the billions (GFLOPs). During the calculation of GFLOPs, the addition and multiplication operation in a convolution operation is considered as two separate operations instead of one.

V. EXPERIMENTAL RESULTS AND DISCUSSIONS

A. RESULTS FROM HYPERPARAMETER SEARCH

Table 1 shows the mAP score for various combinations of initial learning rate, batch size, optimizer type and scheduler type for the YOLOv4-Tiny model. The results of the grid search suggest that the best combination of training hyperparameters is a small initial learning rate of 0.0005 with a batch size of 32 while using the Adam optimizer with a cosine annealing scheduler. From the table, several observations can be made. For the SGDm optimizer, mAP generally increases as learning rate increases. The inverse can be observed for the Adam optimizer where mAP decreases as learning rate increases. Besides that, the mAP score for SGDm trends upwards as batch size decreases. However, the trend of mAP versus batch size for Adam is not that apparent as it depends on other factors like the learning rate and scheduler type. In terms of scheduler type, it can be seen that the step scheduler performs the worst as compared to the cosine annealing scheduler or when no scheduler is used. As a whole, the Adam optimizer performs better than SGDm in most cases except at very high learning rates.

TABLE 1. mAP for Different Sets of Training Hyperparameters for YOLOv4-Tiny.

		$\eta_{max} = 0.0005$			$\eta_{max} = 0.001$			$\eta_{max} = 0.005$		
		8	16	32	8	16	32	8	16	32
none	SGDm	0.6693	0.5440	0.4884	0.7500	0.6281	0.5727	0.8284	0.7964	0.7331
	Adam	0.8309	0.8222	0.8036	0.8302	0.8265	0.8162	0.7446	0.7711	0.7392
step	SGDm	0.4998	0.3659	0.1805	0.6517	0.4710	0.3393	0.7898	0.7548	0.6440
	Adam	0.8122	0.8143	0.8120	0.8301	0.8208	0.8108	0.5138	0.6530	0.7724
cosine	SGDm	0.5877	0.4848	0.3732	0.6763	0.5903	0.5207	0.7996	0.7688	0.6515
	Adam	0.8198	0.8258	**0.8320**	0.8201	0.805	0.8019	0.7455	0.7377	0.7987

B. DATA AUGMENTATION ABLATION STUDY

After obtaining the optimal set of training hyperparameters, an ablation study is performed for different types of data augmentation strategies. The effect of each data augmentation method on the overall performance of the model is shown in Table 2. In Table 2, the four data augmentation strategies were individually omitted to observe their influence on detection accuracy.

From the table, it can be observed that the mosaic data augmentation does not introduce any improvements to the mAP score. This is likely due to the synthetic images being too artificial and unlikely to happen in any real scenario. In fact, the noise introduced by the synthetic images actually caused a slight decrease in the overall mAP score. Therefore, mosaic data augmentation is excluded from the set of optimal data augmentation strategies. From there on, it can be seen that excluding any of the three transformative augmentations will result in a decrease in mAP score. This means that they all play a role in improving the accuracy of the model. Scale jittering has the highest impact on the detection accuracy where it caused a drop of 6.89% in terms of mAP score when omitted. This finding shows that increasing the scale variability of defects in the dataset is a key factor in improving the generalization capability of the model. Among the transformative augmentations, color jittering causes the least improvement in terms of mAP score. This is expected as the defect images were collected in a controlled environment and thus do not benefit significantly from having color and brightness invariability. In essence, the optimal set of data augmentations is to enable scale and color jittering together with random flipping without mosaic data augmentations.

TABLE 2. Ablation Study for Various Data Augmentation Methods.

scale jitter	color jitter	flip	mosaic	mAP
✓	✓	✓	✓	0.9303
✓	✓	✓		**0.9307**
✓	✓			0.9232
✓		✓		0.9288
	✓	✓		0.8618

C. OPTIMAL ATTENTION MODULE

Table 3 shows the detection accuracy, model size and inference speed of YOLOv4-Tiny enhanced by the SE, CBAM and ECA attention modules. From the table, it can be seen that the CBAM module performs the best for knot detection such as IK and DK. For the GS defect, ECA performs the best at a mAP score of 90.65%. All attention modules slightly reduce AP score for the IB defect as compared to the default YOLOv4-Tiny architecture. Nonetheless, in terms of overall accuracy, it is apparent that the ECA attention module performs the best with a mAP score of 94.53%. The result suggests that having high quality channel attention without dimensionality reduction works better than having both channel and spatial attention. Besides that, the increase in terms of model size due to the addition of ECA modules is barely noticeable. ECA is also the fastest among all other attention modules tested. Therefore, it is the obvious choice to select ECA for the enhancement of YOLOv4-Tiny using attention mechanisms.

D. PERFORMANCE COMPARISON

Fig. 7 shows a comparison of the validation loss curves for each of the improved YOLOv4-Tiny models. The natural logarithm of the validation loss is plotted on the vertical axis to allow a clearer distinction between trends at extremely small loss values. From the figure, it can be observed that the YOLOv4-Tiny model with only optimized hyperparameters (HYP) begins to experience stagnation in its validation loss after approximately 100 epochs of training. However, upon introducing data augmentation (DA) into the training pipeline, the models are able to further decrease their

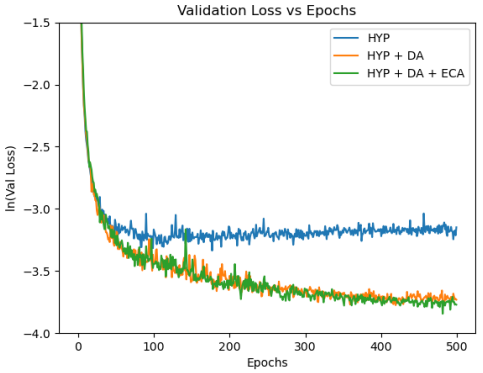

FIGURE 7. Comparison of validation loss curves for the improved YOLOv4-Tiny models. Meaning of symbols: HYP - optimized hyperparameters, DA - optimized data augmentations, ECA - YOLOv4-Tiny with Efficient Channel Attention.

TABLE 3. Performance of YOLOv4-Tiny with Different Attention Modules.

Attention	AP(IK)	AP(DK)	AP(GS)	AP(IB)	mAP	params (M)	GFLOPs	FPS (GPU)
default	0.9483	0.9503	0.8498	**0.9742**	0.9307	**5.881**	**6.826**	**219.06**
SE	0.9562	0.9731	0.8658	0.9338	0.9323	5.924	6.827	214.13
CBAM	**0.9564**	**0.9766**	0.8765	0.9552	0.9412	5.967	6.828	198.15
ECA	0.9385	0.9746	**0.9065**	0.9616	**0.9453**	**5.881**	6.827	215.66

TABLE 4. Comparison of YOLOv4-Tiny-ECA with SOTA Methods.

Model	Image Size	AP(IK)	AP(DK)	AP(GS)	AP(IB)	mAP	Params (M)	GFLOPs	FPS (GPU)
SSD300 [67]	300 × 300	0.8609	0.9222	0.8023	0.9182	0.8759	24.01	61.11	69.58
GC-YOLOv3 [14]	320 × 320	0.8147	0.8117	0.7151	0.7800	0.7804	63.92	19.55	88.07
YOLOv4-Tiny-ECA (ours)	320 × 320	**0.9148**	**0.9740**	**0.8408**	**0.9288**	**0.9146**	**5.881**	**4.040**	**225.6**
YOLOv3 [68]	416 × 416	0.9327	0.9255	0.8005	0.8855	0.8860	61.54	65.54	56.78
YOLOv4 [40]	416 × 416	0.9208	0.9176	0.7905	0.9220	0.8877	63.95	59.78	44.47
YOLOv5s [69]	416 × 416	**0.9734**	0.9607	0.8785	0.9517	0.9411	7.072	6.972	110.58
YOLOX-Tiny [70]	416 × 416	0.9419	0.9401	0.8523	0.9529	0.9218	**5.034**	**6.437**	96.82
Retinanet [71]	416 × 416	0.8898	0.9185	0.8071	0.8258	0.8603	36.39	69.58	50.71
Faster-RCNN [44]	416 × 416	0.8428	0.8783	0.8516	0.9473	0.8800	136.8	252.8	11.47
YOLOv4-Tiny-ECA (ours)	416 × 416	0.9385	**0.9746**	**0.9065**	**0.9616**	**0.9453**	5.881	6.827	**215.7**
Efficientdet-d0 [72]	512 × 512	0.9465	0.9595	0.7180	0.7018	0.8314	**3.830**	**4.607**	35.53
GC-YOLOv3 [14]	512 × 512	0.9192	0.8846	0.6909	0.8384	0.8333	63.92	50.06	58.27
YOLOv4-Tiny-ECA (ours)	512 × 512	**0.9512**	**0.9816**	**0.8774**	**0.9644**	**0.9437**	5.881	10.34	**155.7**

validation losses significantly. This suggests that data augmentation plays an important role in improving the generalization capability of the model and avoid overfitting. Furthermore, the inclusion of ECA modules into the architecture of YOLOv4-Tiny can improve the convergence of validation loss and subsequently improve model performance. This is supported by the observation that the slope of the validation loss curve for the model with ECA is slightly steeper than that of the model without it.

In the following sections, the ECA enhanced model with optimal training hyperparameters and data augmentations (HYP + DA + ECA) will be denoted as YOLOv4-Tiny-ECA. Table 4 shows a comprehensive comparison of YOLOv4-Tiny-ECA with other SOTA models for object detection. As models such as SSD300 and Efficientdet-d0 require fixed size inputs, YOLOv4-Tiny-ECA has been retrained at their respective image sizes to provide fair comparison. To compare with SSD300, the model was trained at an image size of 320 × 320 instead of 300 × 300 because YOLOv4-Tiny-ECA only accepts input image sizes that are multiples of 32. From the table, it is evident that YOLOv4-Tiny-ECA outperforms all SOTA methods in terms of mAP across all image sizes. Furthermore, with the exception of Efficientdet-d0 and YOLOX-Tiny, YOLOv4-Tiny-ECA has the lowest model size as compared to other SOTA models. This shows that despite having significantly fewer learn-able parameters, YOLOv4-Tiny-ECA is still able to outperform other SOTA models in terms of performance. Most importantly, the model is still able to perform inference at a very high speed despite the accuracy gains. This allows wood defects detection to be performed on low-cost devices in real-time without sacrificing accuracy.

E. MODEL PRUNING

Given the exceptionally high inference speeds of more than 200 FPS on a GPU-enabled machine coupled with a substantial accuracy margin when compared with SOTA methods, there is an opportunity for YOLOv4-Tiny-ECA to be further optimized to realize real-time inference on embedded processors. The method used to achieve this is through model pruning as outlined in Section III-E. Table 5 shows the results of model pruning after 7 cycles for the image sizes of 320 × 320, 416 × 416 and 512 × 512. For each iteration of model pruning, the number of parameters of the model is cut by half. After 7 iterations of model pruning, the reduction to the total number of parameters is around a 100× that of the unpruned model.

Fig. 8 shows the mAP score of the model with respect to the total number of model parameters. From Fig. 8, it can be seen that the model is able to maintain a high mAP score even when the model is heavily pruned. The model accuracy only starts falling drastically when the model is below 10% of its original total number of parameters. Moreover, it can be observed that a larger image size slows down mAP degradation from model pruning more effectively than smaller images. This is expected as larger images contain more fine-grained features to aid detection. Fig. 9 shows the mAP score of the model versus the FPS of the model inferenced on a CPU-only Raspberry Pi 4. The figure shows an approximately linear relationship between the mAP score and the FPS of the model. However, in contrast with the previous figure, mAP score decreases slower for smaller image sizes as FPS increases. This is due to the lower number of GFLOPs involved in the inference of smaller images as compared to the larger ones.

TABLE 5. Results of Model Pruning for Different Image Sizes.

Prune Iters	Image Size	Params (%)	Param (M)	GFLOPs	Model Size (MB)	FPS (RPi4 CPU)	mAP
0 (none)	320 × 320	100.0%	5.8810	4.0400	78.99	3.5074	0.9146
1	320 × 320	49.9%	2.9370	2.0360	50.71	5.2330	0.8955
2	320 × 320	24.7%	1.4510	1.0180	33.58	7.9793	0.8749
3	320 × 320	12.1%	0.7133	0.5106	22.89	11.3245	0.8546
4	320 × 320	6.0%	0.3517	0.2516	15.57	15.3798	0.8508
5	320 × 320	3.0%	0.1750	0.1288	11.26	21.2415	0.8084
6	320 × 320	1.4%	0.0836	0.0603	7.64	29.6214	0.6607
7	320 × 320	0.7%	0.0397	0.0279	5.12	35.7437	0.4759
0 (none)	416 × 416	100.0%	5.8810	6.8270	118.01	2.1662	0.9453
1	416 × 416	49.9%	2.9370	3.4520	77.96	3.3207	0.9162
2	416 × 416	24.7%	1.4520	1.7230	52.94	5.0849	0.9187
3	416 × 416	12.1%	0.7120	0.8590	36.81	7.3066	0.8992
4	416 × 416	6.0%	0.3510	0.4230	25.38	10.2198	0.8734
5	416 × 416	3.0%	0.1740	0.2170	18.57	13.8284	0.8363
6	416 × 416	1.4%	0.0835	0.1025	12.70	18.4558	0.7666
7	416 × 416	0.7%	0.0396	0.0470	8.54	24.0172	0.6247
0 (none)	512 × 512	100.0%	5.8810	10.3410	167.20	1.4566	0.9437
1	512 × 512	49.9%	2.9360	5.2070	112.32	2.3460	0.9327
2	512 × 512	24.6%	1.4490	2.6000	77.33	3.5467	0.9281
3	512 × 512	12.1%	0.7116	1.2980	54.36	5.1252	0.9314
4	512 × 512	6.0%	0.3501	0.6415	37.76	7.0180	0.8720
5	512 × 512	3.0%	0.1742	0.3286	27.79	10.0777	0.8571
6	512 × 512	1.4%	0.0831	0.1529	19.07	13.5261	0.7728
7	512 × 512	0.7%	0.0394	0.0702	12.86	18.1229	0.6503

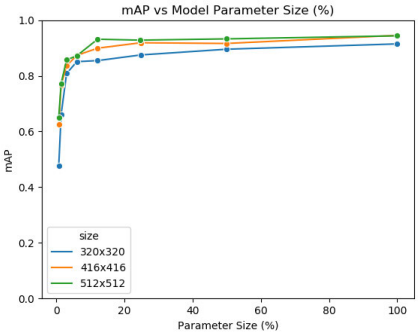

FIGURE 8. mAP versus model parameter size (%) for each pruning iteration.

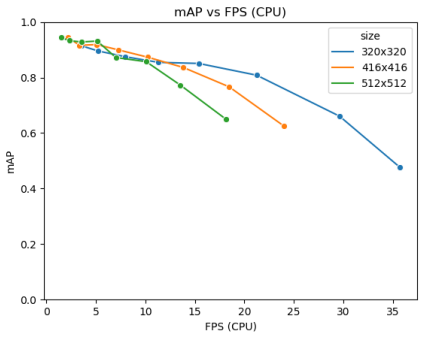

FIGURE 9. mAP versus FPS on a Raspberry Pi 4 for each pruning iteration.

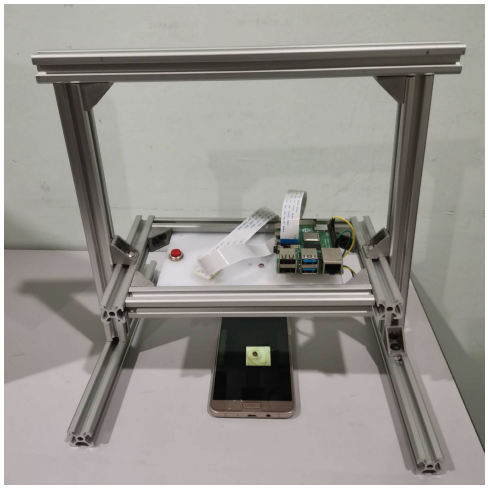

FIGURE 10. The defect detection prototype for embedded processing. It utilizes a Raspberry Pi 4 to execute the pruned YOLOv4-Tiny-ECA model, while a connected Raspberry Pi V2.1 camera module captures input images in real-time.

F. MODEL INFERENCE ON EMBEDDED PROCESSOR

From the results of the model pruning experiments, the optimal number of pruning iterations for the model is 3. At this stage, the model still maintains a high mAP score comparable with other SOTA models. However, the model size has been reduced by a factor of 10 while the inference speed of the model has been increased by a factor of 5. A prototype as shown in Fig. 10 has been developed to test the final model. The prototype has a small form factor which makes it easily portable for demonstration. The prototype uses only a bare Raspberry Pi 4 for inference, which is not connected to any external hardware accelerators. The statistics for the pruned models evaluated using the prototype on the test dataset is given in Table 6. A series of sample outputs is also shown in Fig. 11 to showcase the detection capabilities of the pruned YOLOv4-Tiny-ECA models. From the results, it can be seen that larger images generally improves accuracy

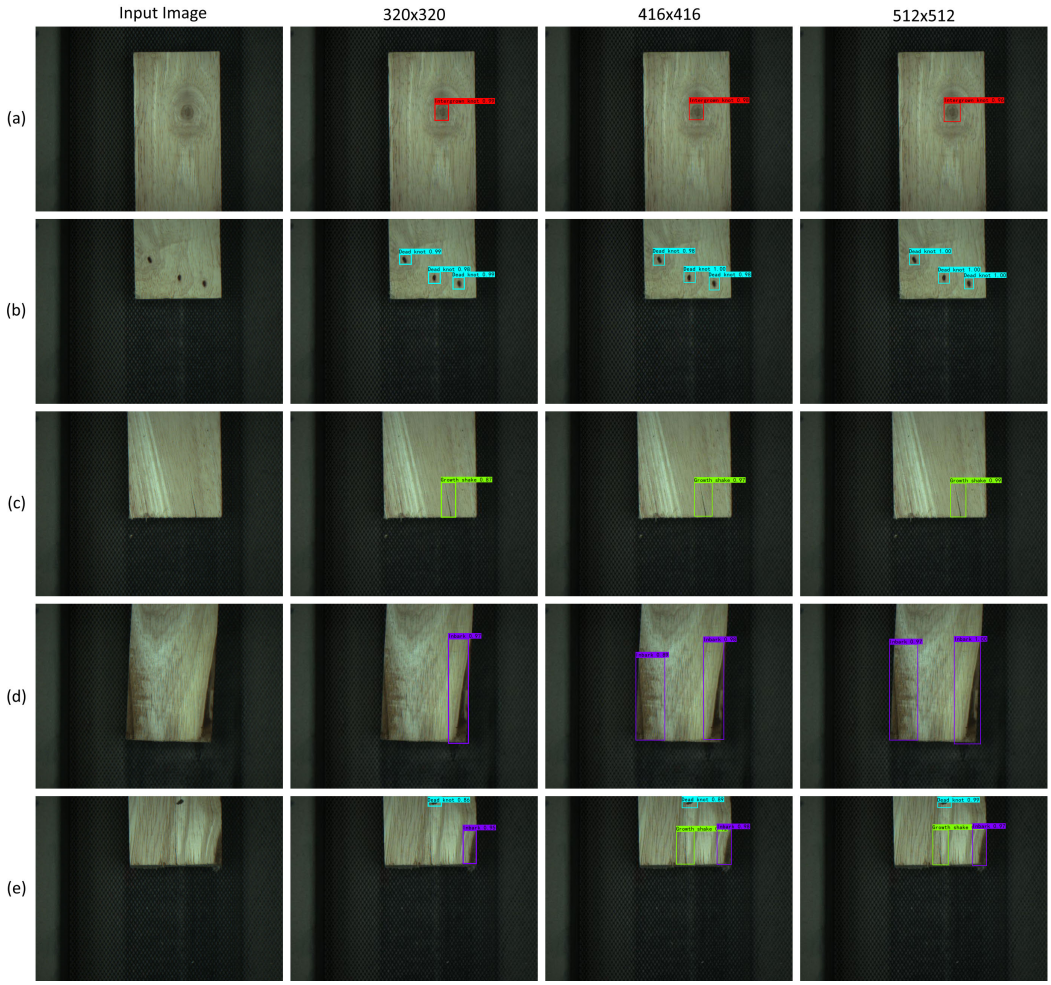

FIGURE 11. The outputs of the pruned models. (a) IK (b) DK (c) GS (d) IB (e) Mixed defects.

TABLE 6. Evaluation of Pruned Models on the Test Dataset.

Metrics	320×320	416×416	512×512
mAP	0.8253	0.8824	0.8649
Precision	0.8517	0.8844	0.8530
Recall	0.7560	0.8004	0.7987
F1	0.8003	0.8397	0.8183
FPS (RPi 4)	11.32	7.307	5.125
Model Size (MB)	22.89	36.81	54.36

with 416 being the optimal image size but at the expense of slower inference speed. Nonetheless, all the pruned models can still perform wood defects detection decently despite their extremely compact architecture. This shows that deep learning based models for wood defects detection can be deployed on cheap CPU-only machines to enable low-cost, real-time and accurate wood quality inspections.

VI. CONCLUSION

In short, an extremely lightweight object detection model has been presented for the detection of wood defects on the surface of sawn lumber. A grid search has been performed to find the optimal training hyperparameters and augmentations have been performed on the training dataset to improve generalization. ECA modules were then added as a means of attention to improve the model's detection capability. The model is ultimately compressed via model pruning to achieve fast and accurate wood defects detection on embedded processors. Future work is directed towards achieving actual real-time performance of more than 30 FPS for embedded processing, which include speeding up post-processing and improving the channel pruning strategy to allow higher compression ratios.

ACKNOWLEDGMENT

The authors would like to extend cordial thanks to the Malaysian Timber Industry Board (MTIB), SIRIM Berhad, Universiti Tunku Abdul Rahman (UTAR), and the Centre for Artificial Intelligence and Computing Applications (CAICA) for their support and valuable feedback to make this research a successful one.

REFERENCES

[1] R. Qayyum, K. Kamal, T. Zafar, and S. Mathavan, "Wood defects classification using GLCM based features and PSO trained neural network," in *Proc. 22nd Int. Conf. Autom. Comput. (ICAC)*, Sep. 2016, pp. 273–277.

[2] C. Clement, T. Lihra, R. Gazo, and R. Beauregard, "Maximizing lumber use: The effect of manufacturing defects on yield, a case study," *Forest Products J.*, vol. 56, no. 1, p. 60, 2006.

[3] G. A. Ruz, P. A. Estévez, and C. A. Perez, "A neurofuzzy color image segmentation method for wood surface defect detection," *For. Prod. J.*, vol. 55, no. 4, pp. 52–58, 2005.

[4] X. YongHua and W. Jin-Cong, "Study on the identification of the wood surface defects based on texture features," *Optik-Int. J. Light Electron Opt.*, vol. 126, no. 19, pp. 2231–2235, Oct. 2015.

[5] P. Prasitmeeboon and H. Yau, "Defect detection of particleboards by visual analysis and machine learning," in *Proc. 5th Int. Conf. Eng., Appl. Sci. Technol. (ICEAST)*, Jul. 2019, pp. 1–4.

[6] D. Riana, S. Rahayu, and M. Hasan, "Comparison of segmentation and identification of swietenia mahagoni wood defects with augmentation images," *Heliyon*, vol. 7, no. 6, Jun. 2021, Art. no. e07417.

[7] Y. X. Zhang, Y. Q. Zhao, Y. Liu, L. Q. Jiang, and Z. W. Chen, "Identification of wood defects based on LBP features," in *Proc. 35th Chin. Control Conf. (CCC)*, Jul. 2016, pp. 4202–4205.

[8] K. Kamal, R. Qayyum, S. Mathavan, and T. Zafar, "Wood defects classification using laws texture energy measures and supervised learning approach," *Adv. Eng. Informat.*, vol. 34, pp. 125–135, Oct. 2017.

[9] Y. Zhang, S. Liu, W. Tu, H. Yu, and C. Li, "Using computer vision and compressed sensing for wood plate surface detection," *Opt. Eng.*, vol. 54, no. 10, Oct. 2015, Art. no. 103102.

[10] S. Li, D. Li, and W. Yuan, "Wood defect classification based on two-dimensional histogram constituted by LBP and local binary differential excitation pattern," *IEEE Access*, vol. 7, pp. 145829–145842, 2019.

[11] M. M. Hittawe, S. M. Muddamsetty, D. Sidibe, and F. Meriaudeau, "Multiple features extraction for timber defects detection and classification using SVM," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2015, pp. 427–431.

[12] U. R. Hashim, S. Z. M. Hashim, and A. K. Muda, "Performance evaluation of multivariate texture descriptor for classification of timber defect," *Optik*, vol. 127, no. 15, pp. 6071–6080, Aug. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0030402616302868

[13] R. Ren, T. Hung, and K. C. Tan, "A generic deep-learning-based approach for automated surface inspection," *IEEE Trans. Cybern.*, vol. 48, no. 3, pp. 929–940, Mar. 2018.

[14] Y. Tu, Z. Ling, S. Guo, and H. Wen, "An accurate and real-time surface defects detection method for sawn lumber," *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–11, 2021.

[15] R. Zhang, L. Xu, Z. Yu, Y. Shi, C. Mu, and M. Xu, "Deep-IRTarget: An automatic target detector in infrared imagery using dual-domain feature extraction and allocation," *IEEE Trans. Multimedia*, vol. 24, pp. 1735–1749, 2022.

[16] B. Chakravarthi, S.-C. Ng, M. R. Ezilarasan, and M.-F. Leung, "EEG-based emotion recognition using hybrid CNN and LSTM classification," *Frontiers Comput. Neurosci.*, vol. 16, Oct. 2022.

[17] F. Fang, L. Li, Y. Gu, H. Zhu, and J.-H. Lim, "A novel hybrid approach for crack detection," *Pattern Recognit.*, vol. 107, Nov. 2020, Art. no. 107474.

[18] Z. Yu, Y. Shen, and C. Shen, "A real-time detection approach for bridge cracks based on YOLOv4-FPM," *Autom. Construct.*, vol. 122, Feb. 2021, Art. no. 103514.

[19] Y. Zheng, S. Wu, D. Liu, R. Wei, S. Li, and Z. Tu, "Sleeper defect detection based on improved YOLO V3 algorithm," in *Proc. 15th IEEE Conf. Ind. Electron. Appl. (ICIEA)*, Nov. 2020, pp. 955–960.

[20] F. Guo, Y. Qian, and Y. Shi, "Real-time railroad track components inspection based on the improved YOLOv4 framework," *Autom. Construct.*, vol. 125, May 2021, Art. no. 103596.

[21] S.-H. Chen and C.-C. Tsai, "SMD LED chips defect detection using a YOLOv3-dense model," *Adv. Eng. Informat.*, vol. 47, Jan. 2021, Art. no. 101255.

[22] H. Xie, Y. Li, X. Li, and L. He, "A method for surface defect detection of printed circuit board based on improved YOLOv4," in *Proc. IEEE 2nd Int. Conf. Big Data, Artif. Intell. Internet Things Eng. (ICBAIE)*, Mar. 2021, pp. 851–857.

[23] Q. Ren, J. Geng, and J. Li, "Slighter faster R-CNN for real-time detection of steel strip surface defects," in *Proc. Chin. Autom. Congr. (CAC)*, Nov. 2018, pp. 2173–2178.

[24] Z. Liu, X. Wang, and X. Chen, "Inception dual network for steel strip defect detection," in *Proc. IEEE 16th Int. Conf. Netw., Sens. Control (ICNSC)*, May 2019, pp. 409–414.

[25] Y. He, K. Song, Q. Meng, and Y. Yan, "An end-to-end steel surface defect detection approach via fusing multiple hierarchical features," *IEEE Trans. Instrum. Meas.*, vol. 69, no. 4, pp. 1493–1504, Apr. 2020.

[26] G. Song, K. Song, and Y. Yan, "EDRNet: Encoder–decoder residual network for salient object detection of strip steel surface defects," *IEEE Trans. Instrum. Meas.*, vol. 69, no. 12, pp. 9709–9719, Dec. 2020.

[27] A. M. Deshpande, A. A. Minai, and M. Kumar, "One-shot recognition of manufacturing defects in steel surfaces," *Proc. Manuf.*, vol. 48, pp. 1064–1071, Jan. 2020.

[28] D. Amin and S. Akhter, "Deep learning-based defect detection system in steel sheet surfaces," in *Proc. IEEE Region Symp. (TENSYMP)*, Jun. 2020, pp. 444–448.

[29] X. Cheng and J. Yu, "RetinaNet with difference channel attention and adaptively spatial feature fusion for steel surface defect detection," *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–11, 2021.

[30] A. Urbonas, V. Raudonis, R. Maskeliunas, and R. Damasevicius, "Automated identification of wood veneer surface defects using faster region-based convolutional neural network with data augmentation and transfer learning," *Appl. Sci.*, vol. 9, no. 22, p. 4898, Nov. 2019.

[31] S. Pan, S. Fan, S. W. K. Wong, J. V. Zidek, and H. Rhodin, "Ellipse detection and localization with applications to knots in sawn lumber images," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2021, pp. 3892–3901.

[32] D. Li, W. Xie, B. Wang, W. Zhong, and H. Wang, "Data augmentation and layered deformable mask R-CNN-based detection of wood defects," *IEEE Access*, vol. 9, pp. 108162–108174, 2021.

[33] J. Shi, Z. Li, T. Zhu, D. Wang, and C. Ni, "Defect detection of industry wood veneer based on NAS and multi-channel mask R-CNN," *Sensors*, vol. 20, no. 16, p. 4398, Aug. 2020.

[34] F. Ding, Z. Zhuang, Y. Liu, D. Jiang, X. Yan, and Z. Wang, "Detecting defects on solid wood panels based on an improved SSD algorithm," *Sensors*, vol. 20, no. 18, p. 5315, Sep. 2020.

[35] Y. Fang, X. Guo, K. Chen, Z. Zhou, and Q. Ye, "Accurate and automated detection of surface knots on sawn timbers using YOLO-V5 model," *BioResources*, vol. 16, no. 3, pp. 5390–5406, Jun. 2021.

[36] C. Yu, E. Bian, and Y. Wang, "Research on surface defect detection method of solid wood board based on improved YOLOv5s," in *Proc. IEEE 4th Int. Conf. Civil Aviation Saf. Inf. Technol. (ICCASIT)*, Oct. 2022, pp. 959–963.

[37] A.-A. Tulbure, A.-A. Tulbure, and E.-H. Dulf, "A review on modern defect detection models using DCNNs—Deep convolutional neural networks," *J. Adv. Res.*, vol. 35, pp. 33–48, Jan. 2022.

[38] B. Wang, C. Yang, Y. Ding, and G. Qin, "Detection of wood surface defects based on improved YOLOv3 algorithm," *BioResources*, vol. 16, no. 4, pp. 6766–6780, Aug. 2021.

[39] Z. Zhao, X. Yang, Y. Zhou, Q. Sun, Z. Ge, and D. Liu, "Real-time detection of particleboard surface defects based on improved YOLOv5 target detection," *Sci. Rep.*, vol. 11, no. 1, pp. 1–15, Nov. 2021.

[40] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[42] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "CSPNet: A new backbone that can enhance learning capability of CNN," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 390–391.

[43] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2117–2125.

[44] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.

[45] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-IoU loss: Faster and better learning for bounding box regression," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 7, pp. 12993–13000.

[46] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 2012, pp. 437–478.
[47] Z. Yao, Y. Cao, S. Zheng, G. Huang, and S. Lin, "Cross-iteration batch normalization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 12331–12340.
[48] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1139–1147.
[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[50] T. Tieleman and G. Hinton, "Lecture 6.5-RMSPROP: Divide the gradient by a running average of its recent magnitude," *COURSERA, Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.
[51] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[52] F. Schneider, L. Balles, and P. Hennig, "DeepOBS: A deep learning optimizer benchmark suite," 2019, *arXiv:1903.05499*.
[53] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, "On empirical comparisons of optimizers for deep learning," 2019, *arXiv:1910.05446*.
[54] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
[55] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.
[56] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
[57] K. You, M. Long, J. Wang, and M. I. Jordan, "How does learning rate decay help modern neural networks?" 2019, *arXiv:1908.01878*.
[58] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," 2016, *arXiv:1608.03983*.
[59] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 56, pp. 1929–1958, 2014.
[60] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[61] M.-H. Guo, T.-X. Xu, J.-J. Liu, Z.-N. Liu, P.-T. Jiang, T.-J. Mu, S.-H. Zhang, R. R. Martin, M.-M. Cheng, and S.-M. Hu, "Attention mechanisms in computer vision: A survey," *Comput. Vis. Media*, vol. 8, no. 3, pp. 331–368, 2022.
[62] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7132–7141.
[63] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "CBAM: Convolutional block attention module," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 3–19.
[64] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "ECA-Net: Efficient channel attention for deep convolutional neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11534–11542.
[65] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.
[66] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*.
[67] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 21–37.
[68] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.
[69] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, Y. Kwon, K. Michael, J. Fang, Z. Yifu, C. Wong, D. Montes, Z. Wang, C. Fati, J. Nadar, V. Sonck, P. Skalski, A. Hogan, D. Nair, M. Strobel, and M. Jain, "Ultralytics/YOLOv5: V7.0—YOLOv5 SOTA realtime instance segmentation," Tech. Rep., Nov. 2022, doi: 10.5281/zenodo.7347926.
[70] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO series in 2021," 2021, *arXiv:2107.08430*.
[71] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988.
[72] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10781–10790.

WEI-HAN LIM received the B.E. degree (Hons.) in electrical and electronic engineering from Universiti Tunku Abdul Rahman (UTAR), Selangor, Malaysia, in 2021, where he is currently pursuing the M.Eng.Sc. degree with the Lee Kong Chian Faculty of Engineering and Science (LKCFES). His research interests include artificial intelligence, computer vision, and object detection.

MOHAMMAD BABRDEL BONAB (Member, IEEE) received the Ph.D. degree in philosophy from Universiti Teknologi Malaysia. He was a Postdoctoral Research Fellow with Universiti Tunku Abdul Rahman (UTAR), Malaysia, where he is currently an Assistant Professor of computer science with the Lee Kong Chian Faculty of Engineering and Science. His research interests include artificial intelligence, deep learning, image processing, machine learning, and evolutionary algorithms.

KEIN HUAT CHUA (Senior Member, IEEE) was born in Johor, Malaysia, in 1979. He received the B.E. degree in electrical, electronic and systems from Universiti Kebangsaan Malaysia (UKM), in 2004, the M.E. degree in electrical energy and power system from Universiti Malaya (UM), in 2009, and the Ph.D. degree in electrical engineering from Universiti Tunku Abdul Rahman (UTAR), in 2016. He is currently an Associate Professor with the Department of Electrical and Electronic Engineering, Lee Kong Chian Faculty of Engineering, UTAR. He is also a Chairperson of the Centre for Railway Infrastructure and Engineering (CRIE), UTAR. His research interests include energy storage systems, energy management and audit, power quality, and railway electrification.