# Performance Analysis of the Pretrained EfficientDet for Real-time Object Detection on Raspberry Pi

Vidya Kamath
*Department of Computer Science and Engineering,*
*Manipal Institute of Technology*
*Manipal Academy of Higher Education*
Manipal, India
vidyakuldeep@gmail.com

Renuka A.
*Department of Computer Science and Engineering,*
*Manipal Institute of Technology*
*Manipal Academy of Higher Education*
Manipal, India
renuka.prabhu@manipal.edu

***Abstract*—Recently there has been a lot of demand for deep learning models that can operate on a constrained device. When it comes to the task of object detection, EfficientDet is a well-known model. In this study, we use the integer quantization technique to perform real-time object detection on a Raspberry Pi using the popular EfficientDet family. We use the pretrained models from the TensorFlow to perform object detection for a specific task and evaluate their on-device performance. We examined the models' performance in terms of average precision and recall, IOU, speed, and model size. When working on a Raspberry Pi, we discovered that EfficientDet1 after quantization, with a moving average decay of 0.95 and a Stochastic Gradient Descent optimizer is a good choice.**

***Index Terms*—Deep Learning, EfficientDet, EfficientNet, Raspberry Pi, Object Detection, Real-time, TensorFlow Lite**

## I. Introduction

In 2012, when AlexNet [1], won the ImageNet challenge, it marked the turning point in history for the field of computer vision and began the era of deep learning. There have been several deep learning based models proposed after that, till date. These models have successfully created a benchmark in the computer vision industry. However, when it comes to using these models in real-time applications, such as IoT, the question of whether we can use these huge models for applications with constrained resource arises. Most of the computer vision applications involve a camera. By 2030, the number of IoT devices is estimated to reach 125–500 billion, and assuming that 20% of them contain cameras, the market for IoT devices with cameras will be worth 13–100 billion units [2]. But the fundamental issue is that IoT devices are resource constrained, with limited memory and computational capability. The size of a deep learning-based model grows as the number of trainable parameter increases, resulting in an increase in inference time. This necessitates the need for more research in building models that work well on a constrained device.

Recently, the deep learning models like MobileNet [3] and EfficientDet [4] have gained popularity with respect to their applicability on constrained devices like mobile phones and IoT edges. The deep learning community has been thriving in providing the essential libraries, computational platforms, and assessment tools to make models compatible with a resource constrained device. However, there is still a lot of work to be done.

The major sub-field in computer vision is the object detection. Detecting an object with your phone or on an IoT edge is a luxury that has yet to be completely realized. Even-though recent models have kept their foot on this task, the precision they provide is simply too low to be acceptable in crucial situations where no errors can be tolerated. TensorFlow [5] is one of the communities that is providing major support for this task through its TensorFlow Lite Model Maker API. In this work, we try to bring a comparison of the latest EfficientDet family of object detectors when deployed for a specific task on a resource constrained environment like Raspberry Pi. We train the models in Google Colab and then convert each of them using the famous integer quantization technique and analyze their on-device real-time performances in terms of average precision, recall, size and frame rate. Our experiments show that EfficientDet1 with a moving average decay of 0.95 and an IOU of 0.5-0.95, together with the SGD optimizer, is an excellent solution for any real-time Raspberry Pi application for object detection, with minimal training data.

## II. Related Work

Recently in the field of object detection, more focus is being given to those architectures that are smaller in size and can be easily portable to any constrained devices like mobile phones and IoT edges. MobileNet [3] and MNasNet [6] are few popular models that attempt to obtain a high level of efficiency for deep learning on mobile devices. The EfficientNet architecture [7], is the new family of architectures proposed by Tan *et al.* in 2019, which used neural architecture search to form the baseline model and further used scaling up. In comparison to prior ConvNets, it claimed to have improved accuracy and efficiency. Scaling models typically use depth, width, or resolution scaling, but this model uses a compound scaling mechanism that scales in all three dimensions, allowing the model to contain fewer parameters than existing models and thus making it perfectly suitable for applications that require lighter models. The EfficientNet model family consisted of eight models ranging from B0 to B7, with each model number indicating a version with more parameters and higher accuracy.

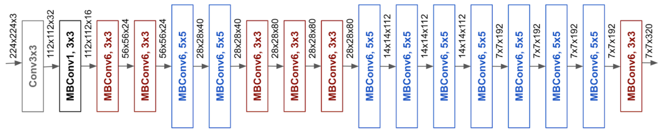

Fig. 1. The architecture for the baseline network EfficientNet-B0. Source: [9]

TABLE I
NUMBER OF PARAMETERS OF THE EFFICIENTNET FAMILY

| Model | Params |
|---|---|
| EfficientNet- B0 | 5.3M |
| EfficientNet- B1 | 7.8M |
| EfficientNet- B2 | 9.2M |
| EfficientNet- B3 | 12M |
| EfficientNet- B4 | 19M |
| EfficientNet- B5 | 30M |
| EfficientNet- B6 | 43M |
| EfficientNet- B7 | 66M |

The EfficientNet-B7 particularly, managed to achieve accuracy of 84.3 percent with just 66M parameters. The main block of the baseline architecture [7] consisted of MBConv [3], [6] and squeeze-and-excitation optimization [8].

Later in 2020, a weighted bidirectional feature network along with the customized compound scaling was used to propose the EfficientDet model [4]. This model has also gained popularity today due to its performance in the constrained environments. The use of this model in object detection and semantic segmentation tasks claimed to achieve state-of-the-art accuracy with fewer parameters. The next section describes this model family in brief.

## III. PRELIMINARIES

### A. The architecture and its background

EfficientNet is an architecture that works on the principle that model scaling does not change layer operators in a baseline network. Based on this the EfficientNet-B0 was developed using a neural architecture search such that it optimizes the accuracy as well as the FLOPS [7]. This architecture uses the same search space as in MNasNet [6] and hence is similar to it except that EfficientNet-B0 is slightly bigger in size. The detailed architecture of the baseline network is shown in Fig. 1.

The compound scaling method was then said to be applied on this baseline model to scale it up to form EfficientNet-B1-B7. Table I shows the number of parameters in each of these scaled up models.

The EfficientDet has the EfficientNet which is pre-trained on ImageNet as its backbone network. It additionally uses a BiFPN (efficient bi-directional cross-scale connections and weighted feature fusion) [4] as its feature network. This Bi-FPN is in charge of choosing feature P3-P7, as shown in Fig. 2, from the backbone network and applying top-down and bottom-up bidirectional feature fusion. To provide object class and bounding box predictions, these fused features are fed into

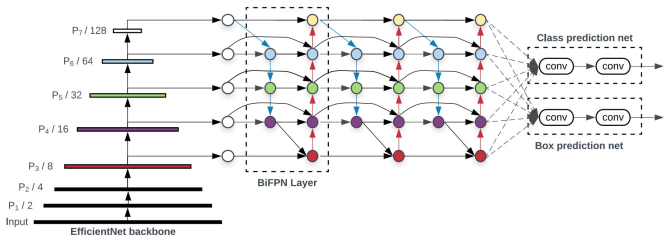

Fig. 2. EfficientDet Architecture with EfficientNet backbone [7] and BiFPN as feature network [4] and shared class/box prediction network. Source: [4]

TABLE II
CONFIGURATIONS OF THE EFFICIENTDET FAMILY

| Model | Input Size | Back-bone net-work | Bi-FPN #Chan-nels | Bi-FPN #Lay-ers | box/ class #Lay-ers | AP on COCO test-dev |
|---|---|---|---|---|---|---|
| EfficientDet0 | 512 | B0 | 64 | 3 | 3 | 33.8 |
| EfficientDet1 | 640 | B1 | 88 | 4 | 3 | 39.6 |
| EfficientDet2 | 768 | B2 | 112 | 5 | 3 | 43.0 |
| EfficientDet3 | 896 | B3 | 160 | 6 | 4 | 45.8 |
| EfficientDet4 | 1024 | B4 | 224 | 7 | 4 | 49.4 |
| EfficientDet5 | 1280 | B5 | 288 | 7 | 4 | 50.7 |
| EfficientDet6 | 1280 | B6 | 384 | 8 | 5 | 51.7 |
| EfficientDet7 | 1536 | B6 | 384 | 8 | 5 | 52.2 |

a class and box network respectively. The configurations of EfficientDet0-EfficientDet7 as given by Tan *et al.* is shown in Table II

EfficientDet models were mainly designed for the task of object detection. The EfficientDet models proposed by Tan *et al.* were trained and evaluated on the MS-COCO dataset [10], with 118K training images. With Stochastic Gradient Descent(SGD) optimizer and 0.9998 exponential moving average decay, swish activations were used.

### B. Integer Quantization

The growing popularity of deep learning models on mobile devices, as well as their high cost, has necessitated the development of more accurate and efficient inference approaches such as integer quantization [11]. Jacob *et al.* introduced a scheme in which both the activation and weights are quantized as 8 bit integers while keeping only a few bias vectors as 32-bit integers. On common hardware, it was shown how integer-only quantization can improve the tradeoff even further when dealing with MobileNet architecture. TensorFlow's newest libraries for transforming deep learning models into lighter versions that can be readily run on any constrained device, have used the integer quantization technique for model conversion and hence has led to the popularity of this technique. The formula used for approximation of floating point values for 8-bit integer quantization [5], [12] is as follows:

$$real\ value = \{int8\ value\ \text{-}\ zero\ point\}\ X\ scale \quad (1)$$

where, int8 two's complement numbers in the range (-127, 127), with zero-point equal to 0 are used to represent per-tensor/per-axis weights and int8 two's complement numbers in the range (-128, 127), with a zero-point in the range (-128, 128) are used to represent per-tensor activations/inputs. Per-axis quantization means that each slice in the quantized dimension will have one scale and/or zero point. The quantized dimension is the dimension of the tensor's shape that corresponds to the scales and zero-points. The models in this study are converted to a lighter version using this integer quantization methodology, and their performance on the constrained device is based only on this strategy, and the same may vary if any other strategy is chosen.

## C. *Moving Average Decay*

Maintaining moving averages of the trained parameters is typically advantageous while training a model. Averaged parameter evaluations occasionally give much better results than the final learned values. There are several libraries that allow us to tweak the decay value to apply for maintaining the training parameters' moving averages. The most used is the exponential moving average decay. In general, a moving average decay value close to 1.0 (such as 0.99) is a good choice.

## D. *Evaluation Metrics*

As specified earlier, the metrics based on which the models were evaluated in this study were the Average Precision and Recall (AP and AR), Intersection over Union(IOU) and the frame rate of the model in frames per second(FPS).

- **IOU:** Intersection Over Union- It finds difference between the ground truth annotations and predicted bounding box annotations. It is mainly used to set threshold for selecting the bounding boxes.

  *IOU= Area of Union / Area of Intersection*
- **Precision and Recall:** Precision is used to measure correct predictions while recall is used to calculate the true predictions from all correctly predicted data.
- **Frame rate:** The frame rate of a video is measured in frames per second (fps), which indicates how smoothly it runs on our computer. The more frames we can fit into a second, the smoother the animation on screen will be.

## IV. Experimental Setup

A platform like Raspberry Pi is a suitable place to start analyzing the performance of the EfficientDet family models in a limited context. As a result, in this research, we try to retrain the EfficientDet models on a small dataset, then run the converted light models on the device to compare and measure their performance. The goal of this research is to determine which model in the family provides the optimum trade-off between size, precision, recall, and frame rate when used in a resource-constrained, on-device, task-specific application. The overall methodology of this study is depicted in Fig. 3. Each model in the EfficientDet family was chosen one at a time and trained on the salads dataset before being tested on various test images. After training, each model was converted into a TensorFlow Lite model using integer quantization, and the converted Lite model was saved for future deployment. The converted models were then loaded onto a Raspberry Pi for real-time testing.

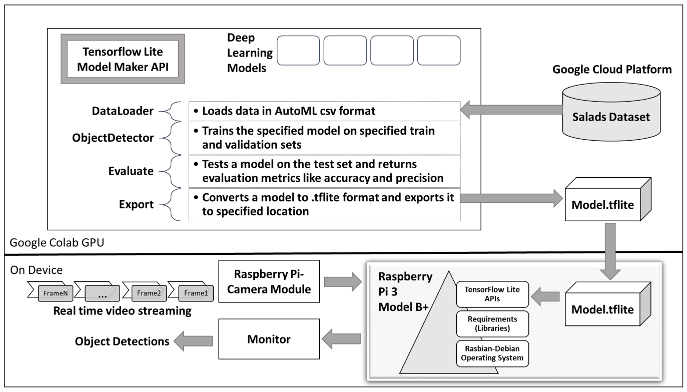
Tensorflow Lite Model Maker API
Deep Learning Models
Google Cloud Platform
DataLoader
• Loads data in AutoML csv format
Salads Dataset
ObjectDetector
• Trains the specified model on specified train and validation sets
Evaluate
• Tests a model on the test set and returns evaluation metrics like accuracy and precision
Export
• Converts a model to .tflite format and exports it to specified location
Model.tflite
Google Colab GPU
On Device
FrameN
...
Frame2
Frame1
Real time video streaming
Raspberry Pi-Camera Module
Raspberry Pi 3 Model B+
TensorFlow Lite APIs
Requirements (Libraries)
Rasbian-Debian Operating System
Model.tflite
Object Detections
Monitor

Fig. 3. Overall Methodology

## A. *Salads Dataset*

The Salads Dataset was used to train EfficientDet architectures. The Salads dataset is a popular publicly available dataset from Google [13]. This dataset was created using the OpenImages v4 dataset [14] and it consists of object images of five classes namely Baked goods, Cheese, Salad, Seafood and Tomato. The training set has 175 images, with 35 images per class. The validation and test set consists 25 images each. The choice of a small dataset ensures to critically analyze the performance of the EfficientDet architectures when used with any other small custom dataset for task specific applications.

## B. *TensorFlow Lite*

The EfficientDet models used in the experiments were imported from the TensorFlow [5]. TensorFlow provides a popular group of libraries that support applications on deep learning by providing several tools, models as well as data-loaders. The TensorFlow Lite Model Maker library provides a set of architectures on EfficientDet which are pretrained on the MS-COCO dataset [10], which is easily convertible to .tflite format. Tooling and kernels for int8 quantization for 8-bit are prioritized in TensorFlow Lite quantization. This is said to be necessary for the simplicity of representing symmetric quantization as a zero-point equal to 0.

The Google Colab GPU was used to train the models in this experiment, and the resulting Lite models were then ported to a Raspberry Pi for performance study. The trained models were also tested and the required metrics were recorded to perform the evaluation.

## C. *Raspberry Pi with Camera module*

For performing on device evaluation of the models, we use the Raspberry Pi 3 Model B+ with the Pi-Camera module as shown in Fig. 4. The Raspbian/Debian buster operating

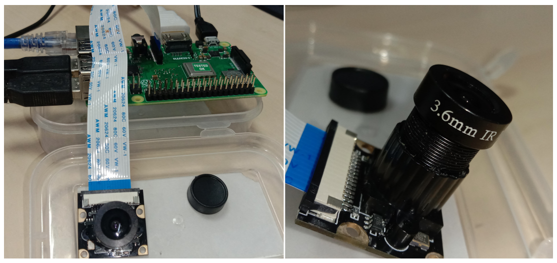

Fig. 4. Raspberry Pi 3 with Pi-Camera Module

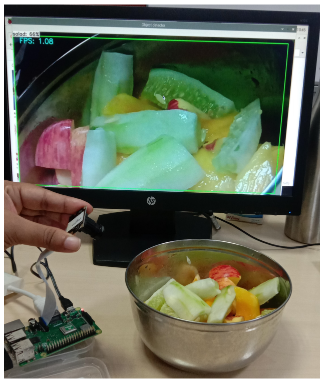

Fig. 5. Testing phase of tflite models on Raspberry Pi 3 with Pi-Camera Module

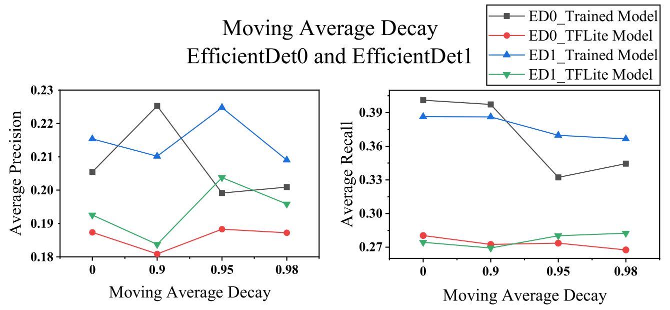

Fig. 6. Comparison of Average Precision and Average Recall of EfficientDet0 and EfficientDet1 for different decay values

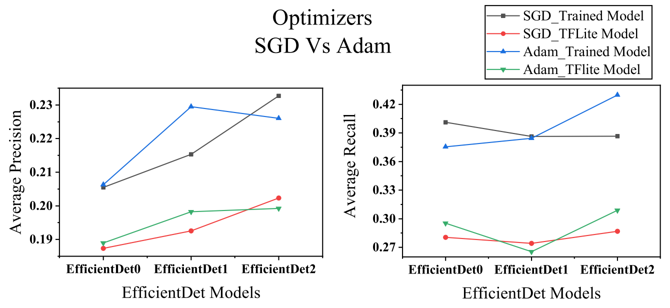

Fig. 7. Comparison of Average Precision and Average Recall from the Stochastic Gradient Descent and Adam Optimizer on EfficientDet models.

system and distribution were installed on the device, which had a 32-bit armv7l architecture. The Python version used was 3.7.3, and the opencv-contrib-python version was 4.1.0.25. The library requirements for running the TensorFlow Lite model on Raspberry Pi is provided by the EdjeElectronics and is available at this link [15]. After the libraries have been installed, the TensorFlow Lite Github repo must be cloned on top of it, and built, in order to execute the models saved in .tflite format [16]. Fig. 5 shows a sample snapshot of the testing phase on Pi.

## V. Analysis of Performance

The integer quantization reduces the size of the model but at the cost of reduced accuracy. As a result, as demonstrated in Fig. 6, the converted models have reduced precision and recall. For the EfficientDet0 and EfficientDet1 models with varying moving average decays, the plots show the average precision and recall. Evidently, EfficientDet0 performs the best at the moving average decay of 0.9 whereas in rest of the cases, EfficientDet1 gives better performance. Furthermore, the quantized EfficientDet1 outperformed the non-quantized EfficientDet0, when the moving average decay was 0.95, when observing the average precision. These findings leads to the conclusion that a moving average decay of 0.95, if chosen, can improve the overall performance of the EfficientDet1 model.

In terms of optimizers, we looked at the precision and recall provided by EfficientDet0 through EfficientDet2 for two of the most commonly used optimizers: the Stochastic Gradient Descent(SGD) and the Adam optimizer. The observed values of average precision and recall for the three distinct models are depicted in Fig. 7. When it comes to quantization, the EfficientDet0 model performs well on the Adam optimizer, while EfficientDet2 with Adam has the best performance. In comparison to Adam optimizer, only EfficientDet1 provides superior performance using SGD. This is because, despite having a high precision, it has the lowest recall after quantization. We stopped at EfficientDet2 for Adam since the next models in the family had poor performance in terms of execution time and were not suitable for usage on our constrained device.

Nonetheless, we used moving average decay of zero to train EfficientDet3 and EfficientDet4 models for SGD. Because the Google Colab runs out of memory when using batch size of eight, we decided to train them in batches of two. The Fig. 8 depicts the performance of all five models for SGD with a moving average decay of zero.

Fig. 9 shows the FPS (frames per second) of the five converted models while running on our Raspberry Pi. Table III shows the size of the converted tflite models after quantization. Fig. 11 shows the plot of loss curves for all five models during the training, for 50 epochs, using the SGD optimizer.

The performance in terms of time, keeps on degrading as we move higher in the family of the architectures. The greater accuracy in terms of precision and recall gets ruled out from the bad performance in time and frame rates, which is another reason to stop at EfficientDet4. Moreover, as we progress higher in the family, the models start to over-fit and depict very

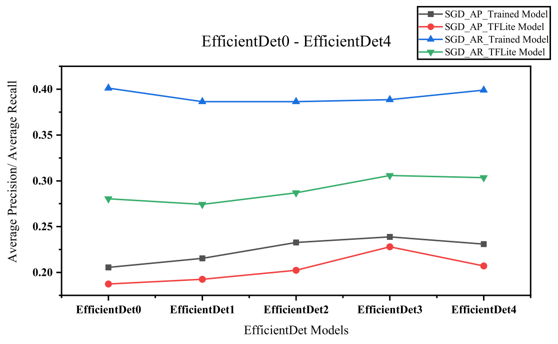

Fig. 8. Average Precision and Average Recall of EfficientDet0 -4

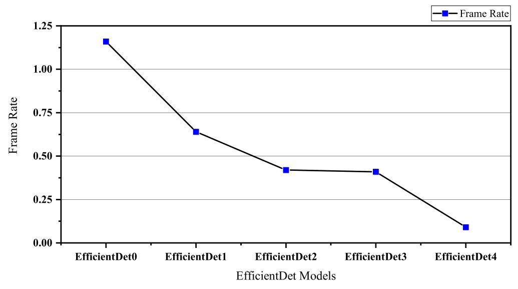

Fig. 9. Models' performance on Raspberry Pi in FPS

unstable behaviour which is visible in Fig. 11. As a result, we only advise the EfficientDet0 or EfficientDet1 models as safer options.

After analyzing the models in terms of accuracy and speed, we focus on another factor called IOU to ensure the models' capability to avoid false positives and false negatives. Giving lower threshold values, according to IOU, improves the model's precision and reduces false positives. However, as seen in Fig. 10, overly low threshold values can lead to more false negatives. The best choice of IOU may be determined by the work at hand and cannot be generalised. With an IOU in the range of 0.5-0.95, EfficientDet models operate pretty well, with few false positives.

## VI. Conclusion and Future Work

Using the Colab-GPU we trained the EfficientDet family of architectures up to EfficientDet4, and the higher models were not considered. If more GPU resources are available for training, the models from EfficientDet5-7 can be evaluated

TABLE III
SIZE OF THE EFFICIENTDET MODELS AFTER CONVERSION

| Model | Size (MB) |
|---|---|
| EfficientDet0 | 4.2 |
| EfficientDet1 | 5.6 |
| EfficientDet2 | 7.0 |
| EfficientDet3 | 7.0 |
| EfficientDet4 | 19.6 |

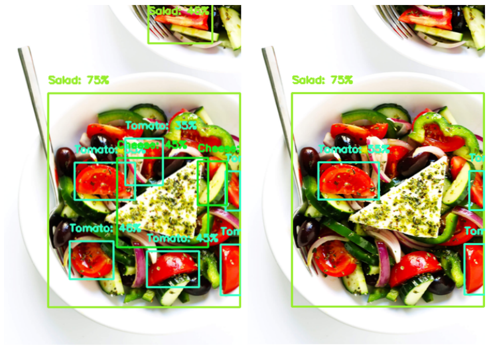

Fig. 10. Comparison of output of model with different IOU. [left: IOU=0.95; right: IOU=0.5]

in the future. However, we do not recommend models higher than EfficientDet4 because the performance graph decreased as we progressed along the family. However, this study reminds readers that while developing constrained applications for object detection on Raspberry Pi using EfficientDet, it is always better to stick to EfficientDet0 or EfficientDet1.

Even though EfficientDet0 has the best frame rate, as evidenced by its modest size, we recommend EfficientDet1 with a moving average decay of 0.95 as a better choice when the speed is not crucial. Also when it comes to EfficientDet1, it is always optimal to use the SGD optimizer. If the application requires better speeds, EfficientDet0 with a 0.9 moving average decay and Adam optimizer is a solid option. In both circumstances, the needed IOU may range from 0.5 to 0.95, with 0.95 being overly specific in detection.

This effort is limited to the Raspberry Pi 3, and the trained EfficientDet models may or may not yield different when tested on other resource constrained devices, such as a mobile phone, and the analysis of that will be undertaken as a separate project from this work.

## References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems, vol. 2, 2012, pp. 1097–1105.
[2] A. Mohan, K. Gauen, Y.-H. Lu, W. Li, and X. Chen, "Internet of video things in 2030: A world with many cameras," 2017.
[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2018, pp. 4510–4520.
[4] M. Tan, R. Pang, and Q. Le, "Efficientdet: Scalable and efficient object detection," in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2020, pp. 10 778–10 787.
[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen et al. , "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/
[6] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2019-June, 2019, pp. 2815–2823.
[7] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in 36th International Conference on Machine Learning, ICML 2019, vol. 2019- June, 2019, pp. 10 691–10 700.
[8] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7132–7141.

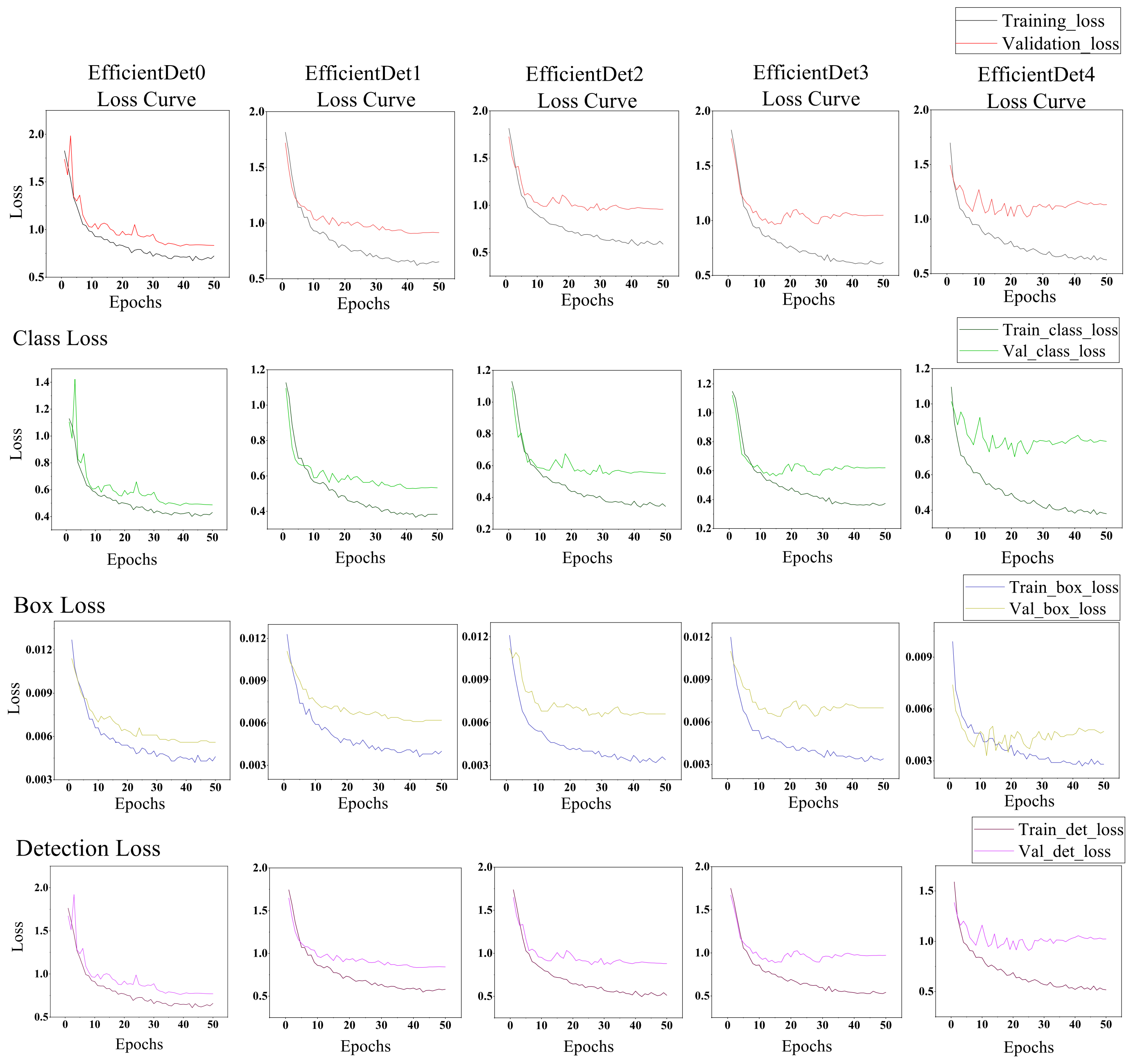

Fig. 11. EfficientDet- Loss curves during training over the Salads Dataset

[9] M. Tan, "Efficientnet: Improving accuracy and efficiency through automl and model scaling," May 2019. [Online]. Available: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

[10] T.Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Doll´ar, and C. Zitnick, "Microsoft coco: Common objects in context," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8693 LNCS, no. PART 5, pp. 740–755, 2014, cited By 6876.

[11] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer arithmetic- only inference," 2018, pp. 2704–2713.

[12] "Tensorflow lite 8-bit quantization specification." [Online]. Available: "https://www.tensorflow.org/lite/performance/quantization spec-specification summary"

[13] "Salads dataset," Offered by Google Cloud Vision API. [Online]. Available: 'gs://cloud-ml-data/img/openimage/csv/salads ml use.csv'

[14] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," International Journal of Computer Vision, vol. 128, no. 7, pp. 1956–1981, 2020.

[15] Website: "Edjeelectronics/ tensorflow-lite-object-detection-on-android-and-raspberry-pi," Dec 2020. [Online]. Available: "https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master"

[16] Website: "tensorflow/tensorflow/lite," Aug 2021. [Online]. Available: "https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite"