Selection of Lightweight CNN Models with Limited Computing Resources for Drone Collision Prediction

Rifqi Nabila Zufar, Student Member and **David Banjerdpongchai**[†], Member

ABSTRACT

The Collision Avoidance System (CAS) is a safety system created to identify and prevent collisions, primarily on drones. The CAS comprises three processes: detection, prediction, and action. The predictive process is crucial as it determines whether a collision will occur, making it the core component of the system. Most drones are equipped with cameras. A visual-based prediction involves the use of a convolutional neural network (CNN). The CNN operates by autonomously learning and extracting hierarchical characteristics from input data through convolution, pooling, and fully connected layers. Currently, there are CNN models called pretrained models that are ready to use. However, not all pretrained models are suitable for compatibility with drones as they possess computational constraints. Our objective is to establish a suitable model selection from a variety of pretrained CNN models with lightweight architectures. The transfer learning technique is applied to customize these models with the ColANet dataset. Subsequently, we evaluate these models regarding their accuracy, model size, inference time, and power consumption. Finally, the selected model is deployed in real time on a Raspberry Pi 3B+ with data input from a DJI Tello drone camera, and the prediction performance is evaluated.

Keywords: Drone Collision Prediction, Lightweight CNN Models, ColANet Dataset, Transfer Learning, Raspberry Pi 3B+

1. INTRODUCTION

Unmanned Aerial Vehicles (UAVs) are an innovative technology that has transformed many industries, including military [1], surveillance [2], agriculture [3], photography [4], and scientific research [5]. The UAVs often referred to as drones, come in various types and configurations, each designed for specific purposes and applications. For example, fixed-wing drones are conventional aircraft with fixed wings that provide stability and are designed for efficiency and endurance, making them ideal for missions such as long-range surveillance, mapping, and cargo delivery [6]. On the other hand, there are also drones that are highly manoeuvrable and stable called multirotor drones. This type of drone uses multiple rotors to generate vertical lift. It is commonly used for aerial photography, videography, and close-up inspections [7]. Based on their size, drones can be divided into several categories, heavy drones, and light drones. Heavy drones provide significant benefits, including substantial payload capacity, extended flight duration, stability, and exceptional computational ability, making them ideal for transporting equipment, incorporating advanced sensors, and performing heavy data processing tasks. In contrast, light drones are valued for their affordability, portability, and ease of use, enabling a widespread user base for various applications. However, limited payload capacity, shorter flight duration, and limited computational ability are drawbacks [8]. In their operations, drones tend to rely on sensors to detect the surrounding environment and provide the input needed to the drone system. According to Yasin et al., [9], drone sensor was categorized into two sets in which active sensor and passive sensor. The active sensor worked on the mechanism of emission of radiation and reading the reflected radiation. An active sensor had its transmitter and receiver. A passive sensor worked to detect the energy discharged by the objects or the scenery under observation. The benefits of the passive sensor consumed less energy compared to active sensors. Optical or visual sensors are cameras such as monocular or stereo cameras that work in visible light. Visual sensors or cameras rely on capturing images of the environment and objects to give useful information to be extracted and then are going to fed to the drone safety system, especially to the Collision Avoidance System (CAS).

In general, a CAS is a system intended to avoid or a lessen collision between a vehicle and an obstacle. It may enhance safety by detecting potential collision risks and implementing corrective measures to avert incidents. The CAS incorporates three procedures which are sensing, prediction, and decision. The sensing procedure employs available sensors to obtain information on the identity and location of obstacles which then is fed to the prediction procedure. In this step, information is processed which then produces an output in the form

Manuscript received on October 5, 2023; revised on November 14, 2023; accepted on January 23, 2024. This paper was recommended by Associate Editor Matheepot Phattanasak.

The authors are with Center of Excellence in Intelligent Control Automation of Process Systems, Department of Electrical Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand.

[†]Corresponding author: bdavid@chula.ac.th

Digital Object Identifier: 10.37936/ecti-eec.2024221.252934

of a prediction of whether the obstacle is in a collision condition or not. According to Wei et al., [10], there were prediction methods namely classical and heuristic algorithm. Compared with the classical methods of searching for the optimal solution, heuristic algorithms aimed to find the appropriate solution according to historical experience. The solution of the heuristic algorithm was approximately optimal. One type of heuristic algorithm was the Convolutional Neural Network (CNN). CNN is a type of artificial neural network designed specifically for processing and analyzing visual data, such as images and videos. The prediction output of CNN leads to the decision step. If the drone determines that a collision is likely, the decision procedure can schedule a flight path that avoids the collision promptly.

Currently, there are existing pretrained models available for application. There is no need to build models from scratch. The models can be further developed including the transfer learning [11]. This technique transfers the knowledge learned from one task is reused to improve the performance of a related task. The lightweight CNN models are a type of CNN architecture that is designed to be computationally efficient, smaller in size, and require fewer computational resources compared to larger and more complex CNN models. The models are particularly important in applications where computational constraints, such as limited hardware resources or real-time processing, are a concern. The design criteria of lightweight CNN models are small model size, small parameters, faster inference time, and lower computational requirements.

In a previous work [12], Zufar and Banjerdpongchai conducted a comparative study on performance assessment of lightweight CNN pre-trained models for the collision avoidance. The pre-trained models were chosen with less than ten million parameters from the Keras library. The assessed performance metrics included validation accuracy, training accuracy, model size, inferential time, and model complexity. The results showed that the best performance was achieved by the pre-trained model of MobileNet. However, the study was only a preliminary simulation. It was suggested that the CNN models on a real-time system should be investigated.

In this work, we conduct a comprehensive study to evaluate the lightweight CNN pre-trained models. We change a design structure of CNN models and employ an image augmentation function for data augmentation. Then, we determine the performance of the models in terms of accuracy, model size, inference time, and power consumption. Hereafter, the best model is implemented in a real-time system, where the model is running on a Raspberry Pi 3B+ with the input images from the DJI Tello drone camera. The key contributions of this paper are as follows.

1. We modify the training process, such as the number of epochs, the percentage of the dropout layer, and the number of dense neuron layers, and employ the RandAugment for image augmentation to enhance the robustness of model. Then, we determine the performance of lightweight CNN pretrained models on the testing (unseen) data. The testing data were four VDOs, which was not used in the previous study.
2. We analyse the impact of transfer learning that has on the testing accuracy. In addition, we reduce model complexity by compressing the model using post-training quantization and determine the effect of compression on the performance.
3. We deploy the best obtained CNN model on edge device Raspberry Pi 3B+ and feed real-time data from the DJI Tello drone. Then, we evaluate the performance of model in real-time system.

The remainder of this paper consists of five sections. We present the problem statement in Section 2 and the methodology in Section 3. Then, we present the result of the experiment and discussion in Section 4. Finally, we provide the conclusion and ongoing work in Section 5.

2. PROBLEM STATEMENT

The prediction model proposed by [13], was specifically designed for predicting collision events in drones. The model utilized a pre-trained MobileNetV2 CNN as its feature extractor (FE). This FE demonstrated a training accuracy of 94.34% and a validation accuracy of 80.29% when applied to classification tasks using the ColANet dataset. The notable disparity between training and validation accuracy suggests a potential issue of overfitting with this FE. Furthermore, it is important to note that the FE has not been evaluated on previously unseen data, indicating a lack of testing on new tasks. Additionally, given the substantial computational processing demands, this FE may not be well-suited for limited computing environments, particularly those associated with drone computing.

1. Model size. Edge devices have limited storage and processing capabilities. Models should be small in terms of memory space and computational complexity.
2. Energy consumption. Edge devices are often battery-powered or have limited power resources. Models should be optimized for energy efficiency, minimizing the CPU usage during inference, and avoiding unnecessary computations.

To determine the suitable model for implementation, we employ the performance assessment as follows:

1. We compare the accuracy of lightweight pretrained CNN models using testing data before and after implementing transfer learning.
2. Based on the performance of the lightweight pretrained CNN models, we determine a suitable model. The performance metrics consist of testing accuracy, inference time, model size, and power consumption Then, we assess the chosen model by using a confusion

***Fig. 1**: The three sample frames of three ColANet dataset VDOs.*

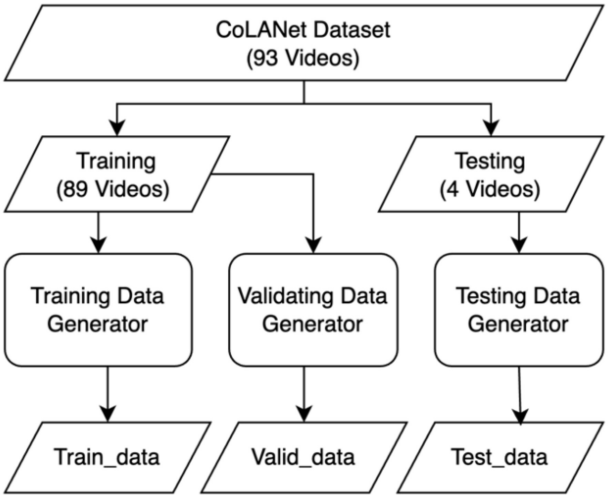

***Fig. 2**: The flow of data preprocessing.*

matrix.

3. We evaluate the performance of TensorFlow, TensorFlow-Lite without optimization and TensorFlow-Lite with optimization to determine the best model according to the performance metrics.
4. We deploy the best model on Raspberry Pi 3B+ with input source from DJI Tello drone camera and analyse the performance.

3. METHODOLOGY

In this section, we present the methodology for the prediction. Subsection 3.1 gives the data preprocessing. In subsection 3.2, we describe how the CNN model is built. In subsection 3.3, we review the transfer learning technique. Subsection 3.4 describes how to convert the models from TensorFlow to TensorFlow-Lite. Subsection 3.5 describes how to deploy the model in real-time. Lastly, subsection 3.6 gives the metrics to evaluate the performance of the models.

3.1 Data Preprocessing

In this work, we use the ColANet dataset as a data resource. The ColANet dataset is an open archive of collisions between drones and is intended to be the first step towards a collision-free and safe operation of drones. This dataset is available for download [15]. This dataset consists of 93 drone VDOs that represent collision events on drones. The data was classified into two categories namely no collision and collision [16]. Some samples from flights of different models in different environmental conditions are shown in Figure 1.

Figure 2 illustrates the flow of data preprocessing. The dataset is processed to produce Train_data for the training process, Valid_data for validation, and Test_data for testing of unseen data. The initial step involves partitioning 93 videos of ColANet dataset into 89 videos for training purposes, with with the remaining four videos allotted for testing or unseen data processing. The 89 videos are then transformed into frames with a total of 5986 frames, 90% of which are used in the training process, 10% of which are used in the validation process. The four remaining videos will also undergo conversion to frames, resulting in a total of 181 frames. To expand the range of training data, we incorporate a data augmentation technique known as RandAugment [17]. This technique simplifies the process of training data augmentation by randomly implementing pre-defined sequences of image transformations with corresponding magnitudes. By introducing this randomness, the training dataset becomes more diverse, thereby improving the model's robustness and generalization abilities. Figure 3 illustrates an implementation of RandAugment on ColANet dataset images. Finally, all frames are inserted into the data generator function by setting the number of batch frames displayed to 16 batches.

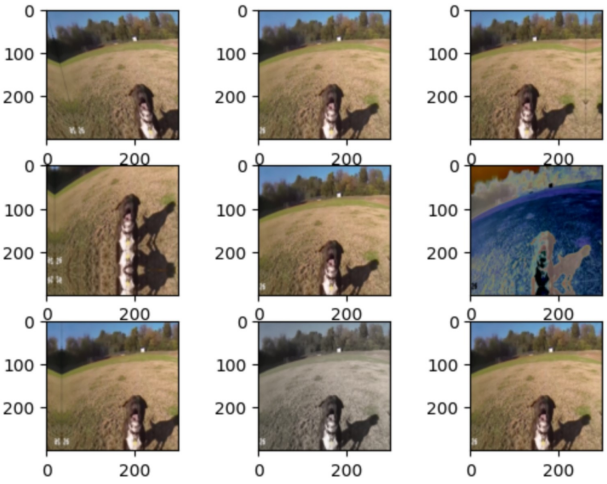

***Fig. 3**: RandAugment implementation on images on ColANet dataset.*

3.2 Lightweight Pretrained CNN Models

The lightweight CNN models refer to a CNN architecture designed to have a relatively small number of

***Table 1**: Lightweight pretrained CNN models.*

Model	Alias	Parameter	Size
EfficientNetB2	ENB2	9.2 M	36 MB
EfficientNetV2B2	ENV2B1	8.2 M	34 MB
DenseNet121	DN121	8.1 M	33 MB
EfficientNetB1	ENB1	7.9 M	31 MB
EfficientNetV2B0	ENV2B0	7.2 M	29 MB
MobilenetV3Large	MNV3L	5.4 M	21 MB
EfficientNetB0	ENB0	5.3 M	29 MB
NASNetMobile	NNB	5.3 M	23 MB
MobileNet	MN	4.3 M	16 MB
MobileNetV2	MNV2	3.5 M	14 MB
MobilenetV3Small	MNV3S	2.9 M	10 MB

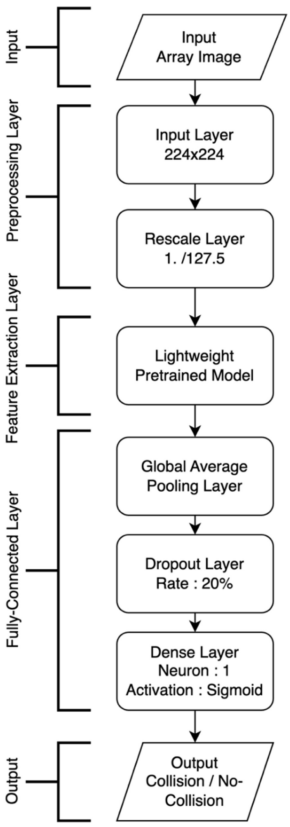

***Fig. 4**: The flow of CNN model design.*

parameters, making it more computationally efficient and suitable for use on resource-constrained devices or systems. The pre-trained CNN models are accessible in the Keras Library. This study employs 11 models from the previous research [12] shown in Table 1. The performance models were derived from the ImageNet validation dataset, deployed on an AMD EPYC processor (with IBPB) (92 cores), 1.7T RAM, and Tesla A100 GPU, using a batch size of 32 [18].

Then, the lightweight pretrained CNN models defined in Table 1 is combined with the input layer, preprocessing layer, fully connected layer, and output layer which is illustrated in Figure 4. The preprocessing layer manages

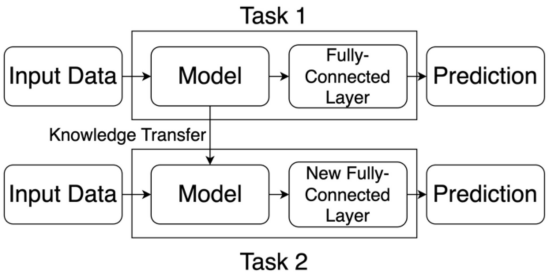

***Fig. 5**: Illustration of transfer learning process.*

the input of batch data. It incorporates an input layer with a shape dimension of 224×224 and a rescale layer with a ratio of 1/127.5. The resulting output from the preprocessing layer feeds into the feature extraction layer. This latter layer extracts vital information from an image, using a lightweight pretrained model with Imagenet as the initial weight. Following this, the model then goes on to include the prediction or fully connected layer. This layer contains a global average pooling layer, a dropout layer with a 20% rate, and a single dense neuron layer. The model output is subsequently activated by a sigmoid function where the outcome denotes 0 for no_collision and 1 for collision. This model design is then compiled using the Adam optimizer with a learning rate of 10e-5 as the optimizer function, binary cross entropy as the loss function, and binary accuracy as the metric function to produce a model called the pretrained model.

3.3 Transfer Learning

Transfer Learning is a machine learning technique that enhances or adjusts models trained on a particular problem for use on another problem. It utilizes knowledge acquired from the initial task to enhance performance on the subsequent task, despite any dissimilarities in the data distribution or characteristics. An illustration of this process is depicted in Figure 5.

The previously built pretrained model illustrated in figure 4 will be trained using the Train_data and validated with Valid_data. There are two processes in transfer learning: the trained process involves freezing the weight of the model, while the fine-tuned process involves unfreezing the weight of the model. In the trained process, the model weights are immobilized to prevent any changes during the training process. In the fine-tuning process, the weights on the model are not locked so that they change and adapt to the given data. It is worth noting that altering the weighting of the optimized model has potential to positively impact the training process, leading to improved accuracy in the model. It is worth considering that such alteration may conversely result in a lower accuracy. Then, the model will be compiled with Adam optimizer with a learning rate of 10^{-5} as the optimizer function, binary cross entropy as the loss function, and binary accuracy as the metric function. At the end, the model is trained with 10 epochs

***Table 2**: Summary of compression techniques.*

Compression technique	Compression parameters
TFLite with float-16 post-training quantization	Weights, biases, activations, and outputs are discounted to 16-bit floats
TFLite with dynamic-range post-training quantization	Only weights are reduced to 8-bit integers. Activations are dynamically down scaled to 8-bit at run-time for faster computation

for the trained process and 10 epochs for the fine-tuned process by setting the batch size of each training process to 16 batches. This process leads to two distinct models, namely, the trained model and fine-tuned model.

3.4 TensorFlow-Lite

TensorFlow-Lite (TFLite) was developed by Google as an open-source framework for deploying machine learning models on mobile and embedded devices where resources are limited [19]. This framework effectively optimizes and converts TensorFlow models into a lightweight format that is well-suited to edge computing scenarios. The TFLite model will be saved in FlatBuffer format which aims to to enhance both memory efficiency and speed when inferring models on edge devices. An optimization technique known as post-training quantization can be used in TFLite models as well. This technique requires altering the numerical representation of the model weights and/or activations to less precise data types. In this research, we will use two types of post-training quantization, float-16, and dynamic-range. We give a brief explanation of the optimization in Table 2.

The TFLite model with float-16 post-training quantization is a compression method that reduces full-precision 32-bit floating point data to lower precision, specifically 16-bit precision. The weights, biases, and activations are stored in 16-bit format and the calculation is executed in the same format. The output is also saved in a 16-bit format, and this compression reduces the model size by approximately 50%. Meanwhile, the TFLite model with dynamic-range post-training quantization as a compression method to combine both 8-bit floating-point and integer precision. The model weights and biases are statistically scaled down from floating-point to integer precision by pre-calculating the scale factors and zero points. During runtime, activations are dynamically quantized to 8-bit precision to enable calculations between activations and weights to occur in 8-bit precision. However, the output is subsequently transformed and saved as a floating-point number. This reduces the computational delay of inference to a level comparable to fixed-point integer operations. As a result, the model size typically decreases by 75%.

***Table 3**: Confusion matrix.*

	Predicted Class	
True Class	TN (True Negative) Correct absence of result	FP (False Positive) Unexpected result
	FN (False Negative) Missing result	TP (True Positive) Correct result

3.5 Running CNN Model on Edge Device

The compressed model of TFlite is deployed on Raspberry Pi 3B+ as a representative of an edge device. This board is a single-board computer designed for a wide range of computing projects. It has a 1.4GHz quad-core ARM Cortex-A53 CPU and 1GB of RAM [20]. The input data source derives from a video stream captured by the DJI Tello drone camera through the UDP protocol, which provides 720p HD video resolution [21]. The process of capturing the video stream and predicting will be carried out simultaneously by creating a two-thread function.

3.6 Evaluation Metrics

We investigate the performance of models using a confusion matrix, model size, inference time, and power consumption. In Table 3, The confusion matrix evaluates prediction accuracy where each row represents the actual class. The predictions of the model are compared with the actual results in a confusion matrix, resulting in four key indicators [22].

In Eq. (1), Accuracy rate is the proportion of correctly predicted samples (both true positives and true negatives) out of the total number of samples in the data set. This metric provides an encompassing evaluation of the model's performance across all classes. In Eq. (2), Precision measures how accurately the model identifies positive instances in its predictions. In terms of the confusion matrix, precision relates to the top row of the matrix (positive predictions), highlighting the significance of reducing false positives (FP) to maximise precision. In Eq. (3), The recall measures the model's ability to capture all positive instances among the actual cases. In the context of the confusion matrix, recall is associated with the left column (actual positive cases), emphasizing the importance of minimizing false negatives (FN) to maximize recall. In Eq. (4), The F1 Score balances the trade-off between precision and recall, ensuring that a model performs effectively in minimizing both false positives and false negatives. When calculating the F1 score using the confusion matrix, it considers both the top row (precision) and left column (recall) to derive a singular metric that takes into account both facets of model performance.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F1 = 2 \times \frac{Precision\ x\ Recall}{Precision + Recall} \quad (4)$$

The size of a CNN model can be determined by adding up the overall number of trainable parameters in the network. This involves tallying the weights and biases in each layer, such as convolutional or fully connected layers, and calculating the sum. The quantity of parameters in a convolutional layer is established by the size of the convolutional kernels, the quantity of input and output channels, and any biases applied. Similarly, the number of input and output neurons, along with biases, determines the size of fully connected layers. It is worth noting that size is usually measured in terms of the total number of parameters, with each parameter represented as a 32-bit (4-byte) floating-point number. To calculate the size in bytes, multiply the total number of parameters by 4 [23].

Inference time, also called latency, is the time that a machine learning model needs for processing and producing prediction results for a single input data point. It means the time that takes to predict an image [24].

The power consumption of a model of a machine learning system such as a neural network or any other computing system is the amount of electric power consumed while the model is running. This is influenced by a variety of factors, comprising the model design, the training or inference hardware and the computational burden.

4. RESULTS AND DISCUSSION

The design model in Figure 4 was trained on a Macbook Pro 2020. It has the M1 chip, which integrates an 8-core CPU, an 8-core GPU, and a 16-core neural engine for AI tasks with 8 GB RAM. Using Visual Studio Code version 1.8.1, python version 3.9.17, TensorFlow version 2.12.0, and Keras Library version 2.12.0 as its environment. Table 4 is a table performance of the lightweight pre-trained CNN models that were evaluated on the testing data from the ColANet dataset before the training.

Table 4 displays that all models have a testing accuracy below 62.00%. EfficientNetB1 has the highest accuracy of 61.11%, whereas EfficientNetV2B0 has the lowest at 31.67%. Concerning model size, EfficientNetB2 is the heaviest with 29.64 MB and 7.71 million parameters, and MobileNetV3Small is the lightest with just 3.58 MB and 0.93 million parameters.

Then, the models are trained by freezing the model weights with the aim of improving the accuracy testing of models. The results indicate an improvement in the testing accuracy for each model, with EfficientNetB2 achieving the highest accuracy of 81.67%, while MobileNetV3Large demonstrated the lowest at 60.01%. Furthermore, there is no significant difference between the accuracy of the training and validation. This suggests that the models can validate the given images effectively. Next, the model is retrained by unfreezing the weights of all the weights of the model. The results show that the training and validation accuracies have increased to over 95% and 93% on average. However, in terms of testing accuracy, only EfficientNetB1, EfficientNetV2B0, and NasNetMobile exhibit an enhancement, while the remaining models illustrate either no improvement or a decline in accuracy. Table 5 and Figure 6 show the accuracy of trained models and fine-tuned models and a comparison of testing accuracy among pretrained, trained, and fine-tuned models, respectively.

***Table 4**: Testing accuracy of pretrained models on unseen data.*

Model	Parameter [M]	Size [MB]	Accuracy [%]
ENB2	7.71	31.71	49.44
ENV2B1	6.91	28.34	57.78
DN121	7.03	28.94	60.56
ENB1	6.63	26.92	61.11
ENV2B0	5.92	23.98	31.67
MNV3L	2.99	12.39	50.56
ENB0	4.05	16.64	41.11
NNB	4.27	18.52	51.11
MN	3.22	13.11	51.67
MNV2	2.25	9.42	45.00
MNV3S	0.93	4.09	48.33

Since most of the models are not improved in the fine-tuned process, we decide to use the models from the trained process for the next assessment. The models are evaluated based on the memory size, inference time, and power consumption. PowerMetrics [25], a library on the Macbook Pro records the power consumed by the CPU and GPU on M1 chip processors. The power consumption is utilized to gauge the resource usage of the models. Note that the average power consumption required to execute the prediction model is 2969.52 mW.

Table 6 shows that DenseNet121 with 28.94 MB has an inference time of 0.070 second/image and consumes power of 0.21 watt which is the highest consumption compared to other models. Meanwhile, EfficientNetB2 with a model size of 31.71 MB has an inference time of 0.042 second/image and consumes 0.12 watts/image which is lower than Densenet121. This shows that EfficientNetB2 has higher efficient computation and better accuracy than Densenet121. On the other hand, we compare the performance of EfficientNetB2 with that of MobilenetV3Small. MobilenetV3Small which is the smallest size achieves better performance with inference time of 0.008 second/image and power consumption of 0.02 watts/image, although it has slightly lower accuracy than EfficientNetB2. In this study, Mobilenet achieves lower performance than MobilenetV3Small. Therefore,

***Table 5**: Accuracy of trained and fine-tuned models.*

Model	Process	Trained Accuracy [%]	Fine-Tuned Accuracy [%]
ENB2	Train	82.64	98.63
	Validation	83.14	98.33
	Test (Unseen)	81.67	79.44
ENV2B1	Train	81.61	98.94
	Validation	82.47	97.83
	Test (Unseen)	71.11	67.22
DN121	Train	77.49	96.42
	Validation	78.96	95.99
	Test (Unseen)	64.99	64.99
ENB1	Train	83.19	98.31
	Validation	83.81	98.33
	Test (Unseen)	68.89	77.22
ENV2B0	Train	82.17	98.38
	Validation	81.47	97.33
	Test (Unseen)	63.89	66.11
MNV3L	Train	80.41	98.22
	Validation	82.81	96.83
	Test (Unseen)	60.01	58.89
ENB0	Train	83.68	98.77
	Validation	84.14	97.33
	Test (Unseen)	77.78	75.56
NNB	Train	78.98	96.71
	Validation	78.46	96.49
	Test (Unseen)	63.89	80.56
MN	Train	81.01	95.93
	Validation	81.64	93.49
	Test (Unseen)	62.78	56.11
MNV2	Train	80.21	96.25
	Validation	82.81	96.99
	Test (Unseen)	70.56	48.33
MNV3S	Train	76.99	97.44
	Validation	78.63	96.16
	Test (Unseen)	78.88	75.01

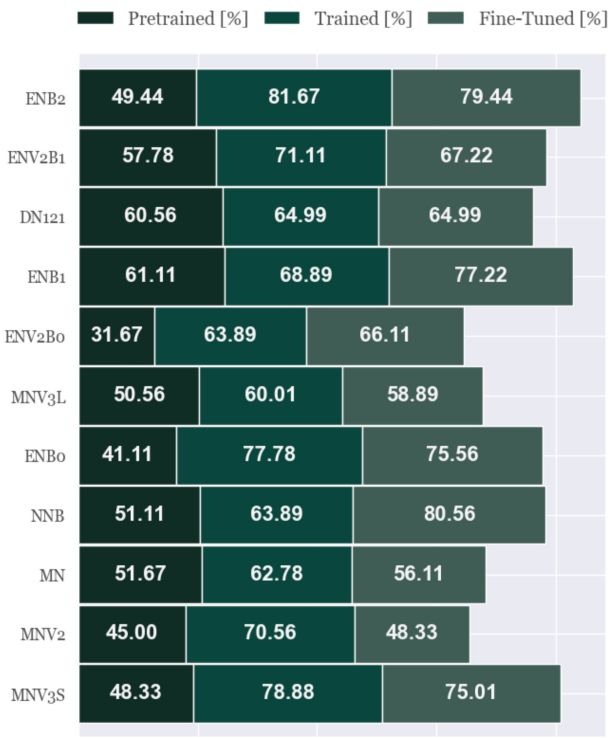

***Fig. 6**: Comparison of testing accuracy.*

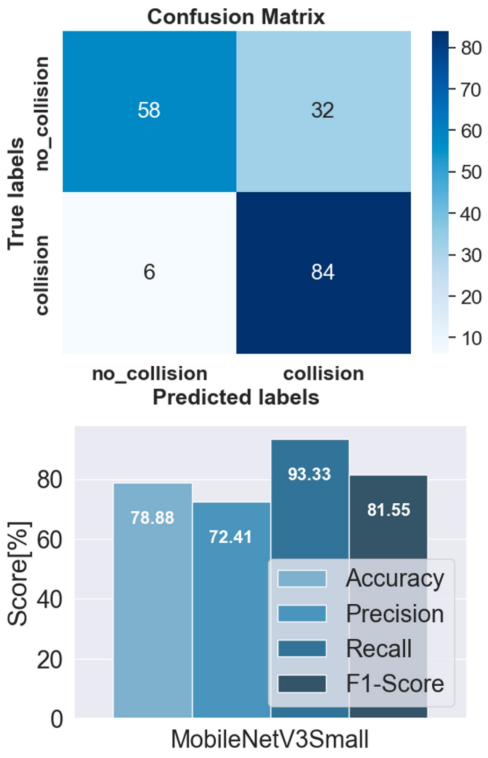

***Fig. 7**: Confusion matrix of MobileNetV3Small model.*

we decide to use MobilenetV3Small for the real-time prediction. Hereafter, the model is measured with confusion matrix to get the precision score, recall score and F1 score. Figure 7 displays the confusion matrix of MobileNetV3Small.

The MobileNetV3Small model achieves an accuracy score of 78.89% and a precision score of 72.41%, demonstrating its ability to accurately identify positive cases among its predictions. With a high recall score of 93.33%, it demonstrates the effectiveness in capturing a significant proportion of actual positive cases, minimising the chance of missing important cases. These combined metrics result in an impressive F1 score of 81.55%, indicating a balanced performance between precision and recall. This model is well suited for tasks where both precision and recall are critical, ensuring accurate positive predictions while minimizing false negatives, making it valuable in collision prediction.

Furthermore, the model is converted to the TFLite environment. There are three models resulting from the

***Table 6**: Performance of trained CNN models.*

Model	Size [MB]	Accuracy [%]	Inf_Time [Sec/ Img]	Power [W/ Img]
ENB2	31.71	81.7	0.042	0.12
ENV2B1	28.34	71.1	0.031	0.09
DN121	28.94	65.0	0.070	0.21
ENB1	26.92	68.9	0.037	0.11
ENV2B0	23.98	63.9	0.024	0.07
MNV3L	12.39	60.0	0.015	0.05
ENB0	16.64	77.8	0.027	0.08
NNB	18.52	63.9	0.031	0.09
MN	13.11	62.8	0.018	0.05
MNV2	9.42	70.6	0.016	0.05
MNV3S	4.09	78.9	0.008	0.02

***Table 7**: Performance comparison of post-training quantization for MobilenetV3Small.*

Model	Size [MB]	Accuracy [%]	Inf_Time [Sec/ Img]	Power [W/ Img]
TFLite w/o optimization	3.74	78.88	0.0034	0.0068
TFLite with Float-16	1.92	79.44	0.0034	0.0069
TFLite with Dynamic-range	1.11	77.78	0.0029	0.0060

***Table 8**: Model performance on Raspberry Pi 3B+.*

Model	Size [MB]	Accuracy [%]	Inf_Time [Sec/ Img]	Power [W/ Img]
TFLite with dynamic range	1.11	77.78	0.090	0.36

conversion, namely TFLite without optimization, TFLite with float-16 post-training optimization and TFLite with dynamic range optimization. Each model is evaluated to determine accuracy, model size, inference time and power consumption. Furthermore, we compare performance between TensorFlow and TFLite on desktop environment. Table 7 shows that the results of converting TensorFlow model to TFLite model reduce 8.56% of the model size. On the other hand, TFLite with post-training float-16 optimization reduces the model size by 53.15%, while TFLite with post-training dynamic range optimization can reduce the model size by 73.01%. In addition, the accuracy of TFLite model is similar to TensorFlow model. The accuracy of TFLite with float-16 model is slightly increased of 0.71%, while the accuracy of TFLite with dynamic range optimization decrease of 1.39%. For inference time and power consumption, TFLite with dynamic range optimization is the fastest and the least power consumption among other models. Therefore, we conclude that MobilenetV3Small TFLite with dynamic range model shall be deployed on Raspberry Pi 3B+.

Finally, the MobilenetV3Small TFLite with dynamic range is deployed on the Raspberry Pi 3B+ to evaluate their performance. Input images from real-time data taken from DJI Tello drone camera. In addition, the plug-in power meter device 220V 10A 2200W is used for measuring the power consumption of the edge device. Table 8 shows that the accuracy of the MobilenetV3Small TFLite with dynamic range optimization does not change when implemented on the edge device. However, the inference time of the model is slower than that of the model running on the desktop. This is due to the difference in CPU between the desktop environment and the edge environment. Therefore, power consumption is different.

5. CONCLUSION

This paper determines the effectiveness of pre-trained lightweight CNN models for collision avoidance systems. Objective findings demonstrate that testing these models results in approximately 62% average accuracy. The ColANet dataset is necessary for training pre-trained models. The accuracy of pretrained models improves significantly when training with freeze weight. However, unfreeze weight does not show any improvement. In certain instances, there may be a decrease in accuracy. Therefore, it is recommended that freeze weight models be employed in future assessments.

In this study, the results reveal that MobilenetV3Small is superior to other models, including MobileNet, which was previously deemed to be the most effective model. The architecture of MobileNetV3Small yields the fastest inference time of 0.008 second/image and requires low power consumption of 0.02 watt/image. Furthermore, it attains accuracy, precision, recall, and F1 score of 78.88%, 72.41%, 93.33%, and 81.55%, respectively. Furthermore, MobileNetV3Small is converted to TFLite with dynamic range as this technique is the best of the rest. However, there is a trade off when using this technique. The model size, inference time, and power consumption are better, but the testing accuracy slightly decreases to 77.78%. As the MobileNetV3Small model exhibits promising performance, it is then implemented for a collision prediction using a Raspberry Pi 3B+ and DJI Tello drone. The model size and accuracy remain constant, yet there exists a discrepancy in the inference time and power consumption, which is attributed to the difference between desktop CPU and edge device CPU. Considering the performance of MobileNetV2 which is used as an FE in the prediction model [13], MobileNetV3Small provides lighter model performance,

faster inference time and less power consumption with better prediction accuracy. For the record, we refer to table 6. Therefore, MobilenetV3Small has the potential to develop collision prediction models on devices with limited computing resources.

ACKNOWLEDGMENTS

We thank the Ratchhadapisek Sompote Fund of Chulalongkorn University for supporting the Center of Excellence in Intelligent Control Automation of Process Systems. The first author is thankful for the Graduate Scholarship Programme for ASEAN or Non-ASEAN Countries, Chulalongkorn University to support his Master program study.

REFERENCES

[1] Z. Xiaoning, "Analysis of military application of UAV swarm technology," *2020 3rd International Conference on Unmanned Systems (ICUS)*, Nov. 2020.

[2] A. Chriki, H. Touati, H. Snoussi, and F. Kamoun, "UAV-based Surveillance System: An Anomaly Detection Approach," *2020 IEEE Symposium on Computers and Communications (ISCC)*, Jul. 2020.

[3] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas, and I. Moscholios, "A compilation of UAV applications for precision agriculture," *Computer Networks*, p. 107148, May 2020.

[4] J. Wu, S. Wen, Y. Lan, X. Yin, J. Zhang, and Y. Ge, "Estimation of cotton canopy parameters based on unmanned aerial vehicle (UAV) oblique photography," *Plant Methods*, no. 1, Dec. 2022.

[5] N. Ruzhentsev *et al.*, "Radio-Heat Contrasts of UAVs and Their Weather Variability at 12 GHz, 20 GHz, 34 GHz, and 94 GHz Frequencies," *ECTI Transactions on Electrical Engineering, Electronics, and Communications*, no. 2, pp. 163–173, Jun. 2022.

[6] B. J. Brelje and J. R. R. A. Martins, "Electric, hybrid, and turboelectric fixed-wing aircraft: A review of concepts, models, and design approaches," *Progress in Aerospace Sciences*, pp. 1–19, Jan. 2019.

[7] Q. Quan, *Introduction to Multicopter Design and Control*. Springer, 2017.

[8] M. Hassanalian and A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences*, pp. 99–131, May 2017.

[9] J. N. Yasin, S. A. S. Mohamed, M.-H. Haghbayan, J. Heikkonen, H. Tenhunen, and J. Plosila, "Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches," *IEEE Access*, pp. 105139–105155, 2020.

[10] Z. Wei, Z. Meng, M. Lai, H. Wu, J. Han, and Z. Feng, "Anti-Collision Technologies for Unmanned Aerial Vehicles: Recent Advances and Future Trends," *IEEE Internet of Things Journal*, no. 10, pp. 7619–7638, May 2022.

[11] F. Zhuang et al., "A Comprehensive Survey on Transfer Learning," *Proceedings of the IEEE*, no. 1, pp. 43–76, Jan. 2021.

[12] R. N. Zufar and D. Banjerdpongchai, "Performance Comparison of Lightweight CNN Models for Drone Collision Avoidance Dataset," *2023 20th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI-CON)*, May 2023.

[13] D. Pedro, J. P. Matos-Carvalho, J. M. Fonseca, and A. Mora, "Collision Avoidance on Unmanned Aerial Vehicles Using Neural Network Pipelines and Flow Clustering Techniques," Remote Sensing, no. 13, p. 2643, Jul. 2021.

[14] ColANet: A UAV Collision Avoidance Dataset. https://colanet.qa.pdmfc.com/ (accessed Sep. 10, 2023).

[15] D. Pedro, A. Mora, J. Carvalho, F. Azevedo, and J. Fonseca, "ColANet: A UAV Collision Avoidance Dataset," *in IFIP Advances in Information and Communication Technology*, Springer International Publishing, pp. 53–62, 2020.

[16] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jun. 2020.

[17] Keras Team, "Keras Applications," *Keras: Deep Learning for humans*. https://keras.io/api/applications/ (accessed Sep. 09, 2023).

[18] A. Abid, P. Sinha, A. Harpale, J. Gichoya, and S. Purkayastha, "Optimizing Medical Image Classification Models for Edge Devices," *in Distributed Computing and Artificial Intelligence, Volume 1: 18th International Conference*, Springer International Publishing, pp. 77–87, 2021.

[19] R. P. Ltd, "Buy a Raspberry Pi 3 Model B+ – Raspberry Pi," Raspberry Pi. https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/ (accessed Sep. 23, 2023).

[20] "Tello," Tello Official Website-Shenzhen Ryze Technology Co., Ltd. https://www.ryzerobotics.com/tello/specs (accessed Sep. 19, 2023).

[21] "3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 1.3.0 documentation," scikit-learn. https://scikit-learn.org/stable/modules/model_evaluation.html (accessed Sep. 9, 2023).

[22] T. Prasayasith, P. Saengudomlert, and W. S. Mohammed, "Performance Evaluation and Improvement of Wireless RSSI Based Animal Theft Detection", *ECTI Transactions on Electrical Engineering, Electronics, and Communications*, vol. 21, no. 3, p. 251462, Oct. 2023.

[23] Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang. "Optimizing {CNN} model inference on {CPUs}," *In*

2019 USENIX Annual Technical Conference (USENIX ATC 19), pp. 1025-1040, 2019.

[24] "Energy Efficiency Guide for Mac Apps: Monitor Usage Regularly," *Apple Developer*, Sep. 13, 2016.

Rifqi Nabila Zufar (ECTI Student Member) was born in 1996 in Kediri, East Java, Indonesia. He received his Bachelor of Engineering (BEng) degree from the State University of Surabaya, Indonesia in 2019. Recently, He has studied in the Master program in electrical engineering with The Graduate Scholarship Programme for ASEAN or Non-ASEAN Countries, Chulalongkorn University, Thailand. His research interests include UAV, artificial intelligence, and embedded systems.

David Banjerdpongchai (ECTI Member) received his B.Eng. degree (First class honors) from Chulalongkorn University, M.S. and Ph.D. degrees from Stanford University, all in Electrical Engineering. Since 1990, he has been with the Department of Electrical Engineering, Chulalongkorn University. Currently, he is a professor of Electrical Engineering, and the head of Intelligent Control Automation of Process Systems Research Unit. He is a senior member of IEEE, a founding chair of IEEE Control Systems Society Thailand Chapter, and President of ECTI Association (2024-2025). He serves as an associate editor of IJCAS and a section editor-in-chief of ASEAN Engineering Journal. His research interests are advanced process control, robust control systems, and energy management systems.