An Edge-based Real-Time Object Detection

Ali Ahmadinia* Jaabaal Shah
Department of Computer Science and Information Systems
California State University San Marcos
San Marcos, CA, USA
*Corresponding author: aahmadinia@csusm.edu

***Abstract*—This paper looks at performance bottlenecks of real-time object detection on edge devices. The "You only look once v4" (YOLOv4) is currently one of the leading state-of-the-art models for real-time object detection, and its tiny version: YOLOv4-tiny, is designed for edge devices. To improve object detection accuracy without sacrificing detection speed, we propose an object detection method based on YOLOv4-tiny and VGG-Net. First, we implement the mosaic data augmentation and Mish activation function to increase the generalization ability of the proposed model, making it more robust. Secondly, to enhance the richness of the features extracted, an extra 3x3 convolution layer is added in a way that two successive 3x3 convolutions are used to obtain 5x5 receptive fields. This would enable us to extract global features in the first CSP (Cross Stage Partial Network) Block and restructure the connections of the subsequent layers to have the same effect on the next CSP blocks. Evaluation results show that the proposed model has comparable performance and memory footprint but significantly greater accuracy than YOLOv4-tiny. Also, the proposed tiny model has similar performance to YOLOv4-tiny, and improves accuracy with much lower memory overhead, which makes it an ideal solution for real-time object detection, especially on edge devices.**

Keywords- Object Detection, YOLOv4-tiny, Edge-ML

I. Introduction

With the advent of Artificial Neural Networks with highly accurate inference capabilities for the myriad real-world complex problems, there has been an exciting amalgamation of IoT and Machine Intelligence. This has brought forth the intuition of "intelligent Edge" and "TinyML," which has been made possible through the innovations in the semi-conductor domain by creating powerful embedded systems which provide comparable performance to various CPUs with much lower overhead of resources. This research work explores and analyzes Object Detection at the edge, focusing primarily on evaluating the performance of the current state-of-the-art YOLOv4-tiny model on the Jetson Nano Developer Kit and improving the performance and accuracy of this model. To the best of our knowledge, this would be a one-of-a-kind comprehensive study involving edge detection pertaining to this specific use-case as there have been numerous research papers published for mask detection using state-of-the-art models; however, they focus primarily on detection using desktop GPUs such as Nvidia RTX 2080, but none of them are dedicated to the edge devices which provide a low-cost and low-power solution and also solves the security and network bandwidth concerns of centralized solution in IoT frameworks.

The contributions of this paper include (1) Analysis of the architecture of the YOLOv4 and YOLOv4-tiny models. (2) Implementation and comparison of the performance and accuracy of the VGG-16 inspired custom models with YOLOv4-tiny on the desktop and embedded environments using a smaller and a larger dataset (3) Proposing a new model with significantly improved accuracy and a negligible memory overhead without degrading the performance of the model.

II. Related Work

In [1], the authors have proposed a fast object detection method based on YOLOv4-tiny. Their model replaces the first two CSPBlock (Cross Stage Partial Network Block) modules with the ResBlock-D (ResNet Block) modules to reduce the computation complexity and introduces an auxiliary residual network block to extract more feature information and minimize the detection error. The authors in [2] implement a proposed method that shows it is suitable to set up a surveillance system in smart cities for people detection, social distancing classification, and body temperature analysis. [3] makes use of a pruned version of YOLOv4 to address the threat of drones intruding into high-security areas and also incorporates small object augmentation to achieve better accuracy.

At present, there are multiple mask detection models which have achieved very high accuracy, such as [4], [5]. Furthermore, there have also been studies on real-time face mask detectors that can be deployed on edge devices such as [6], [7]. However, most of these models have been tested only on the desktop and use custom datasets. Table I provides a comprehensive summary of the FPS (Frame Per Second) and mAP[1] (mean Average Precision) achieved by various edge-based models. It can be seen that the dataset plays a very vital role and also for embedded devices, if a model achieves anywhere between 20 to 30 FPS, it is considered as a good enough performance.

III. Analysis of YOLOv4-Tiny

In this paper, we analyze existing edge-based object detection models, particularly the architecture of YOLOv4-tiny and explore various parameters to design an optimal network structure for object detection on embedded devices. State-of-the-art models such as PAN-Net[9], SPP-Net[10], CSP-Net[11] and VGG-Net[12] have been instrumental in the analysis and design of the proposed model architecture.

YOLOv4-tiny method is designed based on the YOLOv4 technique to compress the network for less computational

[1]mAP@50 is the mAP calculated at IOU (Intersection over Union) threshold of 0.5.

978-1-6654-6283-9/22/$31.00 ©2022 IEEE
DOI 10.1109/ICMLA55696.2022.00075

TABLE I: The performance and accuracy metrics achieved by various edge-based models.

Model	Dataset	Hardware	FPS	mAP@50
1NN[6]	mask custom	Edge TPU	N/A	58.4%
2NN[6]	mask custom	Edge TPU	N/A	54.8%
YOLOv3-Slim[7]	mask custom	Nvidia GTX 1060	18.19	87.17%
Pruned YOLO[3]	drones custom	N/A	69	90.5%
Improved YOLO[1]	MS COCO	Raspberry Pi	0.31	38%
YOLOv4-tiny[8]]	MS COCO	Jetson Nano	16	22%
YOLOv4-tiny(TRT)[8]	MS COCO	Jetson Nano	39	22%

overhead and hence faster process. YOLOv4-tiny method uses a feature pyramid network to extract feature maps with different scales to increase object detection speed without using the SPP and PAN used in YOLOv4. In addition, the tiny method uses two different scale feature maps that are 13×13 and 26×26, to predict the detection results and to further simplify the computation process, YOLOv4-tiny method uses the LeakyReLU function as an activation function in CSPDarknet53-tiny network without using the Mish activation function that used in YOLOv4.

A. Implementation

The YOLOv4-tiny configuration has been implemented natively in the Darknet[17] framework. Darknet is an open-source neural network framework written in C and CUDA. Considering that there is a very limited dataset for mask detection, we adopt a transfer learning approach by using MS COCO for training, which is well labeled and contains a large amount of data. This approach has several advantages, including saving training time, increasing the performance of the neural network. and not requiring extensive training data.

We conduct experiments mainly on a mask dataset [13] consisting of 848 annotated images belonging to 3 different classes, namely "with mask," "mask is worn incorrectly," and "without a mask" for 566 epochs. The dataset has been split into training and testing sets per the Darknet YOLO format requirements and has an 80-20 train-test split. In addition, we have conducted a few experiments on the PASCAL VOC 2012 dataset [14] consisting of 17112 annotated images from 20 different classes of everyday objects available in the Darknet YOLO format from Roboflow for approximately 560 epochs with the dataset split into an 80-20 train-test split.

The experiments conducted have been split into four different categories stated below:

- Based on the implementation of Mosaic data augmentation i.e., whether mosaic data augmentation technique is used or not.
- Based on whether TensorRT [2] inference optimizer is used or not on the embedded device.
- Based on activation function (Leaky ReLU or Mish)

[2]TensorRT is a machine learning framework that is published by Nvidia to run inference on Nvidia GPUs.

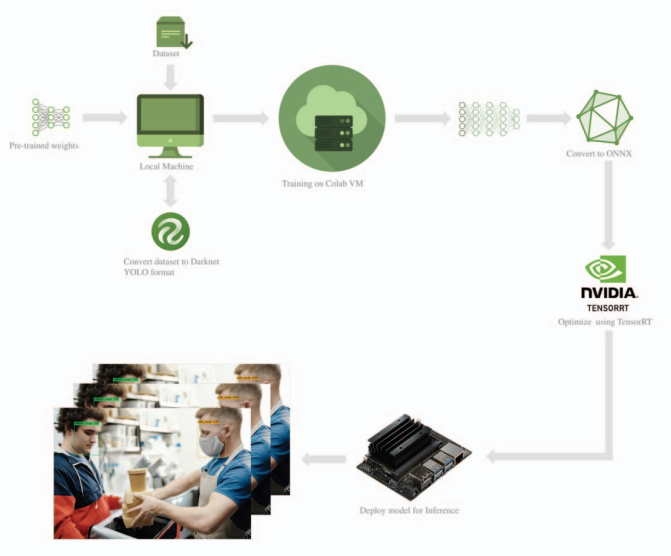

Fig. 1: System Workflow

- Based on the architecture, i.e., CSPDarknet53-tiny or custom architecture based on VGG-Net.

We used Google Colaboratory as the tool for training the model. The pretrained YOLOv4-tiny weights of the MS-COCO dataset are used as the initial weights for training that leads to faster convergence. We use Tesla K80, and Tesla P100 graphical processing units available on Colaboratory. We also implemented on Nvidia Jetson Nano Development Kit with the Nvidia JetPack 4.5.1 for object detection on an edge device.

We have conducted the tests for two different video resolutions (1080p and 360p) and have also divided the results based on Tesla K80 for the desktop, and for the testing on an embedded system, we use Tegra X1 GPU that comes with Jetson Nano Developer Kit System-on-Module(SoM) [16].

B. Impact of Image Size

Inferences from the experiments conducted using the YOLOv4-tiny configuration with respect to the input image size and different platforms have been summarized in Figures 2, and 3.

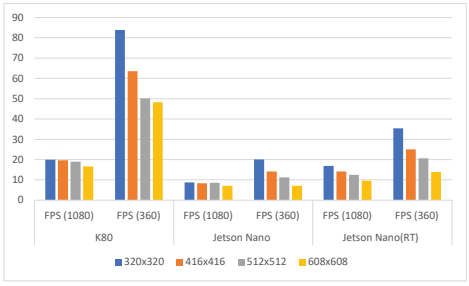

Fig. 2: Performance of YOLOv4-tiny on different platforms and various image sizes

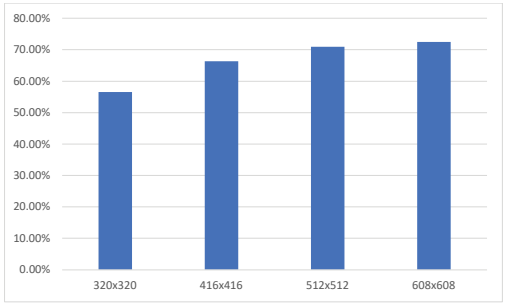

Fig. 3: Accuracy of YOLOv4-tiny with various image sizes (mAP@50)

The results in Figures 2, and 3 clearly show that irrespective of the GPU, with the increase in the input image size, the performance of the YOLOv4-tiny model decreases in most cases; however, there is a consistent increase in the accuracy. There is approximately a 25% increase in the accuracy from input image size of 320 pixels to 512 pixels. And there is a decrease of approximately 20% and 70% in performance in the worst case for 1080p and 360p video resolution, respectively.

C. Impact of Activation Function and Data Augmentation

Figures 4, and 5 show the ablation study of the YOLOv4-tiny model with input image size of 416x416.

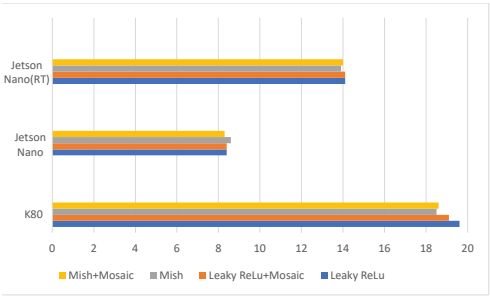

Fig. 4: Performance of YOLOv4-tiny on different platforms and various data augmentation

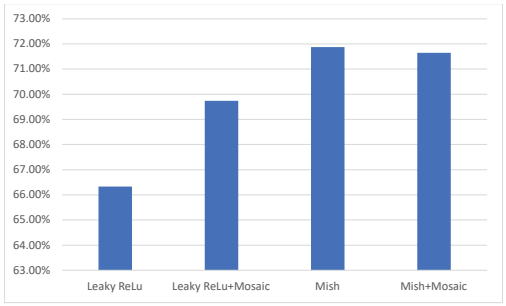

Fig. 5: Accuracy of YOLOv4-tiny with various data augmentation (mAP@50)

Figures 4, and 5 clearly show that irrespective of the GPU, adding the mosaic data augmentation provides similar performance to Leaky ReLU and improves the performance of the model in most cases for the lower resolution videos with an increase of approximately 5% in the accuracy. However, in the case of high-resolution video streams, the performance is worse than or similar to its Leaky ReLU counterpart. In the case of mixing the mosaic data augmentation with the Mish activation function, we achieve the optimal results out of all the combinations, with performance being at par with its Leaky ReLU counterpart yet still achieving approximately 8% increase in the accuracy. In addition, we also observe that in the case of replacing the activation function by Mish, we see an increase of approximately 9% in the accuracy with a very similar performance as compared to the Leaky ReLU counterpart.

D. YOLOv4-tiny vs. VGG-based architectures

Figures 6, and 7 show the results of the custom models inspired by the VGG-16 architecture with an input image size of 512x512, Mish activation function, and mosaic data augmentation. VGG Custom-1, VGG Custom-2, and VGG Custom-3 refer to the custom models inspired by VGG-16 with 16, 18, and 24 layers, respectively.

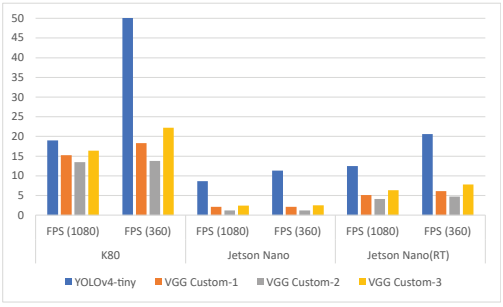

Fig. 6: Performance of YOLOv4-tiny vs. VGG-based custom architectures on different platforms

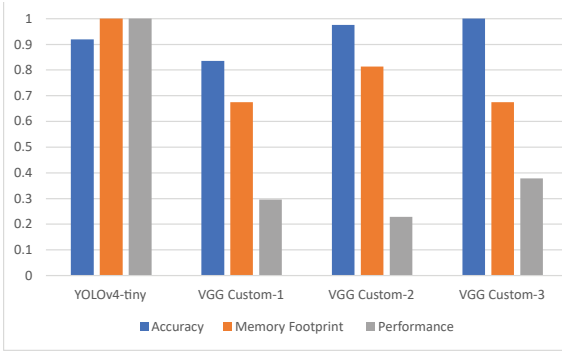

Fig. 7: YOLOv4-tiny vs. VGG-based custom architectures in terms of normalized Accuracy, Memory Footprint, and Performance on Jetson Nano using TensorRT

The results summarized in Figures 6, and 7 clearly show that the YOLOv4-tiny model with an input image size of 512x512 and the Leaky ReLU activation function achieves 3x times

better performance irrespective of whether we implement a quantized model with Nvidia TensorRT or not. However, the custom VGG models 2 and 3 achieve approximately 6% and 9% better accuracy than the YOLOv4- tiny model.
In addition, the results also show that as the number of layers increases, so does the model size, and therefore the YOLOv4-tiny model with 38 layers has the largest model size. In addition to that, we can also see that the size of filters or the depth of the convolution layer plays an important role as well as the VGG Custom-1 model with 16 layers has the same model size as the VGG Custom-3 model with 24 layers due to the difference in the convolution filter sizes where VGG Custom-1 has larger convolution filter sizes, and VGG Custom-3 has smaller convolution filter sizes as compared to VGG Custom-1.

Figure 8 compares the total number of parameters (in Millions) generated by the YOLOv4-tiny model with an input image size of 512x512 and the Leaky ReLU activation function and custom VGG-16 models. The results show that the YOLOv4-tiny model outperforms the custom models by occupying at least 3x less memory and thus achieving faster inference.

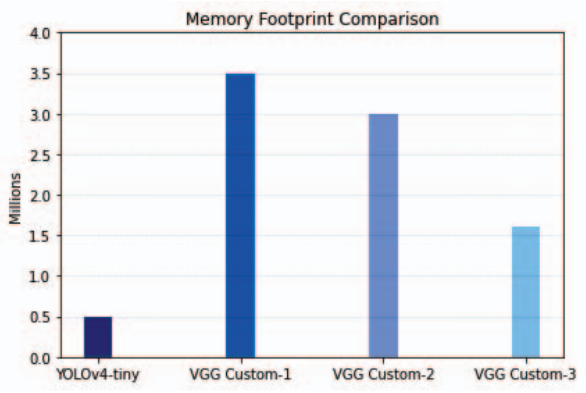

Fig. 8: Comparison between the total number of parameters (in Millions) of YOLOv4-tiny and custom models.

Interestingly, despite having more layers in the VGG custom 3 model, the number of parameters is approximately half of the other two custom models, i.e., VGG custom 1, and VGG custom 2. We can use a similar approach at the smaller depth of the convolutions used in the VGG custom 3 model, which leads to fewer parameters and, therefore, a much faster inference than its counterparts. In addition, these models achieve greater accuracy than the YOLOv4-tiny model due to the stacking of the convolution layers, which increases the receptive field of the model [12]. For example, if two 3 x 3 convolution layers (with padding 1, and stride 1) are stacked one after the other, the receptive field becomes 5 x 5, and if three 3 x 3 convolution layers (with padding 1, and stride 1) are stacked one after the other, the receptive field becomes 7 x 7 and the total number of parameters are fewer than one 7 x 7 convolution layer hence making it computationally more efficient.

To further analyze the effect of the convolution depth or the number of filters on the model, we also designed the VGG custom 4 model with 28 layers. Table II summarizes the comparison of the mAP@50 results between the VGG custom 3 and VGG custom 4 models on the Nvidia Tesla P100 GPU. The results highlight two compelling observations, the first being that the variants of the VGG custom 3 and VGG custom 4 with maximum convolution filter size (depth) 128 achieve superior results than their counterpart with maximum convolution filter size (depth) 256 in both the cases and the second being that the models with maximum convolution filter size 128 have at least a 2.7x lower model size than its counterparts with maximum convolution filter size 256.

TABLE II: Filter Size - Layers Trade off on Nvidia Tesla P100 using mask dataset.

Filter Size	Layers	FPS (1080p)	FPS (360p)	mAP@50	Model Size
128	24	42.8	164.2	78.38%	5.4 MB
256	24	42.4	139.9	77.25%	15.2 MB
128	28	43.4	111.02	82.07%	5.7 MB
256	28	42.2	98	80.41%	15.6 MB

We conduct similar experiments on the PASCAL VOC 2012 dataset to verify our observation that smaller filter size variants achieve superior results irrespective of the dataset. We designed the VGG custom 5 model with 32 layers and maximum convolution filter size (depth) 128 and also designed a variant of the VGG custom 4 model with maximum convolution filter size (depth) 64 for a more robust comparison. Table III summarizes the results achieved by training each model for approximately 187 epochs on the Nvidia Tesla P100 GPU.

Filter Size	Layers	mAP@50	Model Size
128	24	13.67%	5.4 MB
256	24	14.42%	15.3 MB
64	28	11.06%	2 MB
128	28	14.24%	5.7 MB
256	28	15.94%	15.7 MB
128	32	17.46%	7 MB

TABLE III: Filter Size - Layers Trade off on Nvidia Tesla P100 using Pascal VoC 2012 dataset.

It clearly shows that as the number of layers increases, so does the model's accuracy. However, there is not a massive difference between the accuracy of the VGG custom 3 and the VGG custom 4 model, and this slight increase in the accuracy is achieved at the cost of increasing the model size by at least 2.7x times which is a significant compromise in the case of resource-constrained devices. We also infer from the results of Table III that model size and filter size are inversely proportional, and the reduction in the maximum filter size (depth) does not significantly affect the accuracy.

E. Proposed Models

Considering our experiments' results, our optimal model could be a combination of the model size and performance of

YOLOv4-tiny, and the accuracy of the VGG inspired models. For the proposed model, we modify the existing YOLOv4-tiny architecture by adding a convolution of 32 filters in the first CSP computational block and rerouting the layers such that, in the subsequent CSP blocks, each of the 3 x 3 convolutions is repeated twice, just like the first CSP block. This would enhance the features extracted, thereby increasing the number of parameters by a very minimal number. However, this increase is far outweighed by the accuracy achieved by our proposed model, as shown in Figure 10.

We have also designed a tiny version of our proposed model, which has the same architecture as the proposed model. However, the depth of each convolution layer has been reduced by half, i.e., the convolution layer filter size of 64 in the proposed model has been modified to 32. This tiny model has at least a 4x times smaller model size than the proposed model and the YOLOv4-tiny model. In addition to that, the proposed-tiny model has an accuracy of approximately 2% better than the YOLOv4-tiny model, which makes it a very promising model for resource-constrained edge devices; however, the proposed-tiny model has approximately 11% less accuracy than the proposed model. Figures 9 and 10 summarize the FPS and mAP comparison between the original YOLOv4-tiny model, proposed model, and proposed-tiny model on the various GPU architectures for an input image size of 512x512 for the mask detection dataset.

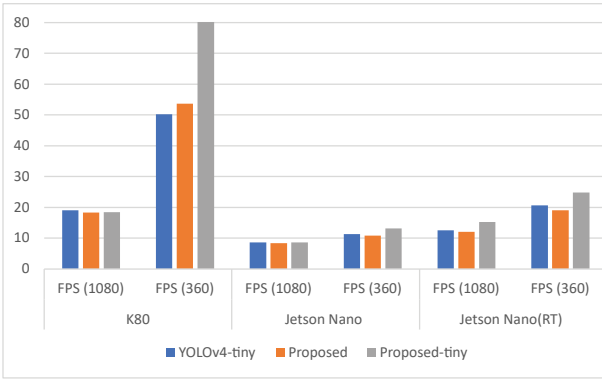

Fig. 9: Performance of the proposed models vs. YOLOv4-tiny on different platforms using mask dataset

Figure 11 and Figure 12 show the differences in the architecture of the CSP (Cross-Stage Partial) block of the YOLOv4-tiny model and the proposed model. Figure 11 depicts the architecture of YOLOv4-tiny model [8] and Figure 12 shows the architecture of the proposed model where k is the convolution kernel size, and b and g refer to the filter sizes.

Our proposed model achieves greater accuracy compared to the YOLOv4-tiny as shown in Figure 10. It improves the accuracy by about 9% and 16% for input image sizes of 416x416 and 512x512, respectively. In addition, the extra layer adds only 0.1 MB to the existing YOLOv4-tiny model size to get the model size of 22.6 MB.

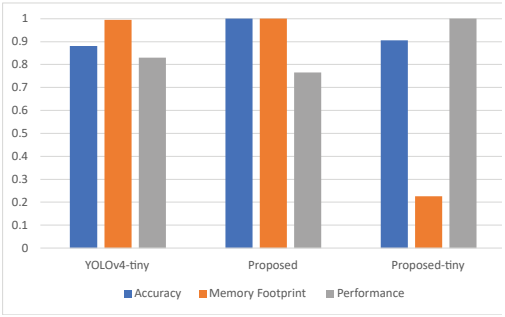

Fig. 10: The proposed models vs. YOLOv4-tiny in terms of normalized Accuracy, Memory Footprint, and Performance on Jetson Nano using TensorRT based on mask dataset

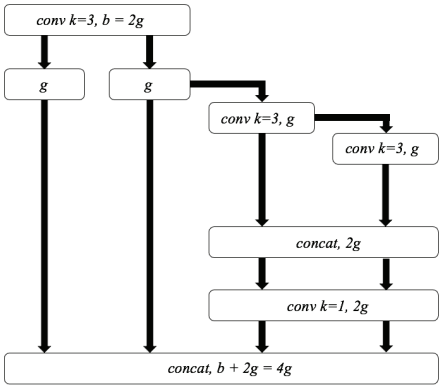

Fig. 11: Architecture of the CSP (Cross-Stage Partial) block in YOLOv4-tiny [8].

The results of Figure 9 assert that the performance of the proposed model is at par with its YOLOv4-tiny counterpart and slightly better in some instances, however considering its performance specifically on the embedded device, we can say that it is almost identical. The proposed model performs worse in some cases because of the extra computations for the additional convolution layer. However, according to the standards of object detection, that degradation is negligible.

Figure 9 shows that the performance of the YOLOv4-tiny model is better than the proposed and proposed-tiny model only on the 1080p video streams for the Mask Detection dataset. In all other cases, the proposed-tiny model outperforms both the YOLOv4-tiny and the proposed model on the embedded device. In addition, the size of the proposed-tiny model is approximately 4.5x times smaller than the YOLOv4-tiny, and the proposed-tiny model with a 1% increase in accuracy over the YOLOv4-tiny model and a 9% decrease in accuracy over the proposed model proving that our proposed models are better than the existing YOLOv4-tiny model as shown in Figure 10.

For further comparison with a larger dataset, we have also trained the YOLOv4 model on the PASCAL VOC 2012 dataset. Figures 13 and 14 summarize the FPS and mAP

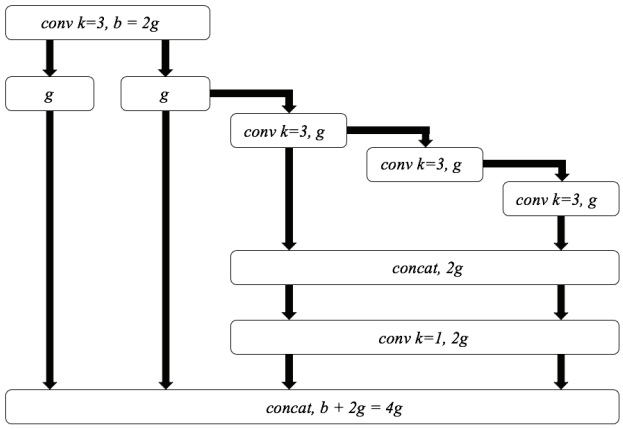

Fig. 12: Architecture of the CSP (Cross-Stage Partial) block in Proposed model.

comparison between the original YOLOv4-tiny model, the proposed model, and the proposed-tiny model on the various GPU architectures for an input image size of 512x512 for the PASCAL VOC 2012 dataset.

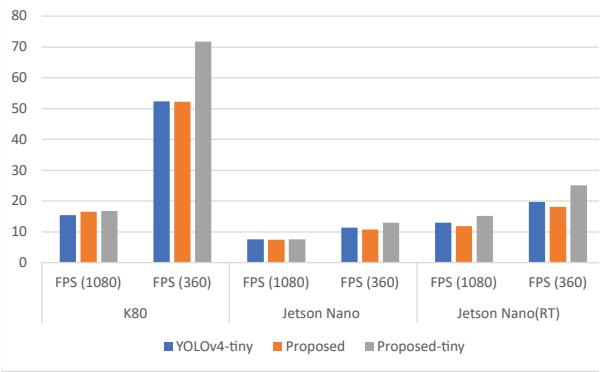

Fig. 13: Performance of the proposed models vs. YOLOv4-tiny on different platforms using PASCAL VOC 2012 dataset

As can be seen in Figures 14, and 13, the YOLOv4 model has approximately 34% and 46% greater accuracy than the proposed model, and the proposed-tiny model, respectively. However, the model size of the YOLOv4 model is approximately 10x times and 50x times greater than the proposed model and the proposed-tiny model, respectively, which further clarifies the trade-off between accuracy and model size. In addition, the FPS of the YOLOv4 model is at least 1.6x times less than the proposed model and the proposed-tiny model.

IV. Conclusion

The proposed models provide a better and more efficient solution to the current problem of mask detection using edge devices or resource-constrained embedded devices, which has real-world applications not limited to object detection. Our proposed model achieves at least 10% higher accuracy, and the

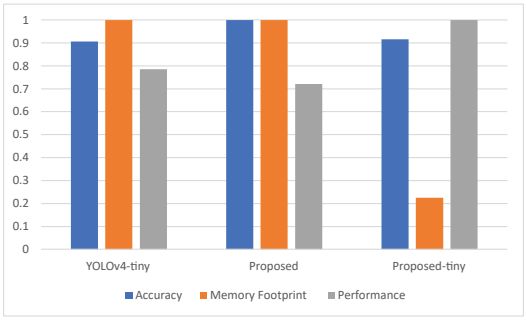

Fig. 14: The proposed models vs. YOLOv4-tiny in terms of normalized Accuracy, Memory Footprint, and Performance on Jetson Nano using TensorRT based on PASCAL VOC 2012 dataset

proposed-tiny model achieves at least 2% higher accuracy than YOLOv4-tiny model for real-time object detection on embedded devices. The proposed model's performance (frame per second) and memory footprint are comparable to YOLOv4-tiny. The tiny proposed model reduces the memory footprint by 70% in comparison to YOLOv4-tiny. Leveraging the CSPized implementation of YOLOv4-tiny model allows us to achieve this higher accuracy, and adjusting filter sizes reduces the memory footprint without a significant performance trade-off.

References

[1] Zicong Jiang et al, "Real-time object detection method for embedded devices," arXiv:2011.04244, 2020.
[2] Sergio Saponara et al, "Implementing a real-time, AI-based, people detection and social distancing measuring system for Covid-19," in Journal of Real-Time Image Processing, 2021.
[3] Hansen Liu et al, "Real-Time Small Drones Detection Based on Pruned YOLOv4," in Sensors, 2021.
[4] Preeti Nagrath et al, "SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2," in Sustainable Cities and Society,Vol. 66,2021,102692.
[5] Jimin Yu and Wei Zhang, "Face Mask Wearing Detection Algorithm Based on Improved YOLO-v4," in Sensors 2021, 21,3263.
[6] Keondo Park et al, "Real-Time Mask Detection on Google Edge TPU,". arXiv preprint:2010.04427v1, 2020.
[7] Xiaoming Jiang et al, "YOLOv3-Slim for Face Mask Recognition," in J. Phys.: Conf. Ser. 1771 012002, 2021.
[8] Chien-Yao Wang et al, "Scaled-YOLOv4: Scaling Cross Stage Partial Network," arXiv:2011.08036, 2021.
[9] Shu Liu et al, "Path Aggregation Network for Instance Segmentation," arXiv:1803.01534v4, 2018.
[10] Kaiming He et al, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," arXiv:1406.4729v4, 2015.
[11] Chien-Yao Wang et al, "CSPNET: A New Backbone That Can Enhance Learning Capability of CNN," arXiv preprint:1911.11929v1, 2019.
[12] Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556v6, 2015.
[13] Larxel, Face Mask Detection, https://www.kaggle.com/andrewmvd/face-mask-detection, 2020.
[14] Mark Everingham et al, The Pascal Visual Object Classes (VOC) Challenge, http://host.robots.ox.ac.uk/pascal/VOC/.
[15] Tsung-Yi Lin et al, "Feature Pyramid Networks for Object Detection," arXiv:1612.03144v2, 2017.
[16] Alexey Bochkovskiy et al, "YOLOv4: Optimal Speed and Accuracy of Object Detection," arXiv:2004.10934, 2020.
[17] Joseph Redmon, "Darknet: Open Source Neural Networks in C", http://pjreddie.com/darknet/, 2022.